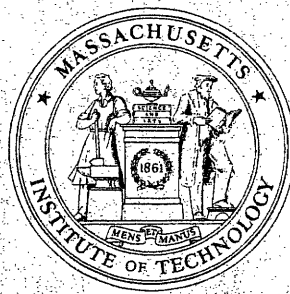


OPERATIONS RESEARCH CENTER

working paper



**MASSACHUSETTS INSTITUTE
OF TECHNOLOGY**

A LINEARIZATION AND DECOMPOSITION
ALGORITHM FOR COMPUTING
URBAN TRAFFIC EQUILIBRIA

by

Hedayat Z. Aashtiani

and

Thomas L. Magnanti

OR 122-83

June 1983

Preparation of this paper was supported in part by the Department of
Transportation under contract DOT-TSC-1058 and by the National Science
Foundation under grand 79-26225-ECS.

A LINEARIZATION AND DECOMPOSITION ALGORITHM
FOR COMPUTING URBAN TRAFFIC EQUILIBRIA

Hedayat Z. Aashtiani and Thomas L. Magnanti

Sloan School of Management, M.I.T.
Cambridge, MA 02139

Several years ago, we proposed and studied a linearization and decomposition algorithm for computing general urban traffic equilibria. The procedure applies to models that include link interactions, multiple commodities (e.g., modes of transit, or user classes), and with demand relationships general enough to model interactions between origin-destination pairs, such as travelers at an origin choosing trips from among several possible destinations. As reported in this paper, our computational experience with this algorithm has been very promising. It suggests that linearization and decomposition methods enable general problems to be solved with computational effort comparable to that of specialized techniques used for solving subclasses of equilibrium problems that have equivalent optimization formulations.

INTRODUCTION

Models of urban traffic equilibria, like similar network applications in communication, water resource planning, and spatially separated economic markets, are specially structured large-scale systems of nonlinear equations and inequalities. Often these models are stated in the form of equivalent fixed point models, nonlinear complementarity problems, or variational inequality formulations--all problem domains that have attracted considerable algorithmic development. Nevertheless, because they are so large, network equilibrium models have eluded solution by general purpose methods devised in any of these problem contexts. This lack of applicability has prompted researchers to search for new algorithms including special purpose methods that exploit the underlying network structure.

In this paper, we study a Newton-type linearization method and several decomposition procedures for solving a general traffic equilibrium model. Our computational results suggest that these solution strategies can be very effective; indeed, they require little more computational effort than algorithms for solving simpler classes of equilibrium models.

The first attempts at solving large-scale traffic equilibria problems appeared in the late 1960's and early 1970's. These methods grew out of seminal work by Beckman, McGuire and Winsten

(7), and related work by Samuelson (34) in a different context, who showed that certain classes of equilibria problems (basically, single mode problems with no link interactions and with travel demand functions with scalar arguments) could be formulated as equivalent nonlinear programming problems with linear network flow constraints. The first algorithms of this type were feasible direction algorithms with linear rates of convergence [Bruynooghe, Gilbert and Sakarovitch (10), Cantor and Gerla (12), Golden (23), and LeBlanc (28)]. More recent studies have considered acceleration procedures [Florian (20)] for these algorithms and second order methods [Bertsekas (8), Dembo and Klincewicz (18)].

As the models of urban traffic equilibria became more general, they no longer could be formulated as equivalent optimization problems. Rather, they gave rise to models formulated as nonlinear complementarity problems [Aashtiani and Magnanti (2)], as stationary point problems [Asmuth (6)] and as variational inequalities [Smith (36)]. Consequently, solution methods had to be tailored for one of these related formulations. This paper summarizes one of the first algorithmic studies of this variety, but which has appeared to date only in unpublished reports [Aashtiani (1), Aashtiani and Magnanti (3)]. Since these results were first developed, several insightful studies have established conditions under which the solution strategies of this paper are guaranteed to converge. Pang and Chan (33) and related results of Josephy (26) have established local convergence for the Newton-type linearization algorithm of our study and Dafermos (15-17), Florian (21) and Pang and Chan (32) have established convergence criteria for Gauss-Jordan and Gauss-Seidel type iterative methods like the decomposition methods that we have proposed. Basically, these results require monotonicity and some form of diagonal dominance of the underlying problem maps (though only locally for Newton's method). Other studies by Ahn (4), Ahn and Hogan (5) and Irwin and Yang (25) have established similar results in the context of energy distribution and production in spatially separated markets. In addition, Bertsekas and Gafni (9) have studied convergence properties of projection type methods. Rather than repeat the detailed analyses and proofs of convergence, we refer the reader to these original sources.

These studies now provide a firmer theoretical foundation for our computational procedures. With the interest that they have generated and with the continued attractiveness of algorithmic research related to our study, publication of this solution approach and of our empirical findings would seem to be timely.

Although we limit the discussion in this paper to urban traffic applications, the algorithms that we consider apply equally as well in a variety of other network equilibria applications.

1. THE MODEL

The traffic equilibrium model aims to predict flow patterns on links of an urban transit network. The basic ingredients of the model are demands for flow between specified origin and destination (O-D) pairs of the network and delay costs (time) on the paths p of the network.

Formally, for a given network $[N, A]$ with a set N of nodes and a set A of directed arcs, the model is a system of nonlinear equations and inequalities:

$$(T_p(h) - u_i)h_p = 0 \quad \text{for all } p \in P_i \text{ and } i \in I \quad (1.1a)$$

$$(T_p(h) - u_i) \geq 0 \quad \text{for all } p \in P_i \text{ and } i \in I \quad (1.1b)$$

$$\sum_{p \in P_i} h_p - D_i(u) = 0 \quad \text{for all } i \in I \quad (1.1c)$$

$$h_p \geq 0 \quad (1.1d)$$

$$u_i \geq 0. \quad (1.1e)$$

The decision variables for this model are

- h_p , the flow on path p of the network,
- h , the vector of path flows h_p with dimension r equal to the total number of O-D pair and path combinations,
- u_i , an accessibility variable, or shortest travel time, between O-D pair i ,
- u , the vector of accessibility variables u_i with dimension s equal to the number of O-D pairs.

The parameters for the model are

- I , the set of O-D pairs,
- P_i , the set of paths joining O-D pair i ,
- $D_i(u)$, the demand function for O-D pair i ;
 $D_i: R_+^s \rightarrow R_+$, and
- $T_p(h)$, the volume delay, or general disutility, function for path p ; $T: R_+^r \rightarrow R_+$.

We assume that at least one path joins each origin and destination pair. Note that any path p joins, and hence defines, a unique origin-destination pair $i \in I$.

The first two equations in (1.1) model Wardrop's traffic equilibrium law requiring that for any O-D pair i , the travel time (generalized travel time) for all paths, $p \in P_i$, with positive flow, $h_p > 0$, is the same and equal to u_i , which is less than or equal to the travel time for any path with zero flow. Equation (1.1c) requires that the total flow among different paths between any O-D pair i equals the total demand, $D_i(u)$, which in turn depends upon the congestion in the network through the shortest path variable u . Conditions (1.1d) and (1.1e) state that both flow on paths and minimum travel times should be nonnegative.

