Theory and Methodology

Solving a class of network models for dynamic flow control

Malachy Carey  
*Faculty of Business and Management, University of Ulster, N. Ireland, UK, BT37 0QB*

Ashok Srinivasan  
*School of Management and Krannert Graduate School, Krannert Building, Purdue University, West Lafayette, IN 47907, USA*

Receiving date: November 1987  
Rev rec date: March 1992

Abstract: In modeling flows and controls in transportation, distribution, communication, manufacturing systems, etc., it is often convenient to represent the system as a store-and-forward network. In such networks it is common for time, space, attention, or other resources, to be shared between sets of neighbouring nodes. For example, neighbouring nodes may share storage space, machine time, operating time, etc. The allocation of this shared resource among nodes determines a set of ‘controls’ on the network arc flows. We develop a multi-period network model which describes such storage and forwarding, and the sharing of resources (controls) between subsets of nodes. To solve the model we develop algorithms which take advantage of the embedded network structure of the problem. Each of the algorithms is based on iterating between (a) solving a least-cost capacitated network flow problem with fixed capacities (controls) and (b) solving a set of simple small scale problems to update these controls. In a series of computational experiments we found that an (‘unoptimized’) implementation of the algorithms performed between 13 and 42 times faster than a good linear programming code, which is the natural alternative. Also, by decomposing the problem, the algorithms make solving larger scale problems tractable, and are suitable for implementation on parallel processors.

Keywords: Networks; Optimization; Transportation; Traffic flows

1. Introduction

We consider networks with time varying demands in which throughput at subsets of nodes is restricted by some resource (e.g., time, or operating capacity) which may be shared among the nodes in each subset of nodes. This includes communications systems with buffer storage at various nodes, logistic systems...
with transportation links and storage facilities, air transportation systems with limited number of landing slots at various airports, and traffic networks in which delays occur due limited throughput capacity at various intersections. A feature of such systems is presence of competing demands for shared resources such as communication lines, buffer capacity, warehouse capacity, landing slot, or time, e.g., ‘green’ time at traffic intersections.

Here we model optimal flows and optimal control settings over time for the above contexts. The problem is formulated as a multiperiod optimization model where the sum of all the travel costs and delay costs on the network is to be minimized. For illustration and for specificity we will adopt terminology and examples from road traffic frequently below. Gazis (1974a, b) and d’Ans and Gazis (1976) propose optimal control and linear programming models for determining optimal control settings at each of the nodes of a store-and-forward network, but assume an exogenously specified route assignment. Here we wish to allow freedom of route choice, while determining the flow controls; in this paper, we develop a model which simultaneously determines optimal control settings or ‘green’ times at the intersections and the optimal pattern of flows for a network with time-varying demands.

We here follow a common practice in both static and dynamic network flow models in adopting a deterministic rather than stochastic approach. We do this for several reasons. First, random variation over time may be small compared with other (deterministic) causes of flow variation. Second, we are specifically concerned with modeling congestion during periods (e.g., peak periods) where the arrival rates at queues (e.g., in front of controlled intersections) exceed the service rates for these queues for much of the time span being modeled. In this situation stochastic effects become less important, and the usual steady state or equilibrium models do not apply. Third, we are specifically concerned with modeling networks with time-varying flows, hence again the usual equilibrium queueing theory results do not apply. Finally, tractable stochastic models capable of optimizing network flows with time-varying demands and route choice have not yet been developed.

The paper is organized as follows. We develop models for fixed flow controls (fixed ‘green’ times, or fixed time sharing, etc.) in Section 2, and for variable traffic controls or variable resource (time) sharing in Section 3. The latter models have an embedded network structure and could be formulated as least cost network problems subject to additional ‘non-network’ type constraints. In Sections 4–7 we develop resource-directive decomposition algorithms to solve the class of models developed in Section 3. These algorithms iterate between (a) solving a least-cost network problem with fixed flow controls and (b) solving a set of simple small scale problems (one for each control point in each time period) to update these controls. This decomposition approach is attractive for several reasons: (a) without it, the problem can easily become intractably large even for moderate size applications, (b) it allows the algorithms to be adapted to drive a decentralized control strategy, and (c) the subproblems can be solved in parallel hence greatly reducing computing time. In Section 8 we set out our computational experience with the algorithms. We find for example that, for a variety of problem sizes, the subgradient algorithm solves the problem from 13 to 42 times faster than a good LP code, which is the natural alternative method of solution. In Section 9 we discuss extending the problem formulation to embrace special forms of flow controls, and we conclude in Section 10.

2. A model with fixed time sharing

Consider a network consisting of a set of nodes $A = \{\ldots, j, \ldots\}$ linked by directed arcs $(j, k)$, $j \in A$, $k \in A$. We will sometimes refer to this as the ‘spatial’ network to distinguish it from the ‘time–space’, or ‘time-expanded’, network below. Let each node in $A$ represent a queue or buffer or store or processing facility, etc. In the class of problems with which we are concerned there is usually an upper bound on the flow through each node. We could enforce this with a bound on the sum of the inflows or outflows from the node. However, to retain the network structure of the problem, we replace each node $j$ with two nodes $j$ and $j’$ linked by a new ‘artificial’ arc $(j, j’$). The bound on node throughput is then a simple bound on the flow on arc $(j, j’$).
Let the time span to be modeled be subdivided into \( t = 1, \ldots, T \), time periods. To represent flows varying over time it is convenient to think of points in time as analogous to points in space, and hence construct a time–space network (also called a time-expanded network or a store and forward network), as follows (see Figure 1 which expands a simple corridor ‘network’ into the corresponding time–space network). Replicate each node in the spatial network for each of the \( T \) time periods. Then construct arcs joining spatial nodes in different time periods as follows.

‘Storage’ arcs or ‘queuing’ arcs. Link node \( j \) at time \( t \) to the same (spatial) node \( j \) in the next period \( t + 1 \). The link is denoted by \((t, j), (t + 1, j)\), and represents storage or holdover from period \( t \) to \( t + 1 \) in waiting areas, storage facilities, etc.

‘Travel’ arcs. Link spatial node \( j' \) at time \( t \) to spatial node \( k \) at time \( t \) if and only if \( t - r \) is the known time needed to travel from node \( j' \) to node \( k \). The arc is denoted by \((t - r, j'), (t, k)\). It represents a road, route, communication line, etc.

‘Service’ arcs. For each node \( j \) a ‘service’ arc or ‘outflow’ arc \((j, j')\) carries the outflow from \( j \). In the time-expanded network the service arc is denoted by \((t, j), (t, j')\).

For example, in the case of a production/manufacturing layout, a ‘service arc’ may represent a particular type of operation to be performed on parts or pieces, a ‘storage arc’ represents parts/pieces waiting to be processed, and a ‘travel arc’ represents parts/pieces being moved between different machines or work-stations. On the other hand, in the case of a road traffic network, a ‘travel arc’ represents a road/street/traffic lane, a ‘queueing arc’ represents the waiting lane/area in front of a traffic light, and a ‘service arc’ represents the outflow from a queueing arc. Also, in the case of road traffic, we assume that travel times are approximately constant on ‘travel’ arcs, that congestion occurs at traffic signals and controlled intersections and we here assume a single destination, e.g., the central business district.

We also introduce the following notation. In defining and using this notation we do not always explicitly distinguish between the spatial network and the time-expanded network, since the intention should be clear from the context. Introduce variables \( x_{tj'k}, x_{tjj} \) and \( x_{tjj'} \) denoting flows setting out at time \( t \) on travel arcs, storage arcs and service arcs respectively. Thus \( x_{tj'k} \) is the inflow to ‘travel’ arc \((t, j'), (t + \tau, k)\) in period \( t \). \( x_{tjj} \) is the volume waiting at node \( j \) in period \( t \) (or equivalently, the flow from node \( j \) in period \( t \) to the same node \( j \) in period \( t + 1 \)). \( x_{tjj'} \) is the outflow from node \( j \) in period \( t \) (or equivalently, the flow on ‘service’ arc \((t, j), (t, j')\) in period \( t \)). Also introduce parameters:

\[
D_{tj} = \text{Exogenous inflow or outflow at node } j \text{ in period } t. \ D_{tj} \geq 0 \text{ if } j \text{ is a supply node, } D_{tj} \leq 0 \text{ if } j \text{ is a demand node and } D_{tj} = 0 \text{ if } j \text{ is a transshipment node which is neither a supply or demand node.}
\]

\[
l_t = \text{Length of time period } t, \text{ in minutes, etc.} \ (l_t \text{ is not needed until Section 3.)}
\]

\[
\tau_{tj'k} = \text{Number of time periods taken to travel from node } j' \text{ to neighbouring node } k, \text{ arriving at node } k \text{ in period } t. \text{ Hence this flow set out from node } j' \text{ at time } \eta(tj'k) = (t - \tau_{tj'k}).
\]
\begin{align*}
c_{ij} \text{ and } c_{ij'k} &= \text{Cost incurred per unit of } x_{ij} \text{ and } x_{ij'k} \text{ respectively. (If costs are proportional to elapsed time, then } c_{ij} = kl_t \text{ and } c_{ij'k} = klt_{ij'k} \text{ where } t \text{ is the length of each time period and } k \text{ is the value of time.)} \\
\bar{x}_{ij} &= \text{Outflow capacity for node } j \text{ in period } t. \text{ That is } x_{ij'} \leq \bar{x}_{ij}.
\end{align*}

Let \( A(j') \) be the set of nodes which are immediate successors of node \( j' \), (i.e., nodes linked to \( j' \) by an arc pointing out of \( j' \)) and let \( B(j) \) be the set of nodes which are immediate predecessors of node \( j \). Let \( T_j = \{1, \ldots, T\} \) for all nodes \( j \in A \), but other definitions are also useful.

