A Path-Based Gradient Projection Algorithm: Effects of Equilibration with a Restricted Path Set under Two Flow Update Policies

UCI-ITS-TS-WP-98-4

Anthony Chen
R. Jayakrishnan

Department of Civil and Environmental Engineering and
Institute of Transportation Studies
University of California, Irvine

January 1998

Institute of Transportation Studies
University of California, Irvine
Irvine, CA 92697-3600, U.S.A.
http://www.its.uci.edu

A Path-Based Gradient Projection Algorithm: 
Effects of Equilibration with a Restricted Path Set 
Under Two Flow Update Policies

Anthony Chen and R. Jayakrishnan 
Department of Civil & Environmental Engineering and 
Institute of Transportation Studies 
University of California at Irvine 
Irvine, CA. 92697

Abstract

In the network traffic assignment problem, it is generally believed that the minimum number of paths defining an equilibrium solution can be obtained by repeatedly equilibrating the set of paths generated thus far in each iteration before creating new ones in the column generation phase. This scheme aims to minimize the number of shortest path calculations by only bringing in paths with greater potential to cause a reduction in the objective and at the same time retaining the existing paths in the path set. Using the path-based gradient projection (GP) algorithm, we test the restricted path equilibration scheme in two flow update policies: “all-at-once” and “one-at-a-time”. We also discuss the applicability of implementing other known traffic assignment algorithms with these two flow update policies. We find that GP works well with the “one-at-a-time” flow update for both large and small networks without the need to employ the restricted path equilibration scheme.

Keywords: Traffic assignment, user-equilibrium, path-based gradient projection algorithm, simplicial decomposition method, path equilibration

1. Introduction

Recently, there has been a renewed interest in path-based traffic assignment algorithms due to the significant advancement in computing facility. It has been found that path-based algorithms are much more efficient than the link-based algorithms (Larsson and Patriksson, 1992; Jayakrishnan et al., 1994; Sun et al., 1996). In addition, path-based algorithms provide not only the link-flow solution but also the useful path-flow solution that may be required in certain applications. Examples of such applications include
optimal routing in route guidance system, analysis of environmental impact, fuel consumption, etc., based on path profiles of travel speeds, and path-based congestion pricing scheme. Though computer memory has improved many-folds in recent years, it is still a burden to store paths in a large scale network unless the number of iterations or the number of paths to define an equilibrium solution can be kept small. It is generally believed that this can be achieved by fully equilibrating the restricted set of paths in each iteration before generating new ones to add to the path set. Algorithms that used this restricted path equilibration scheme are the simplicial decomposition algorithms by Larsson and Patriksson (1992) and Lee (1994). Basically, the master problem is resolved until the paths generated thus far are equilibrated. This helps to minimize the shortest path calculations in the column generation phase by only finding paths that are better than the existing set of paths, but at the same time increases the computations needed to resolve the master problem many times. Results reported in Lee (1994) show that repeatedly equilibrating the restricted set of paths can reduce the number of iterations to reach convergence considerably compared to solving the master problem just once. Results in these two studies are all based on the “all-at-once” flow update policy. No attempt was made to examine the effects of the equilibration of a restricted set of paths under a different flow update policy.

Another path-based algorithm recently introduced to solve the traffic assignment problem is gradient projection (GP), which has shown excellent computational results (Jayakrishnan et al., 1994). In view of the criticism that this algorithm may be deficient due to it not explicitly using restricted path set equilibrations, this paper examines the effects of fully equilibrating the set of paths generated thus far in each iteration under the “all-at-once” and “one-at-a-time” flow update policies. Our results indicate that updating the flow pattern “one-at-a-time” has similar effect as restricted path equilibrations.

Following the introduction, Section 2 introduces the traffic assignment problem and the gradient projection algorithm. Sections 3 and 4 explain the two flow update policies and the possibility of implementing these two flow update policies in other known traffic assignment algorithms, respectively. Section 5 discusses the restricted path equilibration scheme. Section 6 presents some numerical results and examines the effects of the flow update policies, restricted path equilibration scheme, and column droppings, and Section 7 summarizes the results of the paper.

2. The Traffic Assignment Problem and the Gradient Projection Algorithm

The traffic assignment problem is an essential step in efficient planning and real-time applications in optimal routing, signal control, and traffic prediction in urban traffic networks. It concerns with finding a set of paths to satisfy the travel demands, given by an origin-destination (OD) matrix. The problem can be formulated as a multi-commodity flow problem in either the node-link or path-link representations (Patriksson, 1994). Both representations yield unique link-flow patterns. However, the unique optimal link-flow solution of the node-link representation can be decomposed to more than one
optimal path-flow solution of the path-link representation. Hence, path-flow solution is, in general, not unique.

