Faster Path-Based Algorithm for Traffic Assignment

R. JAYAKRISHNAN, WEI K. TSAI, JOSEPH N. PRASHKER, AND SUBODH RAJADHYAKSHA

A fresh look at the arguments against path-enumeration algorithms for the traffic assignment problem is taken, and the results of a gradient projection method are provided. The motivation behind the research is the orders of magnitude improvement in the availability of computer storage over the last decade. Faster assignment algorithms are necessary for real-time traffic assignment under several of the proposed advanced traffic management system strategies, and path-based solutions are preferred. The results show that gradient projection converges in one-tenth of the iterations of the conventional Frank-Wolfe algorithm. The computation time improvement is of the same order for small networks but is reduced as the network size increases. The computer implementation issues are discussed carefully, and schemes to achieve a 10-fold speedup for larger networks are also provided. The algorithm was used for networks of up to 2,000 nodes on a typical computer workstation, and certain data structures that save storage and solve the assignment problem for even a 5,000-node network are discussed.

As is well known traffic assignment is the process of finding the flow pattern in a given network with a given travel demand between the origin-destination (O-D) pairs. Equilibrium assignment finds flow patterns under user equilibrium, when no driver can unilaterally change routes to achieve better travel times. Optimal assignment determines the flow patterns such that the total travel time cost in the network is minimum, usually under external control. Assignment has long been an essential step in the transportation planning process. See Sheffi (*1*) for detailed discussions on traffic assignment.

Real-time traffic assignment could be a part of potential advanced traveler information systems or advanced traffic management system (ATMS) strategies [see Kaysi and Ben-Akiva (*2*) for a discussion of such strategies]. The applications of network assignment in such real-time contexts could be in the on-line estimation of O-D demand matrices or in an on-line dynamic assignment framework. The conventional approach, used in planning applications, is to solve the assignment problem with the Frank-Wolfe algorithm (F-W), also known as the convex-combinations algorithm (*1*). However real-time applications typically require path-based solutions [see Mahmassani and Peeta (*3*)], which are not available with the link-flow-based F-W. Faster convergence is also a very desirable feature for a real-time algorithm.

In this paper we report our investigation of the Goldstein-Levitin-Poljak gradient projection algorithm formulated by Bertsekas (*4*). This algorithm falls under the set of algorithms called *path-enumeration algorithms*, which have traditionally been discarded by transportation researchers as too memory intensive and slow for large networks. In light of the orders of magnitude improvement in the availability of computer memory in recent years, we believe that such algorithms deserve a fresh look. In this paper we describe our implementation of the algorithm and the extremely encouraging results. We discuss the assignment formulation for the sake of completeness in the next section, and follow it by a literature review and further qualitative discussions of the algorithms. We then proceed to discuss the computer implementation issues. We conclude with results on the comparative performances of the algorithms and pointers for future research.

R. Jayakrishnan and S. Rajadhyaksha, Department of Civil and Environmental Engineering, University of California, Irvine, Irvine, Calif. 92664. W. K. Tsai, Department of Electrical and Computer Engineering, University of California, Irvine, Irvine, Calif. 92664. J. N. Prashker, Technion-Israel Institute of Technology, Haifa, Israel.

STATIC USER EQUILIBRIUM ASSIGNMENT PROBLEM

As is well known the static assignment user equilibrium problem is stated as

$$\min Z = \sum_a \int_0^{x_a} t_a(\omega)\, d\omega \tag{1}$$

subject to the demand and nonnegativity constraints given by

$$\sum_k f_k^{rs} = q_{rs} \;\forall\; r, s, k \in K_{rs} \tag{2}$$

$$f_k \geq 0 \tag{3}$$

where

x_a = flow on link a (sum of the flows on the paths sharing link a),
$t_a(\omega)$ = cost (travel time) on link a for a flow of ω,
f_k^{rs} = flow on path k connecting origin r and destination s,
q_{rs} = total traffic demand between r and s, and
K_{rs} = set of paths with positive flow between r and s.

The above problem or variations of the same problem have appeared in some of the recently proposed dynamic assignment algorithms with time-varying demands such as the bilevel algorithm of Janson (*5*) and the instantaneous dynamic assignment algorithm of Ran et al. (*6*). Note also that a system optimal assignment problem reduces to a user equilibrium problem with transformed (marginal) cost functions (*1*), and hence algorithms developed for user equilibrium assignment are applicable to the system optimal assignment as well.

REVIEW OF RELEVANT LITERATURE

Extensive work on network optimization approaches has been done to address the traffic equilibrium assignment problem. A detailed discussion of the conventional approaches to it is presented by Sheffi (*1*). A detailed study by Lupi (*7*) showed that F-W is superior to most other algorithms. Nagurney (*8*) compared F-W with the algorithm of Dafermos and Sparrow (*9*) and found the latter to be in general more efficient. There has been some research to improve the efficiency of F-W. Arezki and Van Vliet (*10*) presented an analytic implementation of the PARTAN technique as applied to F-W and presented results indicating improvements over the original algorithm. LeBlanc et al. (*11*) and Florian et al. (*12*) showed how the PARTAN method could be applied to the traffic network equilibrium assignment problem and showed improved convergence in real networks. Weintraub et al. (*13*) investigated a method of improving the convergence of F-W by making modifications on the step size. One of the most recent improvements was by Larsson and Patriksson (*14*) who employed simplicial decomposition approaches to the original F-W.