Before continuing, let us introduce notation that will streamline our discussion at points. Let $\Delta = [\delta(p, a)]$ denote a path-link incident matrix ($\delta(p, a) = 1$ if link a lies on path p). Then $v = \Delta h$ is a vector of link flows corresponding to the vector h of path flows. For each link $a \in A$, let $t_a(v)$ denote a delay function defined on the link and let $t(v)$ denote the vector-valued function with components $t_a(v)$. Similarly, let $D(u)$ denote the vector-valued function with components $D_i(u)$ and let $T(h)$ denote a vector-valued function with components $T_p(h)$. In addition, let $FL_i(h) = \sum_{p \in P_i} h_p$ denote the flow joining O-D pair i . Finally, let ℓ denote the number of links in the network and let Γ denote an incident matrix of paths versus O-D pairs.

With this notation, we can formulate an important special case of the equilibrium problem (1.1) -- an additive model in which

$$T(h) = \Delta^T t(v). \quad (1.2)$$

That is, the volume delay on path p is the sum of volume delays of the arcs in that path. Throughout the remainder of this paper we assume that the equilibrium model is additive, i.e., equation (1.2) applies.

The notation of the equilibrium model (1.1) is deceptively simple and somewhat disguises its generality. In particular, a judicious choice of network structure permits the formulation to model a wide range of equilibrium applications including multi-modal transit, multiple classes of users, and destination or origin choice. To model multi-modal situations, we might view the equilibrium as occurring on an extended network with a distinct component for each mode of transit. [Dafermos (14) and Sheffi (35) adopt this approach as well]. The component networks might be identical copies of the underlying physical transportation network, as when autos and buses share a common street network. Since the delay $t_a(v)$ on links of the automobile component network depend upon the full vector v of link flows, the delay function can account for congestion added by buses sharing these links. Note, though, that the networks for each mode need not be the same. Consequently, bus routes might be fixed and subway links might be distinct from those of other modes.

Aashtiani and Magnanti (2) specify more details concerning the model's range of applications and demonstrate that only mild continuity assumptions are required to establish the existence of an equilibrium solution.

Notice that the first set of equations (1.1a) in the equilibrium formulation (1.1) state that the product of one of the problem variables h_p and a particular function $f_p(h,u) \equiv T_p(h) - u_i$ is zero. Moreover, by inequalities (1.1b) and (1.1d) both terms h_p and $f_p(h,u)$ in the product $h_p f_p(h,u)$ must be nonnegative. As such, the equilibrium model (1.1) is reminiscent of the well-known nonlinear complementarity problem:

$$\begin{aligned} x_i F_i(x) &= 0 \\ F_i(x) &> 0 & \text{for } i = 1, 2, \dots, m \\ x_i &\geq 0. \end{aligned} \quad (1.3)$$

In referring to this formulation, we let x be an m -dimensional vector with components x_i and $F(x)$ be the vector-valued function with m component functions $F_1(x), F_2(x), \dots, F_m(x)$.

Suppose that we let $g_i(h,u) \equiv FL_i(h) - D_i(u)$ for all $i \in I$ and replace equations (1.1c) in the equilibrium model (1.1) with the constraints

$$\begin{aligned} g_i(h,u) &\geq 0 \\ \text{and } u_i g_i(h,u) &= 0. \end{aligned} \quad (1.4)$$

Then the model becomes a nonlinear complementarity problem with the identifications

$$\begin{aligned} m &= r + s \\ x &= (h,u) \\ \text{and } F(x) &= (f_p(x) \text{ for all } p \in P_i \text{ and } i \in I, g_i(x) \\ &\text{for all } i \in I) \in R^m. \end{aligned}$$

If we make the mild assumption that the travel time $t_a(v)$ on each link a of the network is positive and that demand function $D_i(u)$ is nonnegative, then the equilibrium model (1.1) and the nonlinear complementarity version of the problem are equivalent. Essentially, these assumptions imply that the shortest travel time u_i between each origin-destination pair i is positive. Consequently, the conditions (1.4) imply that $g_i(h,u) = 0$, which is equation (1.1c). Aashtiani and Magnanti (2) prove this equivalence algebraically.

Our solution procedure will be cast as a method for solving this nonlinear complementarity version of the equilibrium model.

2. SOLUTION STRATEGY

The formulation of the traffic equilibrium problem as a nonlinear complementarity problem brings the entire theory of complementarity to bear on the problem, but has little algorithmic consequence. Usually transportation problems are far too large to be solved by available nonlinear complementarity algorithms [Kojima (27), Lüthi (30)]. For example, even a small problem with 100 O-D pairs and with only 10 paths joining each O-D pair contains more than 1000 variables,

whereas the largest nonlinear complementarity problem that general purpose codes can solve has on the order of 100 variables, and even then requires a few minutes of solution (CPU) time. Since our formulation of the traffic equilibrium problem has one variable for each path joining an O-D pair and the number of paths in a network explode combinatorially with the size of the problem, solution by general nonlinear complementarity algorithms would appear to be hopeless.

2.1 Decomposition

To overcome the computational burden imposed by the problem size, we propose an iterative decomposition scheme. In this procedure, we partition the set of variables $\{x_i; i \in I\}$ into a collection of mutually exclusive subsets I_1, \dots, I_M . Then corresponding to each subset I_J , we define a subproblem (SP_J) as follows:

$$(SP_J) \quad \begin{aligned} F_i(x)x_i &= 0 & \text{for all } i \in I_J \\ F_i(x) &> 0 & \text{for all } i \in I_J \\ x_i &\geq 0 & \text{for all } i \in I_J. \end{aligned}$$

In this formulation all of the components of x are fixed except those x_i with $i \in I_J$. Obviously, each (SP_J) is a restricted version of the original nonlinear complementarity problem.

To solve the problem we might use a standard Gauss-Jordan or Gauss-Seidel type solution strategy. That is, given a solution x^q at some iteration of the procedure, we find x^{q+1} by solving (SP₁), (SP₂), ..., (SP_M) in order. Within each subproblem (SP_J) we let $x_i = x_i^q$ for all $i \in I - I_J$ in the Gauss-Jordan procedure and solve for x_i^{q+1} for $i \in I_J$. In the Gauss-Seidel procedure, we use the most recently computed values of each x_i at every step. That is, when solving (SP_J) we let $x_i = x_i^{q+1}$ for every $i \in I_K$ and $K < J$.

The efficiency of this procedure depends heavily upon how the set I is decomposed. Typically, it is best to collect together those variables that are most related to each other, so that the corresponding subproblem inherits the characteristics of the original problem. For example, in the case of destination choice demand functions, we might decompose the problem in terms of origins. We describe the decomposition criteria in more detail in later subsections after first looking more closely at the computational requirements for solving each subproblem.