**Flow conservation constraints**

The inflow to node \( j \) at time \( t \) must equal the outflow \( x_{ij'} \). The inflow consists of (see Figure 1): (a) the exogenous demand \( D_{ij} \), plus (b) the net change in the volume held in waiting (i.e., \( x_{t-1,ij} - x_{ij} \)), plus (c) the sum of the inflows which set out from predecessor nodes \( B(j) \) in earlier periods. The latter (c) is \( \sum_{i \in B(j)} x_{\eta(tij)ij} \), where \( \eta(tij) = t - \tau_{ij} \), since the flow which arrives at node \( j \) at time \( t \) set out from node \( i \) at time \( t - \tau_{ij} \). Thus,

\begin{equation}
x_{ij'} = (x_{t-1,ij} - x_{ij}) + \sum_{i \in B(j)} x_{\eta(tij)ij} + D_{ij} \quad \text{for all } t \in T_j, \quad j \in A.
\end{equation}

(P.1)

On the other hand, the inflow \( x_{ij'} \) to node \( j' \) at time \( t \) must equal outflow from node \( j' \) to successor nodes \( A(j') \) in later periods. Thus,

\begin{equation}
x_{ij'} = \sum_{k \in A(j')} x_{ij'k} \quad \text{for all } t \in T_j, \quad j \in A.
\end{equation}

(P.2)

(\text{Note that the left-hand-side of both (P.1) and (P.2) is } x_{ij'} \text{ hence we could eliminate } x_{ij'} \text{ and combine (P.1) and (P.2) into a single conservation equation, for each } t \text{ and } j. \text{ However, we wish to retain } x_{ij'} \text{ as an explicit variable, since this allows us to retain the pure capacitated network form of the model when we introduce capacities on } x_{ij'} \text{ below.)}

**Flow control constraints**

Let the maximum permissible outflow from node \( j \) in period \( t \) be \( \bar{x}_{ij} \), hence

\begin{equation}
x_{ij'} \leq \bar{x}_{ij} \quad \text{for all } j \in A, \quad t \in T_j.
\end{equation}

(P.3)

Also, there may be upper limits on the flows on some travel arcs or waiting arcs, thus

\begin{equation}
x_{ij} \leq \bar{x}_{ij} \quad \text{and/or } x_{ij'} \leq \bar{x}_{ij}.
\end{equation}

**Objective function and model formulation**

The costs incurred by the flows \( x_{ij} \) and \( x_{ij'} \) are of course \( c_{ij}x_{ij} \) and \( c_{ij'}x_{ij'} \). Thus the problem of finding the set of arc flows which minimize the sum of the travel costs plus queueing costs over all arcs and time periods can now be stated as

\begin{equation}
\text{(FP)} \quad \text{minimize } z_{FP} = \sum_{t \in T_j} \left( \sum_{j \in A} c_{ij}x_{ij} + \sum_{j' \in A} \sum_{k \in A(j')} c_{ij'k}x_{ij'k} \right)
\end{equation}

(P.0)
subject to, for all $j \in A$ and $t \in T_j$,

$$
\{\lambda_{tj}\}, \quad x_{tj} = (x_{t-1,j} - x_{tj}) + D_{tj} + \sum_{i' \in B(j)} x_{t(i')j(i')}.
$$

$$
\{\mu_{tj}\}, \quad x_{tj} = \sum_{k \in A(j')} x_{tkj'},
$$

$$
\{\alpha_{tj} \geq 0\}, \quad x_{tj} \leq \bar{x}_{tj},
$$

$$
(x_{tj/k}, x_{tj'}, x_{tj}) \geq 0 \quad \text{for all } k \in A(j').
$$

The $\lambda_{tj}$'s, $\mu_{tj}$'s, and $\alpha_{tj}$'s are dual variables associated with the constraints: they will be used in Sections 4–7 below. The above linear program is a pure least-cost capacitated network program, and hence can be solved using one of the various available fast efficient network computer codes (Kennington and Helgason, 1980; Grigoriadis, 1986).

The following proposition is useful below, for example in the proof of Proposition 3.

**Proposition 1.** (a) The value $z_{FP}$ of program FP is bounded from below, at $z_{FP} \geq 0$. (b) If the conservation equations (P.1) for demand nodes ($D_{tj} \leq 0$) are rewritten as ‘$\geq$’ rather than as ‘$=$’, then program FP always has a feasible solution.

**Proof.** (a) All coefficients and variables in the objective function are nonnegative, hence its value is nonnegative. (b) It is sufficient to state one feasible solution, as follows. Let $x_{tj} = 0$ and $x_{tj/k} = 0$ for all $t$, $j'$ and $k$. This satisfies (P.2)–(P.4) and reduces (P.1) to $x_{t+1,j} = x_{tj} + D_{tj}$. By recursion the latter has a solution $x_{t+1,j} = \sum_{t=1}^{T} D_{tj}$. □

**Remark.** If we introduce upper bounds on the $x_{tj}$’s, then part (b) of Proposition 1 may not hold due to the inability of some of these arc capacities ($\bar{x}_{tj}$) to handle all of the demand $D_{tj}$. However, even in this case we can always ensure that program FP has a feasible solution for all arc capacities. To achieve this, simply introduce artificial uncapacitated arcs linking origins directly to destinations. These arcs should be assigned large cost coefficients (penalties) in the objective function, to ensure that they do not appear in an optimal solution of program VP.

### 3. A model with variable time (resource) sharing

In model FP above there is a fixed upper bound $\bar{x}_{tj}$ on the aggregate outflow from each storage point $j$ in each period. We now, as discussed in the introduction, let these bounds become variables in the model, and introduce relationships, and hence trade-offs, between the bounds imposed on neighbouring facilities or queues.

Let $y_{tj}$ be the time, or other resource, devoted by a ‘server’ to node/queue $j$ in period $t$. Let the maximum (capacity) outflow rate from node $j$ be $b_j$ per unit time when the node is being served (e.g., when its traffic light is ‘green’), and zero when it is not being served. Then the maximum (capacity) outflow from node $j$ in period $t$ is $\bar{x}_{tj} = b_j y_{tj}$, so that (P.3) becomes

$$
x_{tj} \leq b_j y_{tj} \quad \text{for all } j \in A, \quad t \in T_j.
$$

(P.3')

Let the set $A$ of nodes (facilities/processes/queues) be grouped into mutually exclusive subsets or clusters, each subset being associated with a single ‘server’. Let $R$ be the set of servers and let $J_r$ be the set of nodes associated with server $r \in R$. For example, in the language of road traffic networks, each server $r$ represents a controlled intersection which serves all the traffic lanes $j \in J_r$ which lead into the intersection.
In most applications there will be constraints on \( y \), usually consisting of independent sets of linear (affine) constraints for each control point \( r \in R \) in each time period. Thus let \( y_{tr} \in S_{ytr} \) represent a set of affine constraints on \( y_{tr} \), where \( y_{tr} = [y_{tj}, \forall j \in J_r] \) is the vector of \( y_{tj} \)'s at control point \( r \in R \), and let \( S_y = \{S_{ytr}, \forall t \text{ and } r\} \).

We can now easily extend the fixed flow control model FP to allow variable flow controls: simply let \( y \in S_y \) be variable, thus

**VP:** Same as program FP but with constraints (P.3) replaced by (P.3') subject to \( y \in S_y \).

Note that if \( y \) is held constant in program VP, then VP reduces to program FP, which can be solved as a pure network program.

### 3.1. Special form of control constraints \( y \in S_y \)

In many applications which we have considered (traffic, manufacturing, warehousing distribution, etc.) the constraints \( \{y_{tr} \in S_{ytr}\} \) can be further specialized as in the following paragraphs. Since this specialization of \( S_{ytr} \) is essential in modeling signalized road traffic intersections, it is perhaps easiest to think of that context. We will refer to this form of \( S_{ytr} \) as \( S_{ytr}^* \). Such additional structure on \( S_{ytr} \) may be exploited in algorithms to solve program VP.

Suppose, for the moment, that each control point \( r \) can serve only one node/queue/facility at a time. Then the sum of the time \( y_{tj} \) allocated at control point \( r \) must add up to the length of the time period \( l_r \), thus

\[
\sum_{j \in J_r} y_{tj} = l_r \quad \text{for all } r \in R, \ t \in T_r \quad \text{(P.5')}
\]

For technical and other reasons there are often upper and/or lower bounds on the \( y_{tj} \)'s, thus

\[
y_{tj} \leq y_{tj} \leq y_{tj}^{+} \quad \text{for all } j \in J_r, \ t \in T_r \quad \text{(P.6')}
\]

For example in traffic applications a minimum time is needed to allow flows to start up after a traffic light turns from red to green. Also, a minimum time allocation may be needed to avoid excessive user impatience, or to ensure fairness.

Now let us remove the assumption that only one node at a time can be served at each control point. For many types of controls (e.g., for traffic, manufacturing, communication, distribution, etc.), each control point \( r \in R \) can handle several nodes (queues, operations, etc.) simultaneously. For example, in manufacturing, work-station \( r \) may handle two or more part types simultaneously. In communication or computer networks several channels may operate simultaneously at each control point. And in road traffic each control point (traffic lights) normally allows two or more lanes to operate at the same time.

To see how this affects the constraints (P.5')–(P.6'), let \( J_r^* \subset J_r \) be a subset of the \( J_r \) nodes at control point \( r \), such that none of the nodes in \( J_r^* \) can be served simultaneously with each other, and let

\( J_j \subset J_r \) be a set of nodes which can be served only simultaneously with node \( j \in J_r^* \).