Algorithms for assignment based on Benders decomposition have also been developed by Florian (*15*) and Barton et al. (*16*). The projection-based algorithms that have been developed in the past include those by Pang and Chan (*17*) and Dafermos (*18*). There has, however, not been much drawn from the advances made in the parallel field of optimal flow assignment in computer communication networks. The gradient projection algorithm popularized by Bertsekas and Gallager (*19*) is one such algorithm that we investigate in the present study. In computer communication the networks are usually smaller than the large urban networks in which traffic assignments are carried out for planning purposes, and this may have been why transportation researchers have not paid enough attention to the research in that field.

SELECTION OF ALGORITHMS: HISTORICAL PERSPECTIVE

The choice of an appropriate algorithm for the traffic equilibrium assignment problem is guided by several criteria for selection, depending on the specific needs of the application, with the overriding criteria often being the memory requirements of the algorithm and its speed of convergence. These concerns become increasingly critical as the network size increases. We provide the following discussion to reveal the motivations behind the research.

The conventional choice for the traffic assignment problem so far has been F-W. This choice has been guided largely by the memory requirements criterion. Since F-W at any one iteration deals with only a single path between each O-D pair, its storage requirements are well within the capabilities of most ordinary computers. However it has the drawback that typically the convergence becomes very slow as it approaches the optimal solution. It shows a tendency to flip-flop as it gets close to the optimum. The reason is that the algorithm is driven more by the constraint corners and less by the actual descent direction of the objective function surface once it is close to the solution. This was not considered a serious problem in earlier applications of the traffic assignment because the problem was being addressed from a transportation planning viewpoint. Under this scenario assignment is used for forecasting purposes when the O-D demand data themselves are derived by using the extrapolation of current values or statistical models. This inherent inexactness in the process renders the exact estimation of link volumes unimportant, and so practitioners are often content to stop the algorithm after a few iterations when it reaches within 5 to 10 percent of the solution. The memory requirement criterion was also of importance. When F-W was introduced to the transportation field in the late 1970s computers were incapable of handling the larger memory requirements of path-enumeration algorithms. Recent advances in computing equipment have placed vastly increased computing power in smaller and smaller machines. Computer workstations with 16 megabytes of storage are only about as expensive as a personal computer with 128 kilobytes of storage in the mid-1980s and have more storage than the largest mainframe computers of the late 1970s. Given these possibilities it is important that we rethink our choice of traffic assignment algorithms and take advantage of the technological edge provided by current and future improvements in computer hardware.

Another aspect of F-W is that it does not automatically find the intersection turning movements. This has traditionally been found by microcoding the intersections with specific turning links, which usually increases the numbers of nodes and links in the networks considerably. Path-flow solutions automatically provide such turning counts without any microcoding of the network, thus keeping the network sizes small. However if separate flow-cost functions are to be used for turning movements, microcoding may be necessary with path-based algorithms also.

The recent advent of the intelligent vehicle-highway system (IVHS) brings up the need for real-time traffic assignments with requirements different from those for planning applications. Such assignments may be part of dynamic assignment frameworks or real-time O-D demand estimation frameworks. Faster convergence becomes important, and path-based solutions may be necessary. Moreover increasing emphasis is placed on estimating fuel consumption and modeling air quality over specific routes. Solutions based on path flows provide speed profiles over the network paths that are conceivably useful (although still very approximate) for such applications. Link-flow solutions from F-W are much poorer in this regard.

GRADIENT PROJECTION ALGORITHM

The gradient projection algorithm (GP) is extensively used in computer communication networks, in which path-flow solutions are essential for optimal flow routing. However the networks in these applications are typically much smaller than urban traffic networks and the path-enumeration issues have not been serious concerns. Moreover the network structures are also somewhat different in these two applications. We adapted the basic Goldstein-Levitin-Poljak GP formulated by Bertsekas (*4*) to the traffic assignment problem and concentrate here on the practical convergence and computer implementation issues.

In contrast to F-W, which finds auxiliary solutions that are at the corner points of the linear constraint space, GP makes successive moves in the direction of the minimum of a Newton approximation of a transformed objective function. The objective function includes the demand constraints also, and thus the feasible space for gradient projection is defined only by the nonnegativity constraints, as opposed to both nonnegativity and demand constraints in the case of F-W. Should the move to the minimum in the negative gradient direction result in an infeasible solution

point, a projection is made to the constraint boundaries. As a result of the redefinition of the problem, infeasibility occurs only when a variable violates the nonnegativity constraint, and thus the projection is easily accomplished by making that variable zero. We describe this in detail below.

The formulation of the algorithm focuses on the traffic demand constraints.

$$\sum_{k \in K_{rs}} f_k = q_{rs}$$

where K_{rs} is the set of paths (with positive flow) between origin r and destination s.

If we express the shortest-path flows $f_{\bar{k}_{rs}}$ in terms of other path flows

$$f_{\bar{k}_{rs}} = q_{rs} - \sum_{\substack{k \in K_{rs} \\ k \neq \bar{k}_{rs}}} f_k \tag{4}$$

the standard optimization problem (equations 1 through 3) can be restated as

$$\min \tilde{Z}(\tilde{f}) \tag{5}$$

subject to

$$f_k \geq 0 \; \forall \; f_k \in \tilde{f} \tag{6}$$

where $\tilde{Z}$ is the new objective function, and $\tilde{f}$ is the set of non-shortest-path flows between all of the O-D pairs.

For each O-D pair while at any feasible (nonoptimal solution) a better solution can be found by moving in the negative gradient direction. This gradient is calculated with respect to the flows on the non-shortest paths (which are the only independent variables now), and a move size is found by using the second derivatives with respect to these path-flow variables. Once the flows on these non-shortest paths are updated the flow on the shortest path is appropriately updated so that the demand constraint is satisfied.