2.2 Linearization

Note that a decomposition of the equilibrium problem in terms of its O-D pairs would seem to provide the smallest subproblems that inherit the essential characteristics of the original problem. But, even for this decomposition, the number of variables, corresponding to the number of available paths joining the origin and destination would be so large that no nonlinear complementarity algorithm could be used directly to solve the subproblems. Although the number of paths with positive flow is usually small (on the order of 4 or 5) even by knowing these paths it

is still not efficient to use general purpose nonlinear complementarity algorithms, because they would require an enormous number of functional evaluations. For instance at each vertex in their grid search procedure, simplicial path methods must evaluate all of the link-volume delay functions.

This difficulty, which is in the nature of the nonlinear complementarity problem, can be overcome by introducing an iterative linearization scheme, which is a version of Newton's method for linear inequality and equality systems.

We define the linearized problem for the nonlinear complementarity problem (1.3) at a point \bar{x} as follows:

$$\begin{aligned} [F(\bar{x}) + (x - \bar{x})\nabla F(\bar{x})]x &= 0 \\ \text{(LCP)} \quad F(\bar{x}) + (x - \bar{x})\nabla F(\bar{x}) &\geq 0 \\ x &\geq 0. \end{aligned}$$

We can then attempt to solve the nonlinear complementarity problem by successive linearization. That is, given a feasible solution x^q at some iteration q , we define x^{q+1} as the solution to (LCP) linearized at x^q .

Clearly, (LCP) is a linear complementarity problem. As is well-known, whenever $\nabla F(x)$, the Hessian of $F(x)$, is a positive semi-definite matrix, complementary pivot methods [Cottle and Dantzig(13), Eaves (19), and Lemke (29)] would solve the problem efficiently. These algorithms can solve problems with 100 variables within a few seconds of CPU time. Therefore, if this iterative procedure gives us a "reasonable" solution after only a few linearizations, then it would be much faster than any general purpose nonlinear complementarity algorithm.

This technique has one significant advantage when applied to the traffic equilibrium problem: the linearized problem (LCP) is a traffic equilibrium problem with linear affine functions and inherits any special structure of original problem, but is much easier to solve. Nevertheless, even for this simplified linear problem there is no efficient algorithm for large-scale applications currently available in the transportation literature (for the general case when $\nabla t(v)$ or $\nabla D(u)$ are non-symmetric), even though the problem can be solved by linear complementarity algorithms.

Nonetheless, because the linearized problem is a traffic equilibrium problem, we can exploit the nature of the problem as being cast in terms of path flows. At each iteration, instead of requiring all paths in the problem formulation, we can include only those paths that have positive flows. This is possible because we can generate shorter travel time paths, if there are any, by using a shortest path algorithm at each iteration (see section 3). Therefore, the (LCP) is much smaller in size than the nonlinear complementarity problem and, consequently, much easier to solve, so that problems with 100 O-D pairs

could be solved easily without using any decomposition.

For the traffic equilibrium problem it is easy to see that $\nabla F(x)$ is positive semi-definite when both $\nabla t(v)$ and $-\nabla D(u)$ are positive semi-definite matrices. To see this, using the vector notation introduced earlier we have

$$x = (h, u), \quad v = \Delta h$$

and

$$F(x) = (\Delta^T t(\Delta h) - \Gamma u, \Gamma^T h - D(u)).$$

$$\text{Thus, } \nabla F(x) = \begin{bmatrix} \Delta^T \cdot \nabla t(\Delta h) \cdot \Delta & , & -\Gamma \\ \Gamma^T & , & -\nabla D(u) \end{bmatrix}.$$

Clearly $\nabla F(x)$ is a positive semi-definite matrix, because for any $\bar{x} = (h, u)$ and $\bar{v} = \Delta h$ we have:

$$\begin{aligned} \bar{x}^T \nabla F(x) \bar{x} &= (\bar{h}^T \Delta^T) \nabla t(\Delta h) (\Delta \bar{h}) - \bar{h}^T \Gamma u + \bar{u}^T \Gamma^T \bar{h} - \bar{u}^T \nabla D(\bar{u}) \bar{u} \\ &= \bar{v}^T \nabla t(v) \bar{v} + \bar{u}^T (-\nabla D(u)) \bar{u} \geq 0. \end{aligned}$$

2.3 Composite Strategy

The size of many applications met in practice, with hundreds or even thousands of O-D pairs, precludes the solution of successive linearizations. Therefore, despite its attractiveness, the linearization scheme cannot be applied directly to many real-life problems. We can, however, combine decomposition and linearization in a composite solution scheme:

General Scheme

Step 1 - Choose a starting point x^0 and set $q = 0$.

Step 2 - Set $J = 0$.

Step 3 - Set $J = J + 1$. If $J > M$, go to step 6. Otherwise, set $\bar{x}_J = x_J^q$ and set $qq = 0$.

Step 4 - Solve (LSP_J), linearized at \bar{x}_J^q , to find a new point called \bar{x}_J^{qq+1} .

Step 5 - Set $qq = qq + 1$. If \bar{x}_J^{qq} is a "reasonable" solution to (LSP_J), then go to step 3. Otherwise set $x_J^{q+1} = x_J^q$ and go to step 4.

Step 6 - Set $q = q + 1$. If x^q is a "reasonable" solution to the nonlinear complementarity problem, then stop. Otherwise, go to step 2.

We refer to this procedure as either the composite algorithm or the linearization algorithm.

In this algorithm description, (LSP_J) corresponds to the linearization of (SP_J) at \bar{x} , defined by replacing $F_1(x)$ with its linear approximation $F_1(x) + (x_j - \bar{x}_j) \cdot \nabla F_1(\bar{x})$ where x_j denotes the vector $(x_i; i \in I_J)$ and $\nabla F_1(\bar{x})$ denotes the vector $(\partial F_1(\bar{x}) / \partial x_k; k \in I_J)$.

We refer to each pass through every subproblem in the algorithm as a cycle (i.e., whenever we re-execute step 2) and refer to each solution of a linearized subproblem as an iteration.

Note that when $M = 1$, the composite scheme reduces to the linearization scheme, and when all functions are linear, it reduces to the decomposition scheme.

In the next section, where we describe implementation details of the algorithm, we show how to choose the starting point x^0 and give some practical criteria for assessing when a solution is "reasonable".