(More formally, \( \bigcup_{j \in J_r^*} J_j = J_r \) and \( \bigcap_{j \in J_r^*} J_j \) is an empty set.) Since node \( k \in J_j \) is served concurrently with node \( j \) we must have \( y_{tk} \leq y_{tj} \) for all \( k \in J_j \), or more generally,

\[
y_{tk} \leq a_{tkj} y_{tj} \quad \text{for all } k \in J_j, \ j \in J_r^*, \quad \text{(P.7)}
\]

where \( 0 \leq a_{tkj} \leq 1 \) is a constant. Further, this change (i.e., allowing more than one node to be served simultaneously at each control point) changes \( J_r \) to \( J_r^* \) in (P.5') and changes \( A \) to \( A^* \) in (P.6'), where \( A^* = \{J_r^*\} \) is a subset of \( A = \{J_r\} \).

Thus

\[
\sum_{j \in J_r^*} y_{tj} = l_r \quad \text{for all } r \in R, \ t \in T_r \quad \text{(P.5)}
\]

\[
y_{tj} \leq y_{tj} \leq y_{tj}^{+} \quad \text{for all } j \in A^*, \ t \in T_j \quad \text{(P.6)}
\]
In summary, the controls (constraints) on the time allocation vector \( y_{tr} \) at control point \( r \) at time \( t \) are
\[
y_{tr} \in S_{y_{tr}}^w = \{(P.5) - (P.7), \text{ for the given } r \text{ and } t\}.
\]

4. Solving program VP: Resource directive decomposition approaches

If the constraints \( y \in S \) are linear, then program VP above is a linear program and could be solved as such. However:

(a) Even medium size network problems in communications, traffic flow, etc., tend to yield very large scale linear program formulations. Further, introducing time periods, as in the present context, multiplies the number of variables and constraints by approximately the number of time periods. Thus, realistic size multi-period network problems can easily be larger than can be handled by available linear programming packages.

(b) The constraints \( y_{tr} \in S_{y_{tr}} \) may be nonlinear, or nonconvex (say integer) in which case VP is no longer an LP. In this case VP is even less likely to be solvable using existing standard packages.

We therefore develop alternative methods for solving VP. A further advantage of these methods is that they decompose the problem into smaller subproblems (one for each control point in each time period). These subproblems allow the algorithm to be used as part of a decentralized control strategy. Also, the subproblems can be solved in parallel thus further reducing computing time.

As noted above, if we fix the service times \( y = S_y \) in program VP, then VP reduces to the pure least-cost capacitated network problem FP, with capacity constraints of the form (P.3), i.e., \( x_{ij} \leq b_{ij} y_{ij} \). This immediately suggests the following general approach to solving VP: repeat steps (a) and (b) below until a convergence criterion is satisfied.

(a) Hold service time allocation \( y = [y_{tr}] \) constant in program VP and solve the resultant least-cost network problem FP.

(b) Choose a new value for \( y \in S_y \), so as to yield a better value of the objective function of program VP, and return to step (a).

We are not concerned here with algorithms to solve the pure least-cost network subproblem FP. Fast efficient algorithms and computer codes are already available for this (see for example, Kennington and Helgason, 1980; Jensen and Barnes, 1980; Grigoriadis, 1986). Also, the subproblem FP is a multiperiod network flow problem. Specialized network algorithms have been developed to exploit such multiperiod structure (see the surveys Aronson, 1989; Aronson and Thompson, 1984).

We need to find a direction in which to change \( y \) in (b) above, at each iteration. First note that the dual variables associated with the capacity constraints (P.3) in program FP can be used to find a direction in which to vary \( y \) so as to improve the objective function value \( z_{FP} \) of program FP. Next note that the objective functions of FP and VP are identical (since \( y \) does not appear in either), so that any change in \( y \) which improves \( z_{FP} \) also improves \( z_{VP} \). But a change in \( y \) which improves \( z_{FP} \) may not be feasible in program VP, since it may not satisfy the constraints \( y \in S_y \) which distinguish program VP from program FP. Thus we need to adjust any proposed change in \( y \) so as to satisfy \( y \in S_y \). There are various ways in which this adjustment/choice of \( y \) at each iteration can be accomplished, and each of these approaches yields a different algorithm for solving VP.

The approach outlined above can be described as a resource directive decomposition, where the service time allocation \( y \) is the ‘resource’ being reallocated at each iteration. And since there are various ways in which we can choose/adjust \( y \) at each iteration we have various forms of resource directive decomposition. Two of these are set out in Sections 4 and 5 respectively below. One is based on outer-linearization or tangential approximation and the other on sub-gradient optimization.

These decomposition approaches to solving VP have three major advantages.

- First, at each iteration the subproblem is a minimal cost capacitated network problem which is always feasible (Proposition 1 above).
Second, at each iteration we have a feasible (even if sub-optimal) solution for the original problem VP. This has the advantage of ensuring that the system controller or engineer will have a set of feasible controls to implement, even if (because of computational cost) we stop the algorithm well short of optimality. Feasible controls \( y \) can be very important. For example, in the case of road traffic flows a solution which violates constraints \( y = S_y \) may be impossible to implement, may violate traffic engineering standards or regulations, and hence incur congestion and travel costs which are not included in the model.

Third, it is likely that in a realistic decision support or control context the number of control variables \( \{y_{tj}'s\} \) which will be allowed to vary at any one time in program VP will be relatively small. As a result, there may be only a few \( y_{tj} \) variables and only a few ‘nonnetwork’ type constraints in program VP. This of course makes it even more attractive to solve VP by using an algorithm which takes advantage of network substructure of VP.

The dual of program FP will be needed in setting out the algorithm in the next section, hence it is convenient to set it out here. The dual of FP can be written, for given \( \{y_{tj}\} \), as

\[
\text{(FD)} \quad \maximize \quad z_{FD} = \sum_{t \in T_j} \sum_{j \in A} (D_{tj} \lambda_{tj} - b_j y_{tj} \alpha_{tj})
\]

subject to, for all \( j \in A \) and \( t \in T_j \),

\[
\{x_{tjj} \geq 0\}, \quad \lambda_{tj} - \mu_{tj} \leq \alpha_{tj}, \tag{D.1}
\]

\[
\{x_{tj} \geq 0\}, \quad \lambda_{tj} - \lambda_{t+1,j} \leq c_{tj}, \tag{D.2}
\]

\[
\{x_{tj'k} \geq 0\}, \quad \mu_{tj} - \lambda_{(t+t+\tau_{j'k})k} \leq c_{tj'k}, \tag{D.3}
\]

\[
\alpha_{tj} \geq 0, \tag{D.4}
\]

where \( \{\lambda_{tj}\}, \{\mu_{tj}\} \) and \( \{\alpha_{tj}\} \) are the dual variables corresponding to constraints (P.1), (P.2) and (P.3) respectively of program FP, and \( \{x_{tjj}'\}, \{x_{tjj}\} \) and \( \{x_{tj'k}\} \) are the dual variables corresponding to (D.1)–(D.3) respectively.

Also, the following notation will be useful in Sections 5–6 below. \( S_{FP} \) and \( S_{FD} \) are the sets of feasible solutions of programs FP and FD respectively, i.e.,

\[
S_{FP} = \{(x_{tjj}, x_{tjj'}, x_{tj'k}) | (P.1)-(P.4)\} \quad \text{and} \quad S_{FD} = \{(\lambda_{tj}, \mu_{tj}, \alpha_{tj}) | (D.1)-(D.4)\}.
\]

5. A tangential approximation approach

In this section we discuss how a tangential approximation, or generalized Benders decomposition, approach can be used to solve program VP. When the ‘complicating terms’ in a mathematical program are linear and noninteger, the procedure is referred to as tangential approximation (Geoffrion, 1970; Kennington and Helgason, 1980), and has been used to solve multi-commodity network problems.

The ‘complicating’ variables in model VP are the \( y_{tj}'s \): when the \( y_{tj}'s \) are held constant the model reduces to a pure least cost capacitated network subproblem FP. The algorithm proceeds by iterating between this subproblem FP and a ‘master’ problem, which is obtained as follows. The original problem VP can be rewritten as

\[
\text{(VP)} \quad \min \min \ z_{FP}(x), \quad y \in S, \quad x \in S_{FP}(y)
\]
and since the inner minimization is a linear program (FP) it can be replaced by its dual (FD) to yield the equivalent program

\[(VP')\]

\[
\min_{y \in S_y, \gamma \in S_{\gamma D}(y)} \max_{y \in S_y} z_{FD}(\gamma, y),
\]

where \(\gamma = [\lambda_{ij}, \alpha_{ij}, \mu_{ij}]\). Let the constant vectors \(\{y^h, h = 1, \ldots, H\}\) comprise all of the \(H\) basic solutions to program FD. Then the inner maximization above can be replaced by maximum \(\{z_{FD}(y^h, y), h = 1, \ldots, H\}\), so that \(VP'\) can be rewritten as

\[(M)\]

\[
\min_{y \in S_y, \gamma \in S_{\gamma D}(y)} \max_{h = 1, \ldots, H} z_{FD}(y^h, y).
\]

A standard decomposition would at this stage replace \(M\) with the following equivalent master problem:

\[(M)\]

\[
\min_{y \in S_y} z_M, \\
\text{subject to } z_M \geq z_{FD}(y^h, y) \text{ for } h = 1, \ldots, H.
\]

However, we can here take advantage of the structure of program \(M\) to perform a further decomposition as follows.