The gradient of the objective function written in terms of the non-shortest-path variables can be found using

$$\frac{\partial \tilde{Z}}{\partial f_k} = \frac{\partial Z}{\partial f_k} - \frac{\partial Z}{\partial f_{\bar{k}_{rs}}} \text{ where } k \in K_{rs} \text{ and } k \neq \bar{k}_{rs} \tag{7}$$

which results from the definition of $\tilde{Z}$. Thus each component of the gradient vector is the difference between the first derivative lengths of a path and the corresponding shortest path (*14*). In the case of equilibrium assignment the objective function is in terms of integrals and the first derivative lengths are simply the path costs at that flow solution.

A small increase in the flow on a path k results in an equal decrease in the flow on the corresponding shortest path. This results in no change in the flow on the common part of the two paths. Thus the second derivative is simply the sum of the second derivative lengths of the links on either path k or path $\bar{k}_{rs}$, but not both. A small increase in the flow on path k causes an equal decrease in the flow on the shortest-path $\bar{k}_{rs}$. The flows on the common links on these paths do not change. The increase in flow on the other links on path k causes positive second derivatives. The decrease in flow on the other links on $\bar{k}_{rs}$ also causes positive second derivatives as it increases the negative first derivatives. Once the second derivatives of $\tilde{Z}$ with respect to each path flow are calculated, we assume a diagonal Hessian matrix, and the inverse of each second derivative gives an approximate quasi-Newton step size for updating each path flow.

For the remainder of the paper when we refer to the *first derivative lengths* we mean the first derivatives of the objective function, which is composed of link costs at specific path flows [i.e., $t_a(x_a)$]. Similarly, *second derivative length* refers to the second derivative of the objective function and is composed of first derivatives of link costs (i.e., $\partial t_a / \partial x$ at $x = x_a$).

On the basis of the above discussion the gradient projection algorithm can be formalized as follows:

Step 0—Initialization: Set t_a equal to $t_a(0)$, $\forall a$ and perform all-or-nothing assignments. This yields path flows f_1^{rs}, $\forall$ r,s and link flows x_a^1, $\forall a$. Set iteration counter n equal to 1. Initialize the path set K_{rs} with the shortest path for each O-D pair rs.

Step 1—Update: Set t_a^n equal to $t_a(x_a^n)$, $\forall a$. Update the first derivative lengths d_k^n (i.e., path costs at current flow) of all of the paths in K_{rs}, $\forall$ r,s.

Step 2—Direction finding: Find the shortest-path $\bar{k}_{rs}^n$ from each origin r to each destination s on the basis of $[t_a^n]$. If different from all the paths in the existing path set K_{rs} (no need for path comparison here; just compare d_k^n), add it to K_{rs} and record $d_{\bar{k}_{rs}^n}$. If not tag the shortest among the paths in K_{rs} in $d_{\bar{k}_{rs}^n}$.

Step 3—Move: Set the new path flows.

$$f_k^{n+1} = \max\left[0, f_k^n - \frac{\alpha^n}{s_k^n}\left(d_{\bar{k}^n rs} - d_{\bar{k}_{rs}^n}\right)\right] \; \forall \; r, s, k \in K_{rs}, k \neq \bar{k}_{rs}^n$$

where

$$s_k^n = \sum_a \frac{\partial t_a^n}{\partial x_a^n} \quad \forall \; k \in K_{rs}$$

a denotes links that are on either k or $\bar{k}_{rs}$, but not on both, and α^n is a scalar step-size modifier (say, $\alpha^n = 1$).

Also,

$$f_{\bar{k}^n rs}^{n+1} = q_{rs} - \sum_k f_k^{n+1} \quad \forall \; k \in K_{rs}, k \neq \bar{k}_{rs}^n$$

Assign the flows on the trees and find the link flows x_a^{n+1}.

Step 4—Convergence test: If the convergence criterion is met, stop, or set n equal to $n + 1$ and go to Step 1.

It is better to keep α^n a constant (i.e., $\alpha^n = \alpha$, $\forall n$). It can be shown that given any starting set of path flows there exists an $\bar{\alpha}$ such that if $\alpha \in (0, \bar{\alpha})$ the sequence generated by this algorithm converges to the optimum (*1*), provided that the link-cost functions are convex. Our experience shows that α equal to 1 achieves a very good convergence rate, and all of the results in this paper use this value of α. The solutions reached are unique in terms of the link flows, but the path-flow solution, although it is optimal, is not necessarily unique [see Sheffi (*1*) for a discussion on why the path-flow solutions need not be unique].

A qualitative graphical comparison of GP and F-W is shown in Figure 1. In this case F-W moves in directions that are almost orthogonal to the descent direction, once it is close to the opti-