3. IMPLEMENTATION

3.1 ϵ - Approximate Equilibria

In solving any traffic equilibrium problem we will never compute a solution exactly. Rather, we require a convergence criteria that defines an approximate solution. Toward this end, for any $\epsilon > 0$, we say that a flow pattern h^* is an ϵ -approximate equilibria if it satisfies the conditions:

$$(\hat{u}_i^* - u_i^*) \leq \epsilon \quad \text{for all } i \in I \quad (A1)$$

$$|FL_i(h^*) - D_i(u^*)| / D_i(u^*) \leq \epsilon \quad \text{for all } i \in I \quad (A2)$$

where for all $i \in I$

$$u_i^* = \min_{p \in P_i} T_p(h^*) \quad \text{and} \quad \hat{u}_i^* = \max_{\substack{p \in P_i \\ hp > 0}} \{T_p(h^*)\} \leq \epsilon$$

The first condition (A1) guarantees that the percentage difference between the longest path with positive flow and the shortest path is less than ϵ for all O-D pairs. The second condition guarantees that the percentage difference between the flow $FL_i(h^*)$ and the demand $D_i(u^*)$ between any O-D pair i is less than ϵ for all O-D pairs. Sometimes we refer to ϵ as the accuracy of the solution.

When we are applying the decomposition (or linearization) scheme, it does not pay to solve each subproblem (or linearization) to within the ultimate accuracy ϵ , because the accuracy for any subproblem will be destroyed when another subproblem is solved. Therefore, it is better to start with a less stringent accuracy requirement and to decrease it until the ultimate accuracy is achieved. For example, we can start with an accuracy of $\delta^n \epsilon$ for some integer $n > 0$ and some $\delta > 1$. After achieving the accuracy $\delta^n \epsilon$, the algorithm continues to impose accuracy requirements $\delta^{n-1} \epsilon$, $\delta^{n-2} \epsilon$, . . . and finally, after n steps, accuracy ϵ . This feature increases the efficiency of the algorithm enormously.

3.2 Starting Solution

To find a starting solution to initiate the iterative algorithm, we can use an all-or-nothing assignment which finds, for each O-D pair i , the

shortest path p_i^0 when all links have zero flow, and assigns all of the generated demand to that path. That is, define u_i^0 as the cost of path p_i^0 when all arcs flows are equal to zero and assign $D_i(u^0)$ units of flow to p_i^0 .

Notice that in this all-or-nothing assignment, we assign the flow generated by the demand function to a shortest path for each O-D pair sequentially, without considering the effect of the congestion from the flow previously assigned. This might lead us to assign too much flow on those links that have low free-flow travel times. To avoid this, we can update the minimum travel times, u , after each assignment. Also, since the initial u^0 , as compared to the u at equilibrium, is small, and, since the demand functions are usually increasing, the all-or-nothing assignment procedure generates too much initial flow, far from the value at the equilibrium. To avoid this, we can assign only some fraction of the generated demands to the shortest paths. We have used this modified all-or-nothing assignment, with the choice of 0.5 for the fraction, in our computational results.

3.3 Path Generation

As we mentioned previously, when solving each linearization, we include only those paths that have positive flow; we refer to these paths as the set of working paths, denote P_i^W . We also refer to any solution (h^*, u^*) as an ϵ -approximate equilibrium with respect to the working paths if it satisfies conditions (A1) and (A2) with the sets P_i^W for all $i \in I$. To guarantee that this solution is an ϵ -approximate equilibria over all paths, that is, over the sets P_i for all $i \in I$, we must satisfy the following condition:

$$[u_i^* - \min_{p \in P_i} T_p(h^*)] / u_i^* \leq \epsilon \quad \text{for all } i \in I. \quad (A3)$$

To construct the set of working-paths, we start with the paths in the initial solution. We add any path that gives $\min_{p \in P_i} \{T_p(h^*)\}$ and that satisfies condition (A3) to the set of P_i^W . Also, we always delete any path with zero flow from the set of P_i^W to maintain the size of the working-path sets as small as possible.

Because we must apply the shortest paths so many times, once for each iteration, their computation becomes one of the most time consuming components of the linearization algorithm. Consequently, since most shortest path algorithms find all the shortest paths from one origin to all destinations simultaneously, we will typically want to apply a decomposition by origin. Thus, we do not put two O-D pairs with different origins into the same subproblem unless all the other O-D pairs with the same origins are included in the subproblem.

Moreover, as noted previously, we would like to avoid expending too much time in one subproblem finding a very accurate solution, which will later be destroyed. Instead, we prefer to spread our work to achieve, simultaneously, the same,

but relaxed, accuracy for all subproblems. This suggests that we test condition (A3) and generate a shortest path for each O-D pair only once in each cycle, rather than generating a new shortest path after each iteration (linearization). When no linearization change of flow takes place in one cycle, then the given accuracy has been achieved and condition (A3) is satisfied.

3.4 Decomposition

The traffic equilibrium problem is rich enough to permit various forms of decomposition. The selection from among the various options depends upon the size of the problem and the nature of the demand function. For reasons discussed in the previous sections and also based upon our intuition, we have decided to consider the following levels of decomposition:

- Level 1 - No decomposition,
- Level 2 - Decomposition by origin,
- Level 3 - Decomposition by O-D pair,
- Level 4 - Decomposition by O-D pair and mode.

Moving from level 1 to level 4, we expect to incur more cycles and less iterations within each cycle (because the subproblems become easier to solve). Therefore, it is not clear which level of decomposition is best in terms of efficiency. However, as the size of the problem increases we are forced to use the higher levels of decomposition. On the other hand, as the demand dependency increases, the lower levels of the decomposition will be preferred.

Generally, we will tend to choose level 1 whenever the demand function is completely dependent (i.e., the demand for the O-D pairs depends upon the full vector of accessibility variables). We will choose level 2 when we have a destination choice demand function. Level 3 will be our choice when we have only mode choice demand functions; otherwise, we select level 4. In each case, if the size of the subproblem does not permit us to use that level, we move to the next higher level of decomposition.

Notice that when there is no mode dependency in the demand function, decomposition by mode might be best as the first level of decomposition.

3.5 Summary

To see how the composite linearization and decomposition algorithm applies to the traffic equilibrium problem, in this section we briefly summarize the steps of the algorithm when using a decomposition by O-D pairs.

To reduce the number of applications of the shortest path algorithm, we use a two-level decomposition scheme. In this first level, we decompose the problem by origins and find the shortest path tree for each origin. Then for each origin, in the second level we decompose the

problem in terms of destinations to construct subproblems. In this way, we solve shortest path problems only once in each cycle.

To simplify the discussion, we consider the single mode case. For the multi-modal case, when using O-D decomposition, the steps of the algorithm remain unchanged except that everything is in a vector space corresponding to all modes. For example, each subproblem corresponds to one O-D pair and all possible modes between the O-D pair. The algorithm would be slightly different if we first decompose the problem in terms of modes and then with respect to origins.

The algorithm is a refinement of the skeleton description given in section 2.3. We start by finding the initial solution (h^0, u^0) by the modified all-or-nothing procedure described in section 3.2. The user also specifies several parameters: (i) $\epsilon > 0$ is the convergence parameter to be applied in the convergence criteria of section 3.1; (ii) $\epsilon_n = \delta^n \epsilon$ is a relaxed convergence criteria to be applied when solving the linearization and decomposition problems at any point. The parameter $\delta > 1$ specifies how quickly the algorithm will reduce the convergence criteria. The initial value of n of n is another user supplied parameter. When the solution satisfies (A1), (A2) and (A3) and is an ϵ_n -approximate solution, the algorithm then reduces n by 1 (until it equals 0) and begins step 2 again.