**Proposition 2.** If \(S_y = \{S_{y_{tr}}, \text{ for all } t \in T_r, r \in R\}\), then the master program \(M\) reduces to \(|R| |T|\) independent linear programs of the form

\[(M_{tr})\]

\[
\text{minimize } z_{tr}, \\
\text{subject to } y_{tr} \in S_{y_{tr}}, z_{tr} \geq \sum_{j \in J_r} \left(D_{ij}a_{ij}^h - b_jy_{ij}h_{ij}\right), \text{ for } h = 1, \ldots, H.
\]

**Proof.** Program \(M\) can be written as

\[
\min_{y} \left\{ \max_{h = 1, \ldots, H} \left[ \sum_{i \in I_t} \sum_{j \in A} \left(D_{ij}a_{ij}^h - b_jy_{ij}h_{ij}\right) \right] \mid y_{tr} \in S_{y_{tr}}, \text{ for all } r \in R, t \in T_r \right\}.
\]

The constraints in this program are grouped into independent sets of constraints, one set for each \(t\) and \(r\). Also, the objective function terms can be grouped into independent summation expressions, one for each \(t\) and \(r\). This grouping immediately reduces \(M\) to a set of independent programs, one for each \(t\) and \(r\), thus,

\[(M_{tr})\]

\[
\text{minimize } \left( \max_{h = 1, \ldots, H} \left[ \sum_{j \in J_r} \left(D_{ij}a_{ij}^h - b_jy_{ij}h_{ij}\right) \right] \right) \mid y_{tr} \in S_{y_{tr}}, y_{tr} \{h = 1, \ldots, H\}
\]

which can be rewritten as in the proposition. The proposition follows immediately. \(\Box\)

We can now restate the master problem \(M\) as \(M = \text{solve } M_{tr} \text{ for all } t \text{ and } r\).
Where $M_{tr}$ is as defined above. Actually, in the algorithm set out below, the master problem does not contains all the cut constraints (M.1) of the above ‘master’ problem. Instead, as usual in cutting plane algorithms, these constraints (M.1) are added one at a time at each iteration.

The algorithm starts with a feasible $y$, i.e., satisfying $\{y_{tr} \in S_{y_{tr}}\}$. At each iteration the subproblem FP is solved for fixed $y$ and a cut constraint (a member of (M.1)) is generated for the master problem $M$ (or, more specifically, for each component program $M_{tr}$ of the master problem). Then $M$ is solved to find the next allocation of $y$, and so on. As a termination criterion we compute upper and lower bounds on the (unknown) optimal value of the objective function of program VP at each iteration. The upper bound consists of the objective function value of subproblem FP, and the lower bound consists of the objective function value of the master problem $M$. The algorithm terminates when the upper and lower bounds are sufficiently close together. The algorithm is set out more formally below.

But first it is worth noting that a desirable feature of this algorithm for solving the present problem VP is that no ‘feasibility cuts’ are required. A feasibility cut (Murty, 1976, Appendix 2) is required in the master problem at iteration $i$ if and only if the dual subproblem FD is unbounded at iteration $i$. In the present case, FP is feasible and bounded for all $y \in S_y$ (remark in Proposition 1), hence its dual FD is feasible and bounded, hence no feasibility cuts are required.

Finally, we note that in general in applications of Benders decomposition or tangential approximation to solving mathematical programming problems we can have, (a) unboundedness of the master problem, or (b) infeasibility of the master problem, or (c) unboundedness of the subproblem FP, or (d) infeasibility of the subproblem FP. Fortunately: 

**Proposition 3.** None of these four difficulties can occur in the present case, if $y \in S_y$ is feasible and bounded, e.g., if $S_y$ consists of or includes (P.5') or (P.6') or ((P.5)–(P.7)).

**Proof.** The remark in Proposition 1 (Section 2) ensures that program FP is feasible and bounded, which rules out (c) and (d). Then by LP duality, FD the dual of FP, is feasible and bounded. The constraints (M.1) from FD are therefore feasible. Also, by assumption, the constraints $y \in S_y$ in $M$ have a feasible solution, bounded above and below. Substituting any such finite $y$ in constraints (M.1) yields a finite $z_M$. There are no other constraints in program $M$, hence the objective function $z_M$ is bounded above and below.

**ALGORITHM A1:** to solve program VP.

**Step 1.** (Initialization.) Select a convergence tolerance parameter $\varepsilon > 0$, and initialize upper bound $UB \leftarrow +\infty$, lower bound $LB \leftarrow -\infty$, and iteration counter $i \leftarrow 1$. Select an initial feasible $y$, i.e., a $y_0 \in S_y$ (see Section 6 below). Go to Step 3.

**Step 2.** (Master problem.) Solve the current master problem. Let $(z^i_M, y^i)$ be an optimal solution of $M$. Set $LB \leftarrow z^i_M$.

**Step 3.** (Subproblem.) Solve the least-cost capacitated network problem function $FP$ with $y = y^i$. Let \( \{z_{FP}^i, x_{ij}', x_{ij}, x_{ij}''\} \) be an optimal solution of FP, and let \( \{z_{FD}^i, \lambda_{ij}', \mu_{ij}, \alpha_{ij}'\} \) be the corresponding optimal dual solution. If $z_{FP}^i \leq UB$, set $UB \leftarrow z_{FP}^i$. If $UB \leq LB + \varepsilon$, terminate.

**Step 4.** (Add a Benders cut to the master program $M$.) Include the constraints, $z_M \geq \sum_{j \in J}(D_{ij}x_{ij} - \langle b_{ij}, a_{ij} \rangle y_{ij})$, in the master problems $M_{tr}$. Set $i \leftarrow i + 1$, and return to Step 2.

6. A subgradient approach

Subgradient optimization was first discussed by Shor (1964). It has been used successfully in solving many problems including multicommodity flow problems (Held et al., 1974; Kennington and Shalaby, 1977; Kennington and Helgason, 1980).

A subgradient approach to solving the present problem VP may be stated, in brief, as follows. Beginning with an initial feasible $y$ (i.e., a $y \in S_y$), we solve subproblem FP. At an optimum of FP, a
subgradient of \( z^0_{FP}(y) \) with respect to \( y \) yields a direction in which to change the current value of \( y \) so as to improve the objective \( z_{VP} \) of \( VP \). However, moving along this subgradient direction may result in an infeasible \( y \) (i.e., \( y \not\in S_y \)), hence we must project the proposed \( y \) back onto the feasible region, to obtain a new trial \( y \) for subproblem \( FP \), and so on.

As outlined above, the subgradient decomposition algorithms deals with solving \( VP \) restated in the following equivalent form:

\[
\begin{align*}
VP &= M': \\
\min_y z^0_{FP}(y), \\
\text{subject to } y \in S_y,
\end{align*}
\]

where \( z^0_{FP}(y) \) is the optimal value of program \( FP \) (and hence program \( VP \)) for given \( y \), i.e.,

\[
\begin{align*}
z^0_{FP}(y) &= \min_x \left\{ \sum_{i \in T_j} \sum_{j \in A} c_{ij} x_{ij} + \sum_{k \in A(j)} c_{ij'k} x_{ij'k} \mid x \in S_{FP} \right\}
\]

where \( x = [x_{ij}, x_{ij'}, x_{ij'k}, \text{for all } t, j \text{ and } k] \). Note that \( M' \) is a linearly constrained convex program, since:

**Proposition 4.** \( z^0_{FP}(y) \) is a convex function over \( y \geq 0 \).

**Proof.** Program \( FP \) is an LP, and \( y \) appears in \( FP \) only in the right hand side of the constraints. It is well known that the optimal value of a minimization LP is a convex function of the constraint right-hand side values (Murty, 1983, Chapter 8). □

The subgradient of \( z^0_{FP}(y) \) with respect to \( y \) is obtained as follows.

**Proposition 5.** Let \( \{\alpha_{ij}\} \) be the dual variables corresponding to constraints (P.3) at an optimum of program \( FP \). Let \( \{\alpha_{ij}\} = \{\tilde{\alpha}_{ij}\} \) when the right-hand side of (P.3) is \( \tilde{x}_{ij} = b_j \tilde{y}_{ij} \). Then \( [-b_j \tilde{\alpha}_{ij}] \) is a subgradient of \( z^0_{FP}(y) \), at \( \tilde{y} \).

**Proof.** In program \( FP \) the vector \( y \) appears only in the right-hand side of constraints (P.3). Since \( FP \) is a linear program, consider (for simplicity) an LP of the form minimize \( \{z \mid Ax \leq b\} \). Let \( z^0(b) \) denote the optimal solution of this LP for a given right-hand side \( b \). It is well-known that a subgradient of \( z^0(b_1, \ldots, b_m) \) with respect to \( b_i \) is given by the dual variable \( (-\alpha_i) \) associated with the \( i \)-th constraint in \( Ax \leq b \). But if \( b_i = \bar{b}_i y_i \) (as in program \( FP \)), then a subgradient with respect to \( y_i \) is \( \alpha_i (db_i/dy_i) = -\alpha_i \bar{b}_i \). □

The subgradient algorithm can now be set out as follows.

**ALGORITHM A2:** to solve program \( M' \), and hence program \( VP \).

Step 1. (Initialization.) Select (a) convergence tolerance parameters \( \epsilon > 0, \eta > 0 \), (b) a maximum number of iterations \( I \), (c) a sequence of step-size constants \( \delta_1, \delta_2, \ldots \), (see Section 5.2 below), and (d) an initial \( y^0 \in S_y \) (see Section 7 below). Also, initialize an iteration counter \( i \leftarrow 1 \), and an upper bound \( UB \leftarrow +\infty \). Compute a lower bound \( LB \leftarrow z_{LB} \) where \( z_{LB} \) is the value of the objective function of program \( FP \) either (a) without the constraints \( y \in S_y \) or (b) with the constraints \( y \in S_y \) replaced by bounds which are weaker than \( y \in S_y \).