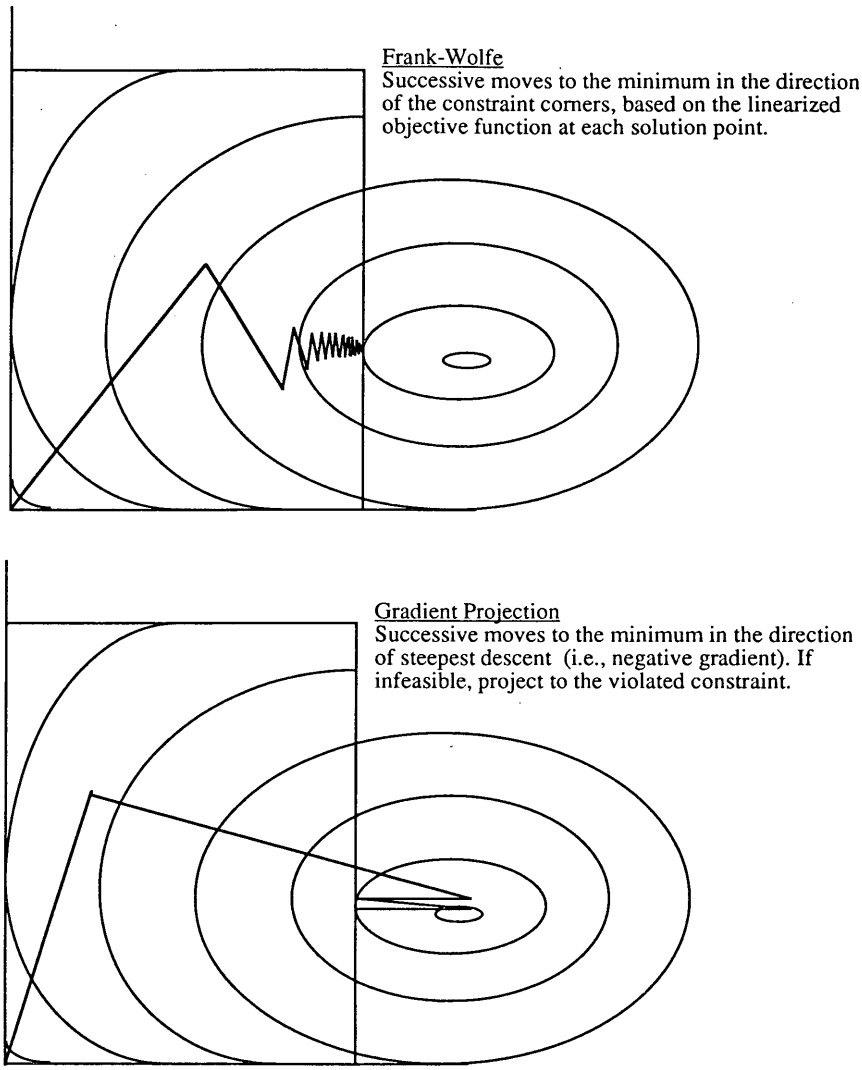

FIGURE 1 Comparison of GP and F-W.

mum, because the moves are toward constraint corners to avoid infeasibility. GP still moves in the descent direction when it is closer to the optimum. Note that this is just an example and is provided only to illustrate the qualitative reason behind the faster convergence of GP. The actual nature of the objective functions and the constraints in the network assignment context are quite different.

GP distributes flows from existing paths to the shortest path during every iteration, with different fractions of flow being taken out of the alternative paths between an O-D pair. A careful look at F-W shows that it also implicitly redistributes flows from alternative paths. However the fractions taken out from the paths are all same, and the path-flow solutions are never kept track of.

COMPUTATIONAL STORAGE CONSIDERATIONS

GP is a path-enumeration algorithm, and the paths need to be carefully stored to prevent memory problems. This is an issue that has rarely been addressed in the computer communication applications, but we address this here because we deal with traffic networks with large sizes. In GP one shortest-path tree is built during each iteration from each origin. The paths between an O-D pair are all generated during different iterations, and each path is part of one of the shortest-path trees built from the corresponding origin node. It is important not to store the paths as node lists but rather as predecessor trees (note that each iteration produces the shortest paths, which are invariably trees). This avoids double storage of the common portions of different paths found from each origin in each iteration. As we store one predecessor node number for each node, each tree in a network of N nodes requires N storage locations. This results in $N_o \cdot N$ storage locations in each iteration, where N_o is the number of origins. Thus the main memory requirement of the algorithm is of the order of $N_o \cdot N \cdot N_i$, where N_i is the number of iterations. We do not see the kind of combinatorial explosion that is expected to occur with path-enumeration algorithms. The fact that the paths in each iteration are part of trees is thus a very handy feature of GP.

However for F-W the memory requirements are fixed by network size and are not affected by the number of iterations until convergence. Typically its requirements are of the order of $N_o \cdot N$ because it stores only one tree (the shortest paths for the all-or-nothing assignment) in every iteration. Thus GP does have a storage disadvantage compared with F-W, but it is able to provide a richer solution for precisely the same reason because it gives us path flows, as opposed to the link flows provided in F-W.

The memory requirement is hardly a significant concern on the basis of our experience. With a simple predecessor array data structure we were able to run networks of up to 1,200 nodes with 22,300 O-D pairs on a SUN workstation. If only about 10 iterations of GP are attempted (which itself generally finds better solutions than 100 iterations of F-W, as our results shown in this paper indicate), we can run networks of more than 2,000 nodes. We briefly describe a new data structure that allows us to run networks of up to 5,000 nodes and 560,000 O-D pairs to 32 iterations. The purpose for attempting to run such networks is to show that the storage problem that kept researchers away from applying path-enumeration algorithms to larger networks is really not a problem anymore, at least in the case of GP.