A solution is said to be "reasonable" in steps (4) and (6) if it satisfies convergence criteria (A1) and (A2) with P_i replaced by $P_i^W = \{p_i^q\}$ where p_i^q is the shortest path joining O-D pair i computed at h^q . [Here $x^q = (h^q, u^q)$.] We add p_i^q to P_i and delete any path p from P_i^W that becomes zero in the solution to h^q obtained when solving for O-D pair $J = \{i\}$.

To solve each subproblem, we will use the solution from previous steps as the starting solution. It is more reasonable, in the overall procedure, that we always use the most recently generated information. To do this, we update all of the data (including path flows, volume delays, minimum travel times, and so forth) whenever any change in the flow occurs. This Gauss-Seidel strategy is applied to the all-or-nothing assignment and to the decomposition and linearization schemes.

For this algorithm, the corresponding subproblem (SP_i) for O-D pair i with the set of working-paths P_i^W can be written in the form of (1.1) with P_i^W replacing P_i . In this formulation,

$$T_p(h) = \sum_{a \in A} \delta_{ap} \cdot t_a(v) \text{ for all } p \in P_i^W$$

and

$$v_a = F_a + \sum_{p \in P_i^W} \delta_{ap} \cdot h_p \text{ for all } a \in A$$

where F_a is the sum of the flows by other O-D pairs on link a . The linearization (LSP_i) of (SP_i) at (h, u) for $p \in P_i^W$ is obtained by replacing $T_p(h)$

by

$$(T_p(\bar{h}) + \sum_{p' \in P_1^W} W_{p'}(\bar{h}_{p'}, -\bar{h}_{p'}) \frac{\partial T_p(\bar{h})}{\partial h_{p'}})$$

and by replacing $D_1(u)$ by:

$$(\sum_{p \in P_1^W} W_p - D_1(\bar{u}) - (u_1 - \bar{u}_1) \frac{\partial D_1(\bar{u})}{\partial u_1})$$

Here for all $p, p' \in P_1^W$

$$\frac{\partial T_p(\bar{h})}{\partial h_{p'}} = \sum_{a \in A} \sum_{a' \in A} \delta_{ap} \cdot \delta_{a'p'} \frac{\partial t_a(\bar{v})}{\partial v_{a'}}$$

Although computation of the coefficient matrix for each (LSP₁) at each iteration looks formidable and time-consuming, there are efficient ways to perform these computations. Aashtiani (1), Aashtiani and Magnanti (3) specify further details on this point and describe the algorithm and its variants in greater detail.

3.6 Assumptions

From the computational point of view, the linearization algorithm only requires some mild assumptions. More restrictive assumptions might be needed, however, to guarantee convergence of the algorithm. These mild assumptions are

- i) The vector functions $t(f)$ and $D(u)$ are continuous and differentiable.
- ii) Both $t(f)$ and $-D(u)$ are monotone functions, i.e., $\nabla t(f)$ and $-\nabla D(u)$ are positive semi-definitive matrices.

It is easy to show that, for any form of decomposition discussed in section 3.4, the coefficient matrix associated with any (LSP₁) is positive semi-definite; as we have already noted this is a sufficient condition to solve (LSP₁) by linear complementarity algorithms.

For transportation applications, these are very mild assumptions that are valid for many of the demand and volume delay model met in practice.

4. COMPUTATIONAL RESULTS

We have tested the composite linearization and decomposition algorithm on a variety of small problems with many different demand models and on some larger problems using data generated by other researchers. In this section, we summarize the algorithm's performance on three of these test cases we have. Aashtiani (1) and Aashtiani and Magnanti (3) report on a broader set of computational studies.

To solve the linear complementarity problem we use Lemke's Algorithm [Lemke (29)], which is an efficient algorithm that can solve the problems with a few hundred variables within a couple of seconds. To find the shortest path trees, we

use the algorithm presented by Golden (24) which is based upon Bellman's method. This algorithm can solve problems with 1000 nodes and 5000 links in less than one second.

All of the programs have been run on an IBM 370/168 using the Fortran G compiler. Reported CPU times do not include I/O times.

Example 4.1 The network for this example has 9 nodes, 36 links, and 12 O-D pairs.

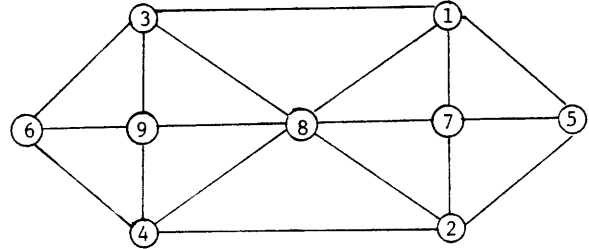


Figure 4.1 Network Configuration for Example 4.1

The volume delay functions are given as

$$t_a(v_a) = \alpha_a + 0.002 \cdot \beta_a \cdot v_a$$

where α_a and β_a are parameters ranging from 0 to 1. Steenbrink (37) specifies their values. There is, for each O-D pair $i-j$ with $i \neq j = 1, 2, 3, 4$, a fixed demand with values specified in table 4.1.

Origin	Destination			
	1	2	3	4
1	-	2000	2000	1000
2	200	-	1000	2000
3	200	100	-	1000
4	100	200	100	-

Table 4.1 Trip Table for Example 4.1

For a decomposition by O-D pair, and a choice of $\epsilon = 1.0$, $\delta = 5$, and $\bar{n} = 2$, the linearization algorithm solved this problem in 0.2 seconds of CPU time after 10 cycles and 45 linearizations. Notice that since the volume delay functions for this problem are linear, the equivalent minimization problem is a quadratic program. For different levels of accuracies, table 4.2 shows the value of

$$\sum_{a=1}^{36} \int_0^{v_a} t_a(x) dx,$$

which is equivalent to the objective value function for the minimization problem. Comparing these values to 16970, the objective value that Steenbrink found by quadratic programming methods, illustrates the accuracy of the linearization algorithm, even though the goal of the algorithm is not minimizing the objective value.

Note that solution with only 5% accuracy is as good in objective value as the solution found by Steenbrink.

Accuracy ϵ_n	$\sum_a v_a \cdot t_a(v_a)$	$\sum_{a=1}^{36} \int_0^{v_a} t_a(x) dx$
Initial	102,031.44	53,246.00
25%	27,369.15	17,198.93
5%	27,003.61	16,971.05
1%	26,965.25	16,958.24

Table 4.2 Total Travel Time for Example 4.1

Example 4.2: This test problem is taken from LeBlanc (28). The network consists of 24 nodes, 76 links, and 552 O-D pairs (all node pair combinations). There is a fixed demand between each O-D pair, and the volume delay functions are defined as

$$t_a(v_a) = \alpha_a + \beta_a \cdot v_a^4.$$

LeBlanc (28) specifies values for the parameters.