Step 2. (Solve subproblem \( FP \).) Solve the least-cost capacitated network problem \( FP \). Let \( \{z^i_{FP}, x^i_{ij}, x^i_{ij'}, x^i_{ij'k}\} \) be an optimal solution of \( FP \), and let the corresponding dual solution be \( \{\lambda^i_{ij}, \mu^i_{ij}, \alpha^i_{ij}\} \). If \( z^i_{FP} < UB \), set \( B \leftarrow z^i_{FP} \) and go to Step 3; otherwise go to Step 4.
Step 3. (Termination test.) If \( \text{max}_{t,j}(y^i_{tj} - y^{i-1}_{tj}) \leq \epsilon \) or UB − LB ≤ \( \eta \) (LB), terminate with the incumbent solution as a near optimal solution. If \( i = I \), terminate with the incumbent solution as the best solution found.

Step 4. (Compute subgradient and update \( y^i \).) Compute a subgradient of \( z^0_{fp}(y) \), i.e., \( \{ -b_j \alpha^j_i \} \), and a proposed new allocation \( \hat{y}^{i+1} \) in the subgradient direction, i.e., \( \hat{y}^{i+1}_{tj} = y^i_{tj} + \delta_i (b_j \alpha^j_i) \). The step size \( \delta_i \) at iteration \( i \) is computed as in Section 5.2 below. Project \( \hat{y}^{i+1} \) into the feasible region \( S_y \) of program \( M' \), by computing a \( y^{i+1} \in S_y \) 'as close as possible' to \( \hat{y}^{i+1} \) (see below). Set \( i = i + 1 \), and return to Step 2.

6.1. The projection operation: Updating \( y^i \)

For the projection operation, in Step 4 above, we can find a new \( y^{i+1} \) 'close' to \( \hat{y}^{i+1} \) by minimizing the sum of squared deviations of \( y^{i+1} \) from \( \hat{y}^{i+1} \), subject to \( y^{i+1} \in S_y \), thus

\[
\text{(Q)} \quad \text{minimize } \sum_{i \in T, j \in A} \left( y_{tj} - \hat{y}^{i+1}_{tj} \right)^2 \\
\text{subject to } y \in S_y.
\]

Two major advantages of this program \( Q \) for finding a new feasible \( y \) are:
(i) When \( y \in S_y \) is of the form \( \{ y_{tr} \in S_{ytr}, \text{ for all } t \text{ and } r \} \), then program \( Q \) immediately decomposes into subprograms, of the form \( Q_{tr} \): minimize \( \{ \sum_{j \in J} (y_{tj} - \hat{y}_{tj})^2 \mid y \in S_{ytr} \} \), for each control point in each period \( t \).
(ii) When each \( S_{ytr} \) is of the form \( S_{ytr}^* = (P.5)-(P.7) \) set out in Section 2.1 above, then program \( Q_{tr} \) further reduces to the following easy-to-solve programs \( Q^*_{tr} \) for each \( r \) and \( t \).

\[
\text{(Q.0)} \quad \text{minimize } \sum_{j \in J} \frac{1}{2} (y_{tj} - \hat{y}_{tj})^2 \\
\text{subject to } \{ \theta \}, \sum_{j \in J^*} y_{tj} = l_t \quad \text{(Q.1)}
\]

\[
\{ \psi_{tj} \geq 0 \}, \quad y_{tj} \leq y^+_{tj} \quad \text{for all } j \in J^*_t \quad \text{(Q.2)}
\]

\[
\{ \phi_{tj} \geq 0 \}, \quad -y_{tj} \leq -y^-_{tj} \quad \text{for all } j \in J^*_t \quad \text{(Q.3)}
\]

Proposition 6. Program \( Q^*_{tr} \) can be rewritten as

\[
\text{(Q.0)} \quad \text{minimize } \sum_{j \in J^*} \sum_{k \in J^*} \frac{1}{2} (a_{tjk} y_{tj} - \hat{y}_{tjk})^2 \quad \text{(Q.0)}
\]

subject to \( \{ \theta \}, \sum_{j \in J^*} y_{tj} = l_t \quad \text{(Q.1)}
\]

\[
\{ \psi_{tj} \geq 0 \}, \quad y_{tj} \leq y^+_{tj} \quad \text{for all } j \in J^*_t \quad \text{(Q.2)}
\]

\[
\{ \phi_{tj} \geq 0 \}, \quad -y_{tj} \leq -y^-_{tj} \quad \text{for all } j \in J^*_t \quad \text{(Q.3)}
\]

where the \( a_{tjk} \)'s are from (P.7).

Remark. Program \( Q^*_{tr} \) has the advantage that it is simpler to solve than \( Q^*_{tr} \), since it contains only the variables \( \{ y_{tj}, j \in J^*_t \subset J_t \} \), whereas program \( Q^*_{tr} \) contains the potentially much larger set, \( \{ y_{tj}, j \in J_t \} \). Having solved \( Q^*_{tr} \) we can immediately compute the remaining variables \( \{ y_{tj}, j \in J_t, j \in J^*_t \} \) from (P.7).
Proof. The objective function of $Q_{r^*}$ can be rewritten as $\sum_{j \in J_r} \sum_{k \in J_j} \frac{1}{2} (y_{t_k} - \hat{y}_{t_k})^2$. The variables $\{y_{ij}, j \in J_r, j \notin J^*_r\}$ appear only in the objective function and on the left-hand side of (P.7). Using (P.7) to substitute for these variables in the objective function eliminates (P.7) and reduces $Q_{r^*}$ to $Q_{r^{**}}$. □

An efficient special purpose algorithm for solving a single constraint quadratic program such as $Q_{r^{**}}$ is set out in Appendix D of Kennington and Helgason (1980). They assume bounds $y^+ \geq y \geq y^-$ To put $Q_{r^{**}}$ in this form we simply shift the origin, by substituting $y_{ij} = y_{ij} - y_{ij}$ for $y_{ij}$ throughout program $Q_{r^{**}}$. This reduces the bounds $y^+ \geq y \geq y^-$ in $Q_{r^{**}}$ to $y^+ \geq y \geq y^-$.

6.2. Step sizes

Step 4 of Algorithm A2 above uses a sequence of step sizes, $\delta_1, \delta_2, \ldots$. In the literature on subgradient algorithms these step sizes have been defined using a prespecified sequence of constants, $\tilde{\delta}_1, \tilde{\delta}_2, \ldots$. (Kennington and Helgason, 1980). Three of these step size definitions are: (1) $\delta_i = \tilde{\delta}_i$, or (2) $\delta_i = \tilde{\delta}_i/(\alpha^T\alpha)^{1/2}$, or (3) $\delta_i = \tilde{\delta}_i(z(y) - \bar{z}_{v^P})/(\alpha^T\alpha)$, where $\bar{z}_{v^P}$ is an estimate of (or lower bound on) the optimal value of the objective function of VP, where $\alpha = [\alpha_{ij}]$. The sequence of constants $\tilde{\delta}_1, \tilde{\delta}_2, \ldots$, are defined to satisfy (a) $\delta_i > 0$ for all $i$, (b) $\lim_{i \to +\infty} \delta_i = 0$, and (c) $\sum_{i=0}^{\infty} \delta_i = +\infty$.

**Proposition 8.** The Algorithm A2 set out above converges to an optimal solution of problem VP if step sizes (1) or (2) above are used.

Proof. A proof of convergence for minimizing a convex function over a non-empty convex compact set is found in Kennington and Helgason (1980). The proof extends to VP recalling that (as noted above) VP = minimize $\{z_{v^P}(y) | y \in S_y\}$ and that $z_{v^P}(y)$ is convex (Proposition 4) and $S_y$ is a non-empty convex compact set. □

Held, Wolfe and Crowder (1974) and Kennington and Shalaby (1978) have used step size (3) above, and a convergence proof is given in Kennington and Helgason (1980).

7. Initial solutions

In Step 1 of the first iteration of each of the algorithms (A1 and A2) set out above, we have to choose a starting point $y^0 \in S_y$. Ideally $y^0$ should be a good initial estimate of the unknown optimal solution of program VP. Such an estimate may be available from previous runs of VP, or from the existing or past values of the control $y$, or from engineering standards or experience.

However, in the absence of such a ‘good’ estimate, any $y^0 \in S_y$ will suffice for initialization. We can obtain a $y \in S_y = \{(P.5)-(P.7)\}$ by a simple one-pass computation as follows. Choose a set of $y_{ij}$’s, so as to satisfy (P.5), (i.e., $\sum_{j \in J_r} y_{ij} = l_i$) and so that each $y_{ij}, j \in J^*_r$, divides its permissible range ($y_{ij}^- \leq y_{ij} \leq y_{ij}^+$) in the same proportions. Thus:

**Proposition 9.** A $y^0$ satisfying $y^0 \in S_y = \{(P.5), (P.6)\}$ is given by the following weighted average (convex combination) of $y_{ij}$ and $y_{ij}^+$:

$$y_{ij}^0 = (1 - f_{ir}) y_{ij}^- + f_{ir} y_{ij}^+, \quad (\star)$$

where $f_{ir} = (l_i - \sum_{j \in J_r} y_{ij}^-)/(\sum_{j \in J_r} (y_{ij}^+ - y_{ij}^-))$. We note that $0 \leq f_{ir} \leq 1$, if the constraints (P.5)–(P.6) have any feasible solution.