The following describes an efficient data structure for larger network problems. The shortest-path trees built in successive iterations often have several identical branches. When these are stored as separate trees it results in a great amount of duplication of storage because we find that several nodes have the same predecessor arcs in successive trees. Rather than storing the predecessor arcs for all of the nodes in every iteration we store a shorter list of nodes for which the predecessor arcs change in an iteration (change being defined from the predecessor in the "anchor" iteration, say, the first one). The information regarding the iteration numbers in which the predecessor of each node changes is stored in terms of the bits of a number. A 1 in bit location i of this number for a node means that its predecessor changed in iteration i, and a 0 would indicate that no change occurred in iteration i. In a computer with 32 bit numbers, this lets us store the information with just N numbers. To find the predecessor for this node during, say, iteration j, we need to find the bit location of the last 1. This can be efficiently done at the hardware or software level and yields the appropriate iteration number, i. We go to the short list of changed predecessors corresponding to iteration i to find the predecessor for the node of concern. This approach will not achieve good computation time results unless it is carefully implemented, and we leave out the complicated implementation details in this paper. With this data structure we were able to reduce the storage requirement for the trees from the $N_o \cdot N \cdot N_i$ above, to about $N_o \cdot N \cdot C$, where C is about 5 to 10 for up to even 100 iterations (i.e., $N_i = 100$). This is because the predecessors of each node change only fewer than 10 times during 100 iterations of GP on the basis of our assignment runs on realistic traffic networks.

COMPUTATION TIME CONSIDERATIONS

A careful implementation is absolutely essential for GP to perform well. Because we found that the algorithm converges generally about 10 times faster than F-W in terms of the number of iterations, our intention was to ensure that GP achieves similar speed in computation time also. Although F-W requires no other operations of computational intensity comparable to that for the shortest-path determination during each iteration, GP has other procedures during its iterations that can be more time intensive than the shortest-path determination for larger networks. Our studies (as dicussed in the next section) show that GP converges 10 times faster than F-W for a 100-node network but only 60 percent faster for a 900-node network, although the number of iterations needed is still less than one-tenth. Because we found that almost all of the time in F-W is spent on the shortest-path routine for larger networks, this means that there are routines in GP whose computational intensities increase faster than that for the shortest-path routine as the network size increases. We identify three such key operations, and these must be carefully implemented: (a) assigning the flow on each path to the links along its length to find the total link flows, (b) finding the first derivative lengths for all of the paths between each O-D pair, and (c) finding the second derivative lengths for each pair. We have been successful in developing efficient schemes for the first two but have not been able to tackle the third problem without modifying the algorithm itself. The results that we provide in this paper are based on a program that includes only the techniques for the first derivative lengths. However we discuss all three aspects here to show the potential of the algorithm to show even better results than we have provided if our suggestions are implemented. Our early results with all three of the following procedures are indeed very encouraging.

Implementation of Flow Assignment Procedure

Implementation of the flow assignment procedure refers to assigning the path flow to all of the links on each path after the path flows are updated during each iteration. There are $N_o \cdot N_o$ O-D pairs in a network (where N_o is typically 10 to 20 percent of N), and the expected number of active paths between an O-D pair, N_p, is typically about 5 to 10 at convergence. Each path in a traffic network has roughly $O(N^{1/2})$ links on them. Thus this operation could be of $O(N_o \cdot N_o \cdot N^{1/2} \cdot N_p)$ effort in each iteration, if each path is considered independently (i.e., the link-level operations are repeated for paths that share common portions). In contrast to this an efficient implementation (say, using a heap) of a routine to find the shortest paths from all origins to all nodes results in $O(N_o \cdot N \log N)$ or better computational intensity. The flow assignment step becomes much more time-consuming for larger networks. We developed an efficient tree-traversal procedure to assign the flows instead of doing it path by path. The procedure starts from a leaf node of the tree and goes up assigning the flow on the links until it reaches a node where there is another branch with no flow assigned. Once the flows are added on all of the branches at a node we find the total flow that should be assigned on the predecessor link of the node and move up. This procedure goes only once over each arc in the tree, and hence it is an $O(N)$ operation for each tree (a maximum of four additions per arc in a typical traffic network). This results in only $O(N_o \cdot N)$ operations for all origins, all destinations, and all paths in each iteration. We leave out further details of this procedure for brevity. It suffices to say that this assigns flows of multiple paths sharing common portions without repeated calculations at the links. This is a significant improvement because the computations now do not depend on the number of paths or the number of nodes on the paths. The computational intensity drops to below that for the shortest-path determination with this method.

Finding First Derivative Path Lengths

In finding the first derivative path lengths the link costs are added up on different paths. Similar to the above, we need to avoid path-based computations of the $O(N_o \cdot N_o \cdot N^{1/2} \cdot N_p)$ order in this case also. Here we again perform a tree traversal procedure. We go up from any node in the tree until we reach the root (origin) node and add the link costs on the links to find the first derivative length to that node. Then we go up from that node once more (second pass), subtracting one link cost at a time to find the costs to each node along the way. The two passes are needed only because we cannot move down the tree when the tree is stored with predecessor representation. A threaded-tree storage will let us do a one-pass traversal, but additional overheads may be involved. Another option is to keep the tree traversal order right after each tree is built, but this requires as many storage locations as the tree itself and doubles the storage requirements. Then we move to any node not yet considered and repeat the procedure, but this time starting the second pass after reaching the origin node or a node with an already-computed path length. Because each node is reached strictly twice, this results in only $O(N_o \cdot N)$ operations in every iteration, which is much faster than the shortest-path determination in larger networks.