We applied the linearization algorithm to this test problem with the choice of $\epsilon = 1$, $\delta = 5$, $n = 2$, and used decomposition by O-D pairs. The algorithm terminated after 18 cycles and 564 linearizations, and required 3.32 seconds of CPU time to find solutions with 1% accuracy.

Table 4.3 shows the number of linearizations, total link travel time, and the maximum percentage change in the link flow after each cycle. In addition, it shows the maximum percentage change after each iteration for the Frank-Wolfe algorithm used by LeBlanc. These results demonstrate how fast the linearization algorithm converges and how it exhibits less of the tailing phenomenon that is typical of the Frank-Wolfe algorithm. In terms of computational time, the linearization algorithm required 2.15 seconds on an IBM 370/168 to achieve 5% accuracy, while the Frank-Wolfe algorithm required 10 seconds on the CDC 74 (notice that the IBM 370 is much faster than the CDC 74).

The accelerated convergence of the linearized algorithm might be expected. For problems like this that leave an equivalent optimization problem, linearization can be interpreted as making a quadratic approximation to the objective function (compare this with Newton's method for unconstrained minimization).

Cycle Number	Number of Linearizations	Total Link Travel Time	Max % Change in Link Flow	Max % Change in Link Flow by Frank-Wolfe
0	-	922.96	-	-
1	185	140.00	190.9	68.7
2	74	108.91	38.5	46.6
3	30	102.22	14.7	39.4
4	9	100.70	14.5	50.0
5	73	97.09	17.8	32.1
6	48	96.14	14.9	100.0
7	25	95.92	7.2	41.1
8	18	95.82	5.3	21.6
9	5	96.01	6.6	35.4
10	37	95.96	3.1	16.3
11	18	95.94	.8	25.0
12	14	95.94	.8	16.0
13	11	95.91	.8	13.9
14	8	95.92	.4	9.6
15	6	95.91	.5	11.4
16	2	95.92	.5	7.7
17	1	95.92	.5	11.2
18	0	95.92	.0	7.9

Table 4.3 Computational Results for Example 4.2

Example 4.2: This example is a moderately-sized test problem with real data from the city of Hull, Canada. It has been used by researchers in several other studies [e.g., Florian and Nguyen (22)]. The computational results for this example give some idea of how well the linearization algorithm performs, compared to the other algorithms, both in terms of convergence and efficiency.

The network has 155 nodes, 376 one-way links, 27 zones, and 702 O-D pairs (all possible zone combinations). There is only one mode of transportation (auto). The volume delay functions are given by the travel time function suggested in the Bureau of Public Roads traffic assignment manual (11), which has the form

$$t_a(f_a) = t_a^0 [1 + .15(v_a/c_a)^4]$$

with parameters t_a^0 specifying the free-flow time on link a and c_a specifying the nominal capacity of link a . Finally, there is a fixed demand between each O-D pair. The data for this problem is a slight modification of that used in Florian and Nguyen (22). In particular, we scaled the demand by a factor of 10, and this explains some differences between our results and those reported elsewhere.

When applying the linearization algorithms for this problem, we chose $\epsilon = 1$, $\delta = 5$, $n = 2$ and a decomposition by O-D pairs. The algorithm terminated after 20 cycles and 590 linearizations, and required 16.37 seconds of CPU time to find a solution with 1% accuracy. The maximum number of paths between each O-D pair with posi-

tive flow was 4 and the maximum number of links in the paths with positive flow was 44.

Table 4.4 shows the number of cycles and linearizations needed to reach different levels of accuracy. Also, it shows the computational times and the total link travel time,

$$\sum_a v_a \cdot t_a(v_a).$$

Accuracy ϵ	No. of Cycles	No. of Linearizations	CPU Time (sec)	Total Link Travel Time (min)
Initial				590,336.
25%	4	179	3.81	257,570.
5%	12	405	10.49	236,631.
1%	20	590	16.37	235,776.

Table 4.4 Computational Results for Example 4.3 with Fixed Demand

Nguyen (31) used the Convex-Simplex Method to solve the equivalent minimization problem for this problem. This algorithm required 42.16 seconds of CPU time on a CDC CYBER 74 to find a solution with an accuracy almost equivalent to 5%, as defined in section 3.1 (Nguyen has used a different criteria to define accuracy).

Florian and Nguyen (22) reported other computational times for both fixed demand and elastic demand variations of this problem for different numbers of O-D pairs, up to 421. For the case of 421 O-D pairs, their algorithm required 43.42 seconds of the CPU time on the CDC CYBER 74 to find a solution with 5% accuracy, as defined in section 3.1. The linearization algorithm required only 10.49 seconds on an IBM 370/168 for a problem with 702 O-D pairs.

As a second test with this problem, we used an elastic demand function with a linear functional form:

$$D_i(u_i) = b_i - a_i u_i \text{ for } i = 1, \dots, 702$$

where a_i and b_i have been selected randomly in a fashion similar to that reported in Florian and Nguyen (22). Table 4.5 shows the results of this test.

Accuracy ϵ	No. of Cycles	No. of Linearizations	CPU Time (sec)	Total Link Travel Time (min)
Initial				197,681.
25%	6	468	8.03	234,532.
5%	14	1542	11.21	234,344.
1%	20	2548	18.46	234,004.

Table 4.5 Computation Results for Example 6.5 with Elastic Demand

The results in Tables 4.4 and 4.5 shows that obtaining an equilibrium assignment with elastic demand requires only 15 percent more computational time than the fixed demand case. Although the number of linearizations increases four fold, the computational time does not grow nearly as rapidly. This is because the computational time for the linearization algorithm depends more on the number of cycles than the number of linearizations. Therefore, the algorithm is not highly dependent upon the type of the demand function.

The algorithm presented by Florian and Nguyen, which is based upon Benders Decomposition Method, required 54.13 seconds on the CDC CYBER 74 to achieve 5% accuracy, even with only 421 O-D pairs. This is almost 25 percent more than the time for the fixed demand problem as compared with a 15 percent increase in time for the linearization algorithm. For approximately 702 O-D pairs, the Florian-Nguyen algorithm required 80 seconds on the CDC CYBER 74 to achieve 5% accuracy. In contrast, the linearization algorithm required only 11.21 seconds on an IBM 370/168. Of course, the IBM 370/168 is faster than the CDC CYBER 74, though not more than four times as fast. Also notice that they have used the optimizing FTN compiler, while we have used the FORTRAN G compiler.

Because of different operating environments, it is difficult to compare these algorithms. At the very least, these results show that the linearization algorithm is as fast as, if not faster than, the specialized algorithms presented by Florian and Nguyen, which are among the fastest existing algorithms for solving the traffic equilibrium problem. Moreover, the linearization algorithm has its own important advantage, which is the generality of the algorithm compared to any algorithm based upon minimization techniques.