**Remark.** (P.5)–(P.6), and hence this proposition, considers only the $y_{ij}$’s such that $j \in J^*_r$. To obtain the remaining $y_{ij}$’s (i.e., for $j \notin J^*_r$, $j \in J_r$) simply substitute the $y_{ij}^0$’s from (\star) into the equality form of (P.7).
Proof. Substituting the above definition of $f_{tr}$ into (*) and summing over all $j \in J_r^*$ yields (P.5), hence (*) satisfies (P.5). Also, by (*), $y_{0j}$ is a weighted average (convex combination) of $y_{kj}$ and $y_{0j}$, hence $y_{0j}$ satisfies (P.6), if $0 \leq f_{tr} \leq 1$. It remains only to show that $0 \leq f_{tr} \leq 1$, as follows. Summing (P.6) over $j \in J_r^*$ and substituting in (P.5) gives $\sum_{j \in J_r^*} y_{ij} \leq (\sum_{j \in J_r^*} y_{ij} = l_j) \leq \sum_{j \in J_r^*} y_{ij}$. Subtracting $\sum_{j \in J_r^*} y_{ij}$ from each side of these two inequalities and dividing through by $\sum_{j \in J_r^*} (y_{ij} - y_{ij})$ gives $0 < f_{tr} < 1$. 

Two additional alternative strategies for choosing an initial set of controls $y^0 \in S_y = [(P.5)-(P.7)]$ are as follows.

(1) Let $y^0 = P([b_j y^+_i])$ where $y^+_i$ is the upper bound on $y_{ij}$ and $P(\cdot)$ is the projection operator (program Q) from Section 5.1.

(2) Let $y = [y^+_i]$. Solve FP to obtain an initial solution $(z^0_{ij}, x^0_{ij}, x^0_{ij}, x^0_{ij}, x^0_{ij})$. Set $y^0 = P([x^0_{ij}/b_j])$, where $P(\cdot)$ is again the projection program Q.

8. Computational experience

The algorithms described in Sections 5 and 6 were implemented in FORTRAN 77 and a series of test problems solved to gain computational experience. For comparison, all problems were also solved using a good LP code, which is the natural alternative. To solve and reoptimize the minimum cost network subproblems which occur in both algorithms we used the set of routines MODFLOW (Ali and Kennington, 1989). The experiments were performed on a SUN Sparcstation running SunOS UNIX. All computing times reported here are in CPU seconds on that computer.

The network. The spatial network used to generate test problems is shown in Figure 2. Nodes 1 through 12 are demand points. Nodes 13 through 21 are intersections. Demands generated at the demand points travel to the destination, which is at intersection 17. The arrows represent queues at intersections: there are 32 queues in all. Using this spatial network, five multiperiod models of the form VP were generated, having 20, 40, 60, 80 and 100 time periods respectively (see Table 1).

The demands. The exogenous demands $D_{ij}$ were generated so that congestion builds up to a peak and falls off again. Each $D_{ij}$ is the sum of two components. The first component is $B_j$ in the base period $t - 1$, increases by $wB_j$ (we set $w = 1.2$) per period up to the peak period $t = p$, and thereafter declines by $wB_j$ per period up to period $e = 2p - 1$. Demands are zero in periods $t = e, \ldots, T$. In computational experiments we used five different sets of base demands $\{B_j, j = 1, \ldots, 12\}$ randomly selected between 2
and 22, i.e. \( B_j \in U[2, 22] \). Also, to avoid bias in the results due to smooth growth or decline of demand, we added a further small random disturbance \( u_{itj} \in U[0, 4] \) to the demand for each period \( t \) and demand point \( j \). Thus
\[
D_{itj} = B_{itj} + u_{itj} + \begin{cases} 
( t-1)wB_{itj}, & 1 \leq t \leq p, \\
( e-t)wB_{itj}, & p < t \leq e.
\end{cases}
\]
The demands were rounded down to the nearest integer. For problems of size 20, 40, 60, 80 and 100 periods respectively, we set \( p = 8, 18, 27, 36 \) and 43 and \( e = 15, 35, 53, 71 \) and 85.

For each of the five problem sizes we have five different versions defined by the five randomly selected demand patterns or seeds \( \{B_j + u_{itj}, j = 1, \ldots, 12; t = 1, \ldots, e\} \).

**Other data.** All time periods are of equal length, the travel time between intersections is \( \tau_{ij} \) = 1 periods, the travel costs and storage/waiting costs are \( c_{itj} \) = 1 and \( c_{itk} \) = 1 unit/period respectively. The minimum and maximum ‘green’ times for each arc are \( y_{itj} = 0.1 \) and \( y_{itj}^+ = 0.7 \) of a period respectively. The arc throughput capacities at all intersections except 17 are \( b_{ij} = 500 \). Thus, for all intersections except 17, the minimum and maximum throughput capacities are \( b_{ij} y_{itj} = 100(0.1) = 10 \) and \( b_{ij} y_{itj}^+ = 100(0.7) = 70 \) respectively: for intersection 17 these are \( 500(0.1) = 50 \) and \( 500(0.7) = 350 \).)

The above pattern of demands and capacities were chosen so that congestion (tight capacity constraints) would be concentrated at intersections 14, 16, 18 and 20, the bottleneck intersections: there is one capacity constraint per intersection per period. Scaling the demand data by a load factor allows us to increase the level of congestion for any given model (Experiments 2 and 3 below).

Below we report the results of our computational experiments. We focus the discussion on the subgradient algorithm (SUB) since in our initial experiments it performed significantly better than the Benders algorithm. In a few problems which we solved with the Benders algorithm it took significantly longer than the subgradient algorithm, though much less time than using LP. For example, for the 20 period model (Table 1) with seed 2, our implementation of the Benders algorithm took 90.13 secs compared to 20.43 secs taken by the subgradient algorithm and 278.56 secs by LP. Also, Benders terminated further from the exact LP solution, 0.24% away as compared to 0.03% away for the subgradient algorithm. Hence in view of time and space limitations we concentrate on the subgradient algorithm here.

**Initial solutions.** To obtain these we tried two different methods. These are:

(a) Allocate intersection capacity equally to all queues at the intersection.

(b) Set \( y = \{y^+_{it}\} \) where \( y^+_{it} \) is the upper bound on \( y_{itj} \). Using this, solve FP to obtain an initial solution \( (x^0_{FP}, x^0_{ij}, x^0_{ij}, x^0_{ij}) \). Set \( y^0 = P(x^0_{ij}/b_{ij}) \) where \( P(\cdot) \) is again the projection program \( Q \).

Method (b) invariably outperformed method (a) and is used in all computations in this section.

**Step sizes.** We tried the three different methods of generating step sizes stated in Section 6.2. In each case we used the following sequence of constants: \( \delta_i = 2.0 \) for \( i = 1, \ldots, 32 \); \( \delta_i = 1.0 \) for \( i = 33, \ldots, 48 \); \( \delta_{i-0.5} \) for \( i = 49, \ldots, 56 \); \( \delta_i = 0.25 \) for \( i = 57, \ldots, 60 \); \( \delta_61 = \delta_62 = 0.125 \); \( \delta_{63} = 0.0625 \). All results are reported at the end of 63 iterations. In step size (3) of Sections 6.2 (i.e., \( \delta_i = \delta_i((\bar{z}(y) - \bar{z}_{VP})(\alpha^T \alpha)/2) \) the \( \bar{z}_{VP} \) used

---

**Table 1**

Size of test problems

<table>
<thead>
<tr>
<th>Problem size (Time periods)</th>
<th>Network size</th>
<th>LP size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Arcs</td>
</tr>
<tr>
<td>20</td>
<td>1464</td>
<td>2447</td>
</tr>
<tr>
<td>40</td>
<td>3004</td>
<td>5107</td>
</tr>
<tr>
<td>60</td>
<td>4544</td>
<td>7767</td>
</tr>
<tr>
<td>80</td>
<td>6084</td>
<td>10,427</td>
</tr>
<tr>
<td>100</td>
<td>7624</td>
<td>13,087</td>
</tr>
</tbody>
</table>
was the objective function value of problem VP with the green times set at their maximum values. Even though this is a quite poor estimate of \( z_{VP} \), step size (3) performed best among all the step sizes considered. Step size (1) performed well on smaller problems, but not as well on the larger problems.

We carried out three sets of experiments to test the performance of the subgradient algorithm. Three measures of performance are reported for each experiment, namely:

(a) the solution time (in CPU seconds),
(b) the ratio of the solution time for the subgradient algorithm to the solution time using a good linear programming code, namely GAMS/MINOS (Brooke, Kendrick and Meeraus, 1988),
(c) the percentage difference from the LP solution.

**Experiment 1.** This experiment was designed to see how the algorithm performed for problems having different patterns of demand and different problem sizes. The 25 problems generated above (5 problem sizes by 5 demand patterns) were solved using the subgradient algorithm and also using LP.

The results are shown in Table 2. The algorithm performed very well: the solution times for the subgradient algorithm were between 13 and 42 times faster than the solution times using an LP code. Also, the percentage difference between the subgradient solution and the LP solution was small, ranging from 0.02% to 0.29%. The results were fairly similar for all five problem sizes and all five random demand seeds. [For example, let \( r \) be the ratio of the LP solution time to the subgradient solution time. With seed 1, \( r \) ranges from 15 to 42 (the widest range) and with seed 2, \( r \) ranges from 13 to 29 (the smallest range). For problem size 20, \( r \) ranges from 13 to 18 (the smallest range) and for problem size 100, \( r \) ranges from 20 to 39 (the largest range).]