Second Derivative Length Calculations

Second derivative length calculations require the addition of second derivative lengths of links not common between each path and the corresponding shortest path. If this is done path by path, adding the second derivatives on each link with the shortest path, this also results in $O(N_o \cdot N_o \cdot N^{1/2} \cdot N_p)$ computations. We have so far not been able to find an $O(N_o \cdot N)$ technique for this without changing GP itself to some extent. The difficulty arises because the path under consideration is on another tree that is different from the tree of which the current shortest path is a part. It is possible that we can improve the situation only by changing the algorithm substantially. We suggest the use of a line search rather than the second derivatives to find the step size in the negative gradient direction. An auxiliary path-flow solution can be easily found in the negative gradient direction, and then an unconstrained line search can be used to determine the step size to reach the minimum in this direction. This line search can be performed fast in the link-flow domain (using the link flows at the current and auxiliary path-flow solutions), and on the basis of the optimal step size a path-flow update is performed. The flow update would be based on path flows. Our early experience with this technique has been encouraging.

ASSIGNMENT RESULTS

The assignment studies compare the performance of GP with that of F-W. To make conclusions on the comparative performances of the algorithms it is necessary that they be tested under sufficiently diverse networks. We studied the algorithms on grid networks of different sizes generated by using a random network generator program that we developed as well as on the network of major arterials and freeways in Anaheim, California.

The test networks are grids only in terms of the connectivities of the links, with the link lengths being determined randomly. There are two links each way between the nodes. The link lengths are randomly picked from a uniform distribution of between 500 and 5,000 ft. The free-flow speeds on the links are randomly picked from a uniform distribution of between 22 and 40 mph. The capacities of the links are based on the number of lanes (one, two, or three), with each lane having a capacity of 1,800 vehicles per hr. Certain nodes from the network are randomly picked as O-D centroids. This is done on the basis of a set of rules that attempts to create a network representative of real-world traffic networks. First approximately 12.5 percent (one-eighth) of the total nodes in the network are picked to be centroid nodes, which is about the fraction of zone-centroid nodes in typical assignment applications. There are at least three links between any two centroid nodes to ensure that they are not too close to each other. Once the centroids are set up the O-D flow matrix is generated. Each centroid generates demand at a prespecified rate (9,600 vehicles per hr was used in our studies), and the generated traffic is distributed to other nodes on the basis of the inverse squared distances to develop the O-D matrix.

The Anaheim network has 416 nodes (of which 38 are O-D centroids), 914 arcs, and 1,406 O-D pairs. A static O-D demand matrix was estimated by using the COMEST program on the basis of some link counts in the network. The demand data refer to the evening peak period in the network, which has moderately high levels of congestion. No microcoding of the intersections was attempted for this network.

The assignments were carried out by using a Bureau of Public Roads link-cost function, $t = t_0[1 + 0.15\ (x/c)^4]$, where t is the link travel time cost, t_0 is the free-flow cost, x is the flow, and c is the link capacity. Both GP and F-W included identical shortest-path routines, which is based on a binary heap data structure. The line search routine for F-W uses an efficient Bolzano search [see Sheffi (*1*)]. The programs were implemented in FORTRAN-77 on a Sun SPARC-II workstation with 64 megabytes of storage. The flows and costs were floating point variables.

Table 1 shows the results from assignments on networks of various sizes. For all of the network sizes, Table 1 shows the number of iterations required by F-W to find the objective function value that GP finds in 2, 4, 6, 8, and 10 iterations as well as the corresponding computation times. F-W requires 30 to 160 iterations to reach the solutions found by GP in just 6 iterations. For all of the networks we found that GP converges between 10 and 15 iterations to solutions that F-W takes between 300 and 2,000 iterations to reach. There is at least an order of magnitude improvement in terms of the number of iterations.

The computation times are also improved similar to the reduction in iterations for smaller networks. This is expected because the main computational step is the shortest-path determination for both GP and F-W in small networks. However the computation times with GP are about 40 percent of those with F-W for 10 iterations of GP in a 1,000-node network. This shows that procedures other than the shortest-path determination use up significant time in GP for larger networks, as explained in the previous section. It should be stressed that these assignments were carried out with an implementation of GP that does not yet include most of the procedures that we explained before. Thus even though a 60 percent improvement is significant, the computation times for GP can be reduced even further than those shown in Table 1, especially for larger networks.

Figures 2 to 5 show the results of the assignments on the network of Anaheim for various demand levels. The demands gen-

TABLE 1 Comparative Performances of GP and F-W

Square Grid Networks	Gradient Projection			Frank-Wolfe		
	Iterations	Obj. function	Time (sec)	Iterations	Obj. function	Time (sec)
36 nodes	2	5385.2	0.08	4	5376.7	0.08
120 arcs	4	4890.2	0.14	27	4889.2	0.51
12 OD pairs	6	4794.5	0.17	54	4793.9	1.02
	8	4747.9	0.22	123	4747.8	2.32
	10	4728.8	0.28	390	4728.8	7.35
100 nodes	2	13076.9	0.38	5	13035.8	0.44
360 arcs	4	12562.6	0.76	30	12561.9	2.64
132 OD pairs	6	12526.0	1.14	168	12526.0	14.81
	8	12522.1	1.52	618	12522.1	68.58
	10	12521.4	1.90	1282	12521.4	198.32
225 nodes	2	33872.1	1.87	7	33757.4	2.73
840 arcs	4	32979.2	4.13	36	32977.7	14.03
756 ODs	6	32912.4	6.26	176	32912.4	68.58
	8	32904.8	8.44	509	32904.8	198.32
	10	32902.3	10.52	1611	32902.3	627.20
400 nodes	2	70849.9	5.78	6	70835.1	6.90
1520 arcs	4	68797.2	13.86	21	68774.8	24.14
2450 ODs	6	68408.2	22.67	65	68407.5	74.71
	8	68343.9	31.33	185	68343.9	212.63
	10	68326.8	38.97	537	68326.8	617.19
625 nodes	2	124874	14.51	7	124225	19.03
2400 arcs	4	120151	38.56	23	120144	62.52
6006 ODs	6	119322	64.04	48	119315	130.49
	8	119100	88.77	88	119100	239.22
	10	119028	112.39	156	119028	424.08
900 nodes	2	206379	30.90	7	206097	39.56
3480 arcs	4	198483	86.53	26	198436	146.93
12432 ODs	6	197974	148.76	31	197929	175.19
	8	196668	207.98	80	196665	452.10
	10	196508	271.74	123	196508	695.10