5. STORAGE REQUIREMENTS AND DATA STRUCTURES

Implementation of the linearization algorithm requires three types of storage--the computer program, the problem data, and path flow information. This section comments briefly on different types of storage schemes that we have tested.

The computer program itself uses 10K words of computer memory, including the main program, the linear complementarity program, the shortest path algorithm, and all other subroutines.

As described in table 5.1, storing all of the problem data requires at most $8|A| + 6|N| + 10|I|$ words of memory as a function of $|A|$ the number of links, $|N|$ the number of nodes, and $|I|$ the number of O-D pairs. This storage includes the network structure, the tree for the shortest path algorithm, the link flows and path flows, parameters of the volume delay and demand functions (such as the data t_a^0 , c_a , a_i , and b_i for the city of Hull problem with elastic demand), and, finally, vectors to store the update values for $t(f)$, $\nabla t(f)$, $D(u)$, and $\nabla D(u)$.

Array Size	Arrays
IAI	Start-node, End node, Link flow f , Link travel time $t(f)$, $Vt(f)$, 2 arrays for parameters of the volume delay function, and one dummy array for the shortest path subroutine.
INI	4 arrays (including a pointer to the first link starting at each node, each node's predecessor node, and level from the root) to represent the shortest path tree, shortest travel time from an origin to all nodes, and one array for nodes to be scanned.
III	Origin node, destination node, 4 arrays for path flows h (those with positive flow), minimum travel time u , total demand $D(u)$, $VD(u)$, and number of used paths between each O-D pair.

Table 5.1 Memory Requirements to Store the Problem Data

This data can easily be kept in memory on a main-frame computer such as the IBM 370 even for networks with 10,000 links. For implementations with limited storage, we can reduce storage requirements to $6|A| + 6|N| + 8|I|$ at the expense of additional computation time by reevaluating $t(v)$, $Vt(v)$, $D(u)$, and $VD(u)$ whenever they are needed, instead of storing this data.

The last, and major, requirement for storage is the path information. If we assume that the maximum number of paths with positive flows is $M1$ and the maximum number of links in path is $M2$, then for each O-D pair we might allocate a fixed space equal to $M1 * M2$ to store arc-path chains. Therefore, storing all path information requires $M1 * M2 * |I|$ words of memory. For the choice of $M1 = 4$, $M2 = 50$, and $|I| = 700$, as is the case of example 4.2, the storage requirement would be 140K words, which can be stored in core on an IBM 370. Nevertheless, most computers will charge for using extra core storage.

To make the linearization algorithm capable of solving larger problems and, also, to reduce the cost of using extra core storage, we must reduce the storage requirement for the path information. There are two ways to achieve this goal—modifying the data structure for storing path information, or using out-of-core storage (e.g., disk or tape).

5.1 A Modified Data Structure

Previously, we allocated a fixed space equal to $M1 * M2$ for each O-D pair. We refer to this scheme as S1. In practice, though, not all O-D pairs have the maximum number $M1$ of paths with positive flow (for the city of Hull, there are only 947 paths with positive flow which on average is only 1.35 paths with positive flow joining each O-D pair), and not all of the paths have $M2$

links. Therefore, there is a great deal of unused allocated storage. However, this fixed storage scheme has an advantage: it permits fast access to groups of paths with the same O-D pairs or with the same origins (this is important for the decomposition schemes that we use). In fact this allocation stores the paths in a sequential order in terms of O-D pairs and origins.

Since we are generating the paths, it is not easy to keep this sequential ordering when we use a variable number of paths $M(1, i)$ and a variable path length $M(2, i)$ for each O-D pair i . However, by introducing some pointers we can still have good access to a group of paths. Naturally, the access time to any path would increase beyond that required by the fixed space scheme. Thus, there is a tradeoff between CPU time and storage utilization.

We have implemented the linearization algorithm with a new storage scheme S2 with variable $M(1, i)$ and fixed $M2$. Two pointers are sufficient for locating any path; we call these FIRST and NEXT. For O-D pair i , FIRST(i) indicates the location of the first path joining O-D pair i , and NEXT(p) indicates the location of the next path with the same O-D pair as path p . NEXT(p) is set to zero when p is the last path joining an O-D pair.

When applied to the city of Hull example with elastic demand functions, the new allocation scheme requires 60K words to store the path information, compared to 140K for the original scheme. This savings in storage reduced the in-core storage cost by \$2.54, but increased the CPU cost by \$0.61. Thus, the total savings in cost was \$1.93.

This modification makes the algorithm capable of solving larger traffic assignment problems and, at the same time, reduces the total running cost. An even better improvement might be achieved by allocating variable space for $M(2, i)$, the number of links in the path, as well.

5.2 Out-of-Core Storage

In theory, we can always use out-of-core storage. But how efficiently? This depends upon the choice of record size, number of times we need to access to the records, and, more importantly, upon the order we need to access the records (sequential or random).

A fixed space allocation scheme could store all of the paths with the same O-D pair and same origin sequentially. Thus, a decomposition scheme by O-D pair or by origin, would require, within each cycle of the algorithm, only sequential access to all of the records. This is not the case for the modified data structure. For this reason, it seems that an out-of-core storage facility is more appropriate for the fixed storage scheme than for the modified scheme. Now the question is what is the optimal record size in terms of total computer running cost.

We have examined two different record sizes. For the first of these, denoted S3, we chose the record size equal to $M1 * M2$ so that we could fit all of the path information corresponding to each O-D pair in one record. This storage scheme required only 45K words of in-core storage which is a reasonable requirement for any small computer; this compares to 174K words of storage for the original storage scheme S1. Therefore, the variable space storage scheme is practical for solving much larger traffic assignment problems. However, the total running cost increases, as we might expect, and by a factor of 3. The first factor contributing to this cost increase is the accessing cost to out-of-core storage. The second factor is the increase in in-core storage cost, even though this run requires less in-core storage, because the program remains idle during the accessing process. Finally, the last factor is the increase in the CPU cost due to substantial swapping to transfer data from the out-of-core storage to the arrays in the memory. To reduce the total running cost for S3, we must decrease the number of accesses to the out-of-core storage. To accomplish this, we need to increase the record size.

For the next storage scheme S4, we chose the record size equal to $M1 * M2 * d$, where d is the maximum number of destinations associated with each origin (for the city of Hull, $d = 36$). In other words, we store all of the paths originating at each origin in one record. When equipped with this storage scheme, the algorithm required an order of magnitude less accesses to out-of-core storage for the Hull problem (from 15,620 to 1,333); therefore, the in-core storage cost and accessing cost decrease substantially when using S4 instead of S3 (by a factor of 5 1/2). Moreover, S4 required only 7K additional words of in-core storage.