### Table 2

<table>
<thead>
<tr>
<th>Seed</th>
<th>Problem size</th>
<th>LP time (secs)</th>
<th>SUB time (secs)</th>
<th>( \frac{LP \ time + SUB \ time}{SUB \ time} )</th>
<th>% Diff. from LP sol.</th>
<th>PROP1</th>
<th>PROP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed 1</td>
<td>20</td>
<td>243.00</td>
<td>15.49</td>
<td>15.69</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1561.57</td>
<td>52.16</td>
<td>29.93</td>
<td>0.09</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>5161.00</td>
<td>124.35</td>
<td>41.50</td>
<td>0.15</td>
<td>0.13</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>8580.70</td>
<td>270.85</td>
<td>31.68</td>
<td>0.28</td>
<td>0.27</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>9345.22</td>
<td>451.90</td>
<td>20.68</td>
<td>0.29</td>
<td>0.28</td>
<td>0.64</td>
</tr>
<tr>
<td>Seed 2</td>
<td>20</td>
<td>278.56</td>
<td>20.43</td>
<td>13.61</td>
<td>0.05</td>
<td>0.03</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1498.87</td>
<td>83.25</td>
<td>24.08</td>
<td>0.12</td>
<td>0.33</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>3498.40</td>
<td>145.48</td>
<td>24.08</td>
<td>0.12</td>
<td>0.33</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>7447.64</td>
<td>255.14</td>
<td>24.08</td>
<td>0.12</td>
<td>0.33</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>8396.10</td>
<td>322.75</td>
<td>24.08</td>
<td>0.12</td>
<td>0.33</td>
<td>0.73</td>
</tr>
<tr>
<td>Seed 3</td>
<td>20</td>
<td>293.47</td>
<td>16.89</td>
<td>17.36</td>
<td>0.05</td>
<td>0.03</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1759.00</td>
<td>63.29</td>
<td>27.79</td>
<td>0.20</td>
<td>0.28</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>3947.51</td>
<td>114.56</td>
<td>34.46</td>
<td>0.17</td>
<td>0.35</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>7774.71</td>
<td>187.05</td>
<td>41.56</td>
<td>0.10</td>
<td>0.38</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>7489.80</td>
<td>266.08</td>
<td>28.15</td>
<td>0.05</td>
<td>0.39</td>
<td>0.87</td>
</tr>
<tr>
<td>Seed 4</td>
<td>20</td>
<td>265.60</td>
<td>15.84</td>
<td>18.06</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1665.40</td>
<td>65.52</td>
<td>25.42</td>
<td>0.20</td>
<td>0.23</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>3858.69</td>
<td>119.96</td>
<td>32.17</td>
<td>0.11</td>
<td>0.34</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>6063.27</td>
<td>205.34</td>
<td>29.53</td>
<td>0.09</td>
<td>0.37</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>11303.77</td>
<td>284.02</td>
<td>39.80</td>
<td>0.06</td>
<td>0.38</td>
<td>0.86</td>
</tr>
<tr>
<td>Seed 5</td>
<td>20</td>
<td>285.00</td>
<td>17.59</td>
<td>16.20</td>
<td>0.00</td>
<td>0.04</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1326.02</td>
<td>74.20</td>
<td>17.88</td>
<td>0.20</td>
<td>0.30</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>3318.62</td>
<td>117.58</td>
<td>28.22</td>
<td>0.24</td>
<td>0.36</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>5903.80</td>
<td>188.08</td>
<td>31.39</td>
<td>0.13</td>
<td>0.38</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>6755.11</td>
<td>286.31</td>
<td>23.51</td>
<td>0.10</td>
<td>0.39</td>
<td>0.89</td>
</tr>
</tbody>
</table>

\( a \) PROP 1: Proportion of all constraints tight.

\( b \) PROP 2: Proportion of 'bottleneck' constraints tight.
Table 3
Solution times by problem size (Seed 2) for the same level of congestion (PROP2 = 0.9)

<table>
<thead>
<tr>
<th>Problem size</th>
<th>LP time (secs)</th>
<th>SUB time (secs)</th>
<th>LP time % diff. from SUB time</th>
<th>% diff. from LP solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>249.64</td>
<td>19.92</td>
<td>12.53</td>
<td>0.01</td>
</tr>
<tr>
<td>40</td>
<td>1297.12</td>
<td>67.96</td>
<td>19.09</td>
<td>0.05</td>
</tr>
<tr>
<td>60</td>
<td>3137.49</td>
<td>127.30</td>
<td>24.65</td>
<td>0.07</td>
</tr>
<tr>
<td>80</td>
<td>5343.44</td>
<td>189.20</td>
<td>28.42</td>
<td>0.04</td>
</tr>
<tr>
<td>100</td>
<td>7051.62</td>
<td>300.42</td>
<td>23.47</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 2 also reports the proportion of all constraints tight (PROP1) and the proportion of ‘bottleneck’ constraints tight (PROP2). This is a measure of network congestion and here happens to increase with the problem size.

Experiment 2. In Experiment 1, the congestion level (the proportion of constraints tight) increased with the problem size, due to the form of the demand data. It is therefore of interest to see how the performance of the algorithm would vary with problem size if the level of congestion is held (approximately) constant. We chose a congestion level defined by PROP2 = 0.9. However, recall that the congestion level is a characteristic of the solution, and is not known in advance. We therefore varied the congestion level by scaling the demand data by a load factor. Several load factors were tried (for each problem size) until solutions having a congestion level of PROP = 0.9 were achieved, for each problem size. The same seed (Seed 2) was used throughout.

The results are shown in Table 3. The performance of the algorithm relative to LP is much the same as in Experiment 1: the order of improvement over LP is between 12 and 28 times as compared with 13 to 29 times in Experiment 1 (with Seed 2).

Experiment 3. This experiment was conducted to examine the behaviour of the subgradient algorithm with increasing levels of congestion, holding the problem size and random seed fixed. The demand data for the 80 period model, with seed 2, was scaled by load factors ranging from 0.2 to 1.6, to generate increasing levels of congestion.

The results are shown in Table 4. An interesting pattern emerges. For both the subgradient algorithm and LP the solution times increase with the congestion level, up to about PROP2 = 0.4, and decline thereafter. This is intuitively reasonable: if few constraints are tight, or if almost all constraints are tight, then there are fewer options to consider, so that solution times are lower. Also, the relative performance

Table 4
Solution times for varying levels of congestion for the 80 period model (Seed 2)

<table>
<thead>
<tr>
<th>Load factor</th>
<th>PROP 1 (secs)</th>
<th>PROP 2 (secs)</th>
<th>LP time (secs)</th>
<th>SUB time (secs)</th>
<th>LP time % diff. from SUB time</th>
<th>% diff. from LP solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.00</td>
<td>0.00</td>
<td>5240.89</td>
<td>58.86</td>
<td>89.03</td>
<td>0.00</td>
</tr>
<tr>
<td>0.4</td>
<td>0.02</td>
<td>0.04</td>
<td>7329.90</td>
<td>99.51</td>
<td>73.66</td>
<td>0.09</td>
</tr>
<tr>
<td>0.6</td>
<td>0.17</td>
<td>0.38</td>
<td>7477.09</td>
<td>231.67</td>
<td>32.28</td>
<td>0.24</td>
</tr>
<tr>
<td>0.7</td>
<td>0.23</td>
<td>0.42</td>
<td>7834.78</td>
<td>271.40</td>
<td>28.87</td>
<td>0.27</td>
</tr>
<tr>
<td>0.8</td>
<td>0.30</td>
<td>0.66</td>
<td>7704.34</td>
<td>213.88</td>
<td>36.02</td>
<td>0.04</td>
</tr>
<tr>
<td>1.0</td>
<td>0.36</td>
<td>0.81</td>
<td>7447.64</td>
<td>255.14</td>
<td>29.19</td>
<td>0.10</td>
</tr>
<tr>
<td>1.2</td>
<td>0.39</td>
<td>0.87</td>
<td>5823.44</td>
<td>198.41</td>
<td>29.35</td>
<td>0.07</td>
</tr>
<tr>
<td>1.4</td>
<td>0.40</td>
<td>0.89</td>
<td>4820.88</td>
<td>213.70</td>
<td>22.56</td>
<td>0.12</td>
</tr>
<tr>
<td>1.6</td>
<td>0.41</td>
<td>0.91</td>
<td>5811.62</td>
<td>183.43</td>
<td>31.68</td>
<td>0.06</td>
</tr>
</tbody>
</table>
of the subgradient algorithm (the ratio of LP time to SUB time) is highest (89) when congestion is low. A major reason for this is that about 70% to 75% of the time taken by the subgradient algorithm is in the projection operation. This operation is fastest when there is little or no congestion (few constraints tight). The speed of the subgradient algorithm relative to LP declines fairly steadily as congestion increases, but the minimum performance (22 times faster than LP) is still very good.

9. Additional flow controls

9.1. Time-of-day controls

The variable time-sharing model VP allows the controls (service times or green-times) $y$ to be different in each time period. However, in practice it is likely that these controls be held fixed over (pre)specified sets of time periods. For example, there may be one set of controls for each of the following periods: the morning peak period, evening peak, daytime off-peak and night-time. In general, this can be included in the model as follows: divide the time periods in the model into $K$ subsets $T_1, T_2, \ldots, T_K$, and set, $y_{tj} = y_{t'j}$ for all $t \in T_k, t' \in T_k, k = 1, \ldots, K$, for all queues $j \in A$.