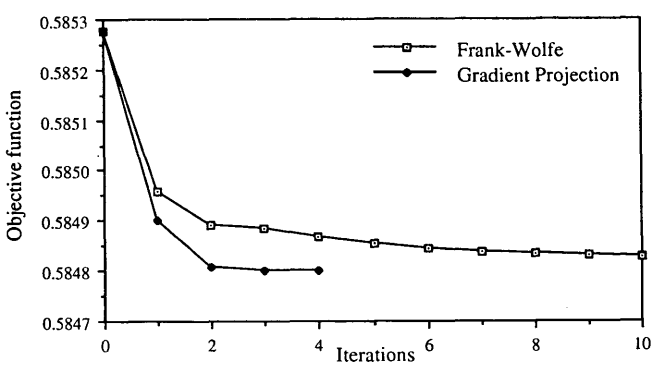

FIGURE 2 Comparison of GP and F-W on Anaheim, California, network (volume level = 0.5).

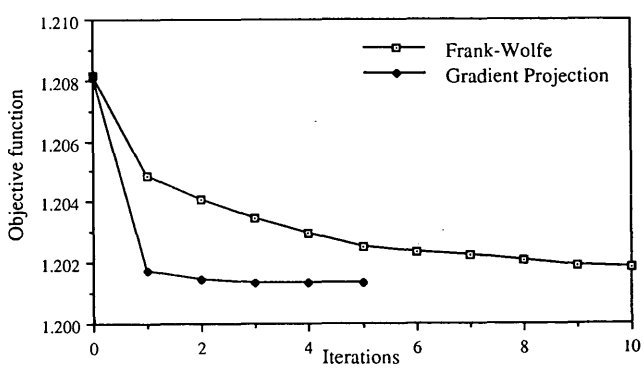

FIGURE 3 Comparison of GP and F-W on Anaheim, California, network (volume level = 1.0).

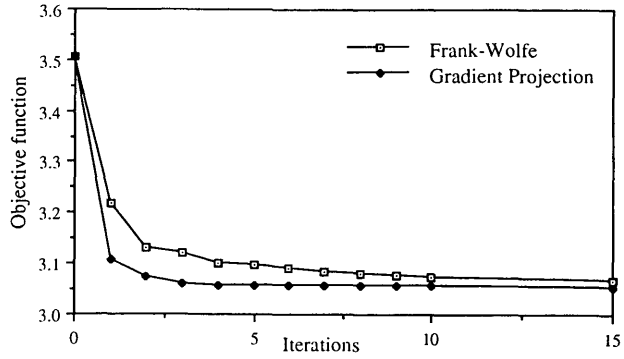

FIGURE 4 Comparison of GP and F-W on Anaheim, California, network (volume level = 1.5).

erated by the O-D matrix estimated from actual link counts are denoted as a demand level of 1.0. For other demand levels the cells in the trip table were all multiplied by appropriate fractions. These assignments were carried out to examine the effect of the demand level on the relative performances of GP and F-W. Again we see that for all cases F-W takes more than 10 to 15 iterations to reach the solutions found by GP in 1 to 3 iterations. There does not appear to be a significant change in the performance of GP in comparison with that of F-W as the demand level increases. Both algorithms require more iterations to converge for higher demand levels, but GP still shows 5 to 10 times faster convergence.

We did not compare the PARTAN version of F-W with GP. However, published results on PARTAN (*10,11*) indicate that this typically is about twice as fast as ordinary F-W, in the number of iterations, in finding solutions. On the basis of the improvements that we found with GP we decided that the comparison of the PARTAN version of F-W with GP was not warranted at this time. Moreover PARTAN is still not commonly used for assignments by practitioners. We do intend, however, to carry out these comparisons in the future.

CONCLUSIONS

In this paper we provided a detailed discussion and supportive results to show that path-enumeration algorithms such as gradient projection deserve a fresh look for applications in traffic assignment. There were two main motivations behind the research: (a) the tremendous improvement in recent years of the availability of computer memory, and (b) the need for fast assignment algorithms for certain possible IVHS strategies for optimal routing and guidance on the basis of dynamic assignment frameworks, real-time trip table estimation, and so on. We show that path-based algorithms can be applied to networks of thousands of nodes. We provide data structures that can be used to handle path-based storage problems, and we suggest techniques for achieving the fast completion of path-based procedures in the algorithm. These techniques are also applicable to other path-based algorithms. Our implementation of GP converges in an order of magnitude fewer iterations than conventional F-W and can be made to show similar computation time speedup if implemented carefully.

There are advantages to the path-based solutions generated by GP. Such solutions can be used directly in path-based routing frameworks. Another advantage is the direct determination of node turning counts without microcoding the intersections and increasing the network size. In addition we can find the link-to-link flow variation on each path. This may provide some opportunities for finding approximate estimates of fuel consumption, environmental impacts, and so on, for selected paths or O-D pairs.