On the other hand, S4 incurred almost 67% more CPU time and cost than did S3. (The total cost for S4 was about 40% less than S3 for the Hull problem). With S4, at each step the algorithm must substitute all of the information from the one record into the corresponding array in memory. Unfortunately, there is a great deal of non-usable information in most records, because not all of the O-D pairs have $M1$ paths with positive flow. In contrast, for the implementation S3, we only transfer the paths with positive flows, which are located at the top of the record, and discard the remainder of the record. This is not possible for the implementation S4.

Overall, among the four schemes, when there is no in-core storage limitation, the modified data structure scheme S2 would be best in terms of total cost, and the original scheme S1 would be best in terms of speed. When storage is limited, scheme S4 seems to be best in terms of total running cost, as least for the computational testing on the Hull example. We suspect that as problem size increases even more, the savings of S4 relative to S3 might become even more pronounced.

ACKNOWLEDGEMENT

Preparation of this paper was supported in part by the Department of Transportation under contract DOT-TSC-1058 and by the National Science Foundation under grant 79-26225-ECS.

REFERENCES

1. AASHTIANI, H., "The Multi-Modal Traffic Assignment Problem," Ph.D. Dissertation, Operations Research Center, MIT, Cambridge, MA (May 1979).
2. AASHTIANI, H. and T. MAGNANTI, "Equilibria on a Congested Transportation Network," SIAM Journal on Algebraic and Discrete Methods 2 (1981) 213-226.
3. AASHTIANI, H. and T. MAGNANTI, "Modeling and Computing Extended Urban Traffic Equilibria," U.S. Department of Transportation, Draft Report (September 1979).
4. AHN, B., Computation of Market Equilibria for Policy Analysis: The Project Independence Evaluation System (PIES) Approach, Garland Publishing Inc., New York, NY (1979).
5. AHN, B. and W. HOGAN, "On Convergence of the PIES Algorithm for Computing Equilibria," Operations Research 30 (1982) 281-300.
6. ASMUTH, R., "Traffic Network Equilibrium," Technical Report SOL-78-2, Stanford University, Stanford, CA (1978).
7. BECKMAN, M., C. MCGUIRE and C. WINSTEN, Studies in the Economics of Transportation, Yale University Press, New Haven, CT (1956).
8. BERTSEKAS, D., "Algorithms for Optimal Routing of Flow in Networks," Coordinated Science Lab. Working Paper, University of Illinois at Champaign-Urbana (1976).
9. BERTSEKAS, D., and E. GAFNI, "Projection Methods for Variational Inequalities with Application to the Traffic Assignment Problem", Technical Report, Laboratory for Information and Decision Systems, MIT, Cambridge, MA (1980).
10. BRUYNNOOGHE, M., A. GIBERT, and M. SAKAROVITCH, "Une Méthode d'Affectation du Traffic," in Fourth Symposium of the Theory of Traffic Flow, Karlsruhe (1968).
11. BUREAU OF PUBLIC ROADS, "Traffic Assignment Manual," U.S. Department of Commerce, Urban Planning Division, Washington, DC (1964).
12. CANTOR, D. and M. GERLA, "Optimal Routing in a Packet Switched Computer Network," IEEE Transactions on Computers 10 (1974) 1062-1068.
13. COTTLE, R. and G. DANTZIG, "Complementary Pivot Theory of Mathematical Programming," Linear Algebra and It's Applications. (1968) 103-125.
14. DAFERMOS, S., "An Extended Traffic Assignment Model with Applications to Two-Way Traffic," Transportation Science, (1971) 366-389.
15. DAFERMOS, S., "An Iterative Scheme for Variational Inequalities," Mathematical Programming, to appear.

16. DAFERMOS, S., "Traffic Equilibrium and Variational Inequalities", Transportation Science 14 (1980) 42-54.
17. DAFERMOS, S., "Relaxation Algorithms for the General Asymmetric Traffic Equilibrium Problem", Transportation Science 16 (1982), in press.
18. DEMBO, R. and J. KLINCEWICZ, "An Approximate Second-Order Algorithm for Network Flow Problems with Convex Separable Costs," Working Paper No. 21, Yale School of Organization and Management, New Haven, CT (1978).
19. EAVES, B., "The Linear Complementarity Problem," Management Science 17 (1971) 612-634.
20. FLORIAN, M., "An Improved Linear Approximation Algorithm for the Network Equilibrium (Packet Switching) Problem," Publication #57, Centre de recherche sur les transports, University of Montreal, Montreal (July 1977).
21. FLORIAN, M. and H. SPIESS, "The Convergence of Diagonalization Algorithms For Asymmetric Network Equilibrium Problems," Transportation Research, to appear.
22. FLORIAN, M. and S. NGUYEN, "A Method for Computing Network Equilibrium with Elastic Demand," Transportation Science, 8 (1974) 321-332.
23. GOLDEN, B., "A Minimum Cost Multi-Commodity Network Flow Problem Concerning Imports and Exports," Networks, 5 (1975) 331-356.
24. GOLDEN, B., "Shortest-Path Algorithms: A Comparison," Operations Research 24 (1976) 1164-1168.
25. IRWIN, C. and C. YANG, "Iteration and Sensitivity for a Spatial Equilibrium Problem with Linear Supply and Demand Functions," Operations Research 30 (1982) 319-335.
26. JOSEPHY, N., "Newton's Method for Generalized Equations," Technical Report 1965, Mathematics Research Center, University of Wisconsin, Madison, WI, (1979).
27. KOJIMA, M., "Computational Methods for the Nonlinear Complementarity Problem," Department of Administration Engineering, Keia University, Yokohama, Japan (1973).
28. LEBLANC, L., "Mathematical Programming Algorithms for Large Scale Network Equilibrium and Network Design Problems," Ph.D. Dissertation, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL (1973).
29. LEMKE, C., "Bimatrix Equilibrium Points and Mathematical Programming," Management Science 11 (1965) 681-689.
30. LUTHI, H., "A Simplicial Approximation of a Solution for the Nonlinear Complementarity Problem," Mathematical Programming 9 (1975) 278-293.
31. NGUYEN, S., "An Algorithm for the Traffic Assignment Problem," Transportation Science 8 (1974) 203-216.
32. PANG, J. and D. CHAN, "Gauss-Seidel Methods For Variational Inequality Problems Over Product Sets," Working Paper, Graduate School of Industrial Administration, Carnegie-Mellon, Pittsburgh, PA (1982).
33. PANG, J., and D. CHAN, "Iterative Methods for Variational and Complementarity Problems," Mathematical Programming, to appear.
34. SAMUELSON, P., "Spatial Price Equilibrium and Linear Programming," American Economic Review 42 (1952) 283-303.
35. SHEFFI, Y., "Transportation Network Equilibrium with Discrete Choice Models," Ph.D. Dissertation, Department of Civil Engineering, MIT, Cambridge, MA (1978).
36. SMITH, M., "Existence, Uniqueness and Stability of Traffic Equilibria", Transportation Research 13B (1979) 295-304.
37. STEENBRINK, P. A., Optimization of Transport Networks, Wiley, London (1974).