Substituting these equations into program VP dramatically reduces the number of $y_{tj}$ variables, and hence greatly reduces the number of variables and constraints in the 'non-network' part of program VP. In the extreme, if the service time allocation is held constant over all time periods, the number of non-network type constraints at each control point $r$ is reduced from $|T_r|$ to one.

9.2. Synchronization of controls

It may be important in practice to ensure that some control settings ($y$'s) at neighbouring control points $r$ bear some linear relationship to each other. For example, for a busy road traffic corridor we may wish to set the lengths of the 'green' times ($y$'s) at successive intersections equal to each other. Equal green times for a successive pair of nodes $j$ and $k$, can be implemented as follows. Set $y_{tj} = y_{(t+I_{jk})k}$, for all $t \in T_j$, where $I_{jk}$ is the time taken to traverse arc $(j, k)$ starting out from queue $j$ in period $t$. These constraints need not appear explicitly in program VP, since they can be used to eliminate $y_{tj}$'s from the problem by substitution.

9.3. Local area controls

It is likely in practice that at any one time we would want to consider changes in the flow controls for only a (small) subset of the controls points $R$ in the network. For example, traffic engineers or traffic controllers may want to regularly (re)optimize the controls for some new intersections, or for some existing critical intersections, while treating the controls for the rest of the network as given. This greatly reduces the number of $y_{tj}$ variables and the number of non-network constraints in program VP.

10. Concluding remarks

The above models and solution algorithms may be used in a decision support system for network flow control. We have set out (in Sections 5 and 6) two algorithms for solving the flow control problem VP, both based on repeatedly solving a fixed controls model for a sequence of feasible fixed controls. As already noted, an advantage of these algorithms is that if we stop at any iteration we have a feasible implementable (if suboptimal) solution. For example, even if we perform only a few iterations of a feasible directions solution algorithm, starting with controls ($y_{tj}$'s) equal to the current actual controls, then we have a new feasible solution to implement which is an improvement on the old (current) control
settings. This is important since stopping short of optimality may be necessary for large scale problems in order to save on computing costs, especially if the control program were to be used on a daily basis or used for real time control setting.

A further advantage of the above algorithms for solving program VP is that they can very naturally and easily be used interactively by an engineer or controller, to explore other control policies, and to answer ‘what if’ questions which might be difficult to incorporate explicitly into the basic model. For example, having seen an optimal solution the engineer may wish to experiment with introducing or deleting, say, one-way streets, or restricted turning movements, or other changes in traffic light arrangements. Such features (e.g., choice of one-way versus two-way streets) could be explicitly incorporated into the basic model formulation, by using zero–one integer variables. However, that approach may be prohibitively costly computationally, since there can be an enormous number of combinations of such options to consider and since we may not know in advance which are worthy of consideration. In contrast, once the engineer or other expert has seen an optimal feasible solution he is in a much better position to decide which additional control design changes appear worth exploring.

Appendix

In this Appendix we outline a special purpose algorithm for solving the single constraint quadratic programs \( Q^*_t \) in Section 6.1 above. The solution method is based directly on that set out in Appendix D of Kennington and Helgason (1980).

**Proposition 7.** (a) The solution \( y_{tr} = \{ y_{tj}, \text{ all } j \in J^*_t \} \) of program \( Q^*_t \) is unique. (b) Program \( Q^*_t \) is equivalent to the following problem: Find a value of \( \theta \) such that \( \sum_{j \in J^*_t} y_{tj}(\theta) = l_t \), where each \( y_{tj}(\theta) \) is defined as the following piecewise linear function of \( \theta \):

\[
y_{tj}(\theta) = \begin{cases} y_{tj}^+ & \text{if } \theta \leq \theta_{tj}^+, \\ (\bar{y}_{tj} - \theta)/\bar{a}_{tj} & \text{if } \theta_{tj}^+ > \theta > \theta_{tj}^-, \\ y_{tj}^- & \text{if } \theta_{tj}^- \leq \theta,
\end{cases}
\]

(Q.4)

where \( \bar{y}_{tj} = \sum_{k \in J} a_{tj} y_{tk}, \theta_{tj}^+ = \bar{y}_{tj} - \bar{a}_{tj} y_{tj}^+, \theta_{tj}^- = \bar{y}_{tj} - \bar{a}_{tj} y_{tj}^- \) and \( \bar{a}_{tj} = \sum_{k \in J} a_{tj}^2 \) are constants.

**Proof.** (a) Since program \( Q^*_t \) has a strictly convex objective function and a linear (hence convex) constraint set, it has a unique global optimum.

(b) Since \( Q^*_t \) has a convex objective function and linear constraints the Kuhn–Tucker conditions are both necessary and sufficient to characterize an optimum (solution) of \( Q^*_t \). The Kuhn–Tucker conditions for program \( Q^*_t \) consist of (Q.1)–(Q.3), and

\[
((\bar{a}_{tj} y_{tj} - \bar{y}_{tj}) + \theta + \psi_{tj} - \phi_{tj}) = 0 \quad \text{for all } j \in J^*_t,
\]

and complementary slackness of the pairs of inequalities in (Q.2) and (Q.3).

We can assume that \( y_{tj} < y_{tj}^+ \), since if \( y_{tj} = y_{tj}^+ \), then from (Q.2)–(Q.3) we have \( y_{tj} \) fixed (\( = y_{tj}^- \)), in which case \( y_{tj} \) can be eliminated from the set of variables in program \( Q^*_t \). Then from (Q.2)–(Q.3) there are exactly three possibilities for \( y_{tj} \), namely \( y_{tj}^- < y_{tj} < y_{tj}^+ \), \( y_{tj} = y_{tj}^- \) and \( y_{tj} = y_{tj}^+ \). We will show that these three cases yield:

\[
y_{tj} < y_{tj}^+ \Rightarrow y_{tj} = (\bar{y}_{tj} - \theta)/\bar{a}_{tj} \Rightarrow \theta_{tj}^+ > \theta > \theta_{tj}^-
\]

\[
y_{tj}^+ = y_{tj} \Rightarrow \theta_{tj}^+ \leq \theta
\]

\[
y_{tj} = y_{tj}^+ \Rightarrow \theta \leq \theta_{tj}^-.
\]

(*)
Case (i): $y_{ij}^+ < y_{ij}^* < y_{ij}^-$. The complementarity conditions in (Q.2) and (Q.3) imply $\psi_{ij} = 0$ and $\phi_{ij} = 0$, and substituting these in (KQ.1) yields $y_{ij} = (\tilde{y}_{ij} - \theta)/\tilde{a}_{ij}$. Substituting the latter in $y_{ij}^+ < y_{ij} < y_{ij}^-$ yields (*.1).

Case (ii): $y_{ij}^* = y_{ij}^-$. Since, by definition $y_{ij}^* < y_{ij}^+$ we have $y_{ij}^* = y_{ij} < y_{ij}^+$, hence from complementarity in (Q.2), $\psi_{ij} = 0$. Substituting $\psi_{ij} = 0$ and $\phi_{ij} \geq 0$ into (KQ.1) yields $y_{ij} = (\tilde{y}_{ij} - \theta)/\tilde{a}_{ij}$ and substituting the latter in $y_{ij}^* = y_{ij}^- < y_{ij}^+$ yields (*.2).

Case (iii): $y_{ij}^* = y_{ij}^+$. By an argument similar to that in case (ii) we can prove (*.3).

But note that (a) the set of three possibilities on the left-hand side of (*.1)--(*.3) above are mutually exclusive and collectively exhaustive (from (Q.2)--(Q.3)), and (b) the set of three possibilities on the right-hand side of (*.1)--(*.3) above are mutually exclusive. It follows immediately that the interference ‘⇒’ in (*.1)--(*.3) must apply both ways, i.e., ‘⇒’ becomes ‘⇔’ in (*.1)--(*.3). Part (b) of the proposition follows immediately. □

Thus program Q$_r^*$ can be solved by finding the value of $\theta$ which simultaneously solves equations (Q.4) and (Q.1). For a given $j \in J_r^*$, $y_{ij}(\theta)$ is piecewise linear and non-increasing in $\theta$, with two breakpoints, at $\theta = \theta_{ij}^+$ and $\theta = \theta_{ij}^-$ and $\theta = \theta_{ij}^-$. Hence, $\Sigma_{j \in J_r^*} y_{ij}(\theta) = h_r(\theta)$ is a piecewise linear function which is non-increasing in $\theta$ and has $2|J_r^*|$ breakpoints (where $|J_r^*|$ is the cardinality of $J_r^*$), at $\theta = \theta_{ij}^+$ and $\theta = \theta_{ij}^-$ for all $j \in J_r^*$. Figure 2 gives an example of $h_r(\theta)$ for a control point with only two control settings, i.e., $|J_r^*| = 2$. Note that in this case $h_r(\theta)$ has 4 breakpoints.

A method for solving $h_r(\theta) = l_r$, is to evaluate $h_r(\theta)$, starting with an initial value for $\theta$, and adjusting the value of $\theta$ (increasing or decreasing) at each iteration until $h_r(\theta^*) = l_r$. Since $h_r(\theta)$ is piecewise linear we need only evaluate it at its breakpoints, until we can bracket the value of $l_r$ between two neighboring breakpoints (i.e., $h_r(\theta_2) > l_r > h_r(\theta_1)$), and then interpolate to find $\theta^*$, the optimal value of $\theta$. In the case of road networks, $h_r(\theta)$ has at most only a few breakpoints, since there are only a few control settings ($j \in J_r^*$) at each control point $r$. The details of an algorithm to solve a quadratic program similar to $Q$ are set out in Appendix D of Kennington and Helgason (1980).

References