Several aspects of the algorithm require further study. One important aspect is the convergence rates under different link-cost functions. Although we have carried out some research in this area and have found the results to be reasonably robust, our research has by no means been exhaustive. Application to other related problems such as dynamic assignment and variable demand assignment would provide more insights on the algorithm's performance. Research is also under way at the University of California, Irvine, on developing gradient projection with hierarchical decomposition schemes for traffic network assignment.

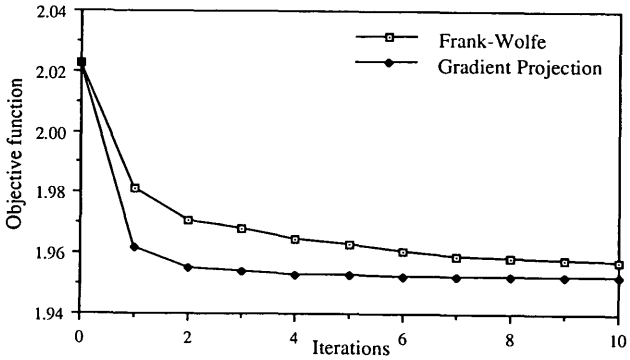

FIGURE 5 Comparison of GP and F-W on Anaheim, California, network (volume level = 2.0).

ACKNOWLEDGMENTS

We wish to thank Wilfred Recker, Institute of Transportation Studies at the University of California, Irvine, for his constant encouragement. The research was supported by the California Department of Transportation under the Anaheim ATMS test bed research program, and we acknowledge the generous support.

REFERENCES

1. Sheffi, Y. *Urban Transportation Networks*. Prentice-Hall, Incorporated, Englewood Cliffs, N.J., 1985.
2. Kaysi, I., and M. Ben-Akiva. An Integrated Approach to Vehicle Routing and Congestion Prediction for Real-Time Driver Guidance. Paper presented at 72nd Annual Meeting of the Transportation Research Board, Washington, D.C., January 1993.
3. Mahmassani, H. S., and S. Peeta. Network Performance under System Optimal and User Equilibrium Dynamic Assignment. Paper presented at 72nd Annual Meeting of the Transportation Research Board, Washington, D.C., January 1993.
4. Bertsekas, D. P. On the Goldstein-Levin-Poljak Gradient Projection Method. *IEEE Transactions on Automatic Control*, Vol. AC-21, 1976, pp. 174–184.
5. Janson, B. Dynamic Traffic Assignment with Schedule Delay. Presented at 71st Annual Meeting of the Transportation Research Board, Washington, D.C., January 1992.
6. Ran, B., D. E. Boyce, and L. J. Leblanc. A New Class of Instantaneous Dynamic Assignment Models. *Operations Research*, Vol. 41, No. 1, 1993.

7. Lupi, M. Convergence of the Frank-Wolfe Algorithm in Transportation Networks. *Civil Engineering Systems*, Vol. 19, 1985, pp. 7–15.
8. Nagurney, A. B. Comparative Tests of Multimodal Traffic Equilibrium Methods. *Transportation Research B*, Vol. 18B, No. 6, 1984, pp. 469–485.
9. Dafermos, S. C., and F. T. Sparrow. The Traffic Assignment Problem for a General Network. *Journal of Research of the National Bureau of Standards-B*. Vol. 73B, No. 2, 1969, pp. 117–119.
10. Arezki, Y., and D. Van Vliet. A Full Analytical Implementation of the PARTAN/Frank-Wolfe Algorithm for Equilibrium Assignment. *Transportation Science*, Vol. 24, 1990, pp. 58–62.
11. LeBlanc, L. J., R. V. Helgason, and D. E. Boyce. Improved Efficiency of the Frank-Wolfe Algorithm for Convex Network Problems. *Transportation Science*, Vol. 19, 1985, pp. 445–462.
12. Florian, M., J. Guélat, and H. Spiess. An Efficient Implementation of the PARTAN Variant of the Linear Approximation Method for the Network Equilibrium Problem. *Networks*, Vol. 17, 1987, pp. 319–339.
13. Weintraub, A., C. Ortiz, and J. Gonzales. Accelerating Convergence of the Frank-Wolfe Algorithm. *Transportation Research B*, Vol. 19B, 1985, pp. 113–122.
14. Larsson, T., and M. Patriksson. Simplicial Decomposition with Disaggregated Representation for the Traffic Assignment Problem. *Transportation Science*, Vol. 26, 1992, pp. 4–17.
15. Florian, M. A Traffic Equilibrium Model of Travel by Car and Public Transit Modes. *Transportation Science*, Vol. 11, 1977, pp. 166–179.
16. Barton, R. B., D. W. Hearn, and S. Lawphongpanich. The Equivalence of Transfer and Generalized Benders Decomposition Methods for Traffic Assignment. *Transportation Research B*, Vol. 23B, 1989, pp. 61–73.
17. Pang, J. S., and D. Chan. Iterative Methods for Variational and Complementarity Problems. *Mathematical Programming*, Vol. 24, 1982, pp. 284–313.
18. Dafermos, S. C. A Variable Inner-Product Projection Algorithm for Variational Inequalities with Application to the Traffic Equilibrium Problem. Working paper. Lefschetz Center for Dynamical Systems, Brown University, Providence, R.I., 1983.
19. Bertsekas, D. P., and R. G. Gallager. *Data Networks*. Prentice-Hall, Incorporated, Englewood Cliffs, N.J., 1987.

Publication of this paper sponsored by Committee on Transportation Supply Analysis.