Though not unique, the constraint set of the path-link representation is highly structured that allows it to be expressed as a Cartesian product as follows:

\[ F = \prod_{i=1}^{W} F_i \]  

(1)

where \( F_i \) is the simplex associated with the path-flow variables of OD pair \( i \) and \( W \) is the set of OD pairs. Thanks to its simple structure, the constraint set can be disaggregated into individual OD pairs in which the path-flow variables for each OD pair are defined by its own non-negativity constraints and one additional constraint to ensure that the sum of the path flows should satisfy the travel demand of that OD pair:

\[ f_{k}^{rs} \geq 0, \quad \forall k \in K_{rs}, \ r \in R, \ s \in S \]  

(2)

\[ \sum_{k \in K_{rs}} f_{k}^{rs} = q_{rs}, \quad \forall r \in R, \ s \in S \]  

(3)

where \( R \) and \( S \) are the sets of origins and destinations, \( f_{k}^{rs} \) is the flow on path \( k \) for OD pair \((r,s)\), \( q_{rs} \) is travel demand between \( r \) and \( s \), and \( K_{rs} \) is the set of paths connecting \( r \) and \( s \).

Let \( x_a \) be the traffic flow on link \( a \in A \), then the definitional equations can be written as:

\[ x_a = \sum_{r \in R} \sum_{s \in S} \sum_{k \in K_{rs}} f_{k}^{rs} \delta_{ka}^{rs}, \quad \forall a \in A \]  

(4)

where \( \delta_{ka}^{rs} = \begin{cases} 1 & \text{if link } a \text{ belongs to path } k \text{ connecting } r \text{ and } s \\ 0 & \text{Otherwise} \end{cases} \)

Using the standard Bureau of Public Road (BRP) link cost function, the average travel time expressed as a function of its link flow is given by:

\[ t_a(x_a) = t_0 (1 + 0.15(x_a/C_a)^4) \]  

(5)

where \( t_0 \) and \( C_a \) are the free-flow travel time and link capacity, respectively. This link cost function is convex and twice continuously differentiable. All numerical results reported in this paper use the BPR link cost function unless stated otherwise.

The assignment can be carried out for finding traffic flow patterns under user equilibrium (when no driver can unilaterally change route to achieve better travel time) or under system optimal (when the total system travel time is minimum, usually under external
control). The objective functions corresponding to these two assignments can be mathematically stated as:

\[
Z_{UE} = \sum_{a \in A} \int_0^{x_a} t_a(w) \, dw \tag{6}
\]

\[
Z_{SO} = \sum_{a \in A} t_a(x_a) \, x_a \tag{7}
\]

where \(Z_{UE}\) and \(Z_{SO}\) are the objective functions of the user equilibrium and system optimal formulations, respectively. The constraint sets for the two assignments are the same.

The path-link formulation of the traffic assignment problem is to minimize either (6) or (7) (depending on the assignment purpose) subject constraints (2) - (4).

For the user equilibrium objective function \(Z_{UE}\), which will be referred as the objective function \(Z\) for the rest of the paper, the optimality conditions are:

\[
f^{rs}_k > 0 \quad \text{if} \quad d^{rs}_k = \bar{d}_{rs} \tag{8a}
\]

\[
f^{rs}_k = 0 \quad \text{if} \quad d^{rs}_k \geq \bar{d}_{rs} \tag{8b}
\]

where \(d^{rs}_k\) denotes the travel time on path \(k\) for OD pair \((r,s)\) and \(\bar{d}_{rs}\) is the minimum travel time between \(r\) and \(s\). The optimality conditions simply state that all used paths of an OD pair will have the minimum travel time and no unused paths can have lower than the minimum travel time.

The most common algorithm used to solve the above traffic assignment problem is the link-based Frank-Wolfe algorithm introduced by LeBlanc et al. (1975). This algorithm is simple and requires very modest storage memory, but has a poor rate of convergence and does not provide path flows automatically. Here, we discuss the Goldstein-Levitin-Poljak gradient projection algorithm, a path-based algorithm formulated by Bertsekas (1976) in the context of communication networks and applied successfully to the transportation networks by Jayakrishnan et al. (1994). In each iteration, the algorithm finds a steepest descent search direction by solving a shortest path problem and updates the solution by scaling it with the second derivative information. The basic update equations are as follows:

\[
f^{rs}_k(n + 1) = [f^{rs}_k(n) - \alpha(n) \, (s^{rs}_k(n))^{-1} \, (d^{rs}_k(n) - d^{rs}_{k_{rs}(n)})]^{+} \tag{9}
\]

\[
f^{rs}_{k_{rs}(n)}(n + 1) = q_{rs} - \sum_{k \in K_{rs}} f^{rs}_k(n + 1) \tag{10}
\]

where \(n\) is the iteration counter, \(\alpha(n)\) is the stepsize, \(s^{rs}_k(n)\) is a diagonal, positive definite scaling matrix, \(d^{rs}_k(n)\) and \(d^{rs}_{k_{rs}(n)}\) are the travel times on path \(k\) and the shortest path \(\bar{k}_{rs}\) between \(r\) and \(s\), and \([ \cdot ]^{+}\) denotes the projection of the argument onto the positive orthant.
of the independent variables (i.e., max \{ 0, f \}). The reason for the path travel time difference to yield the steepest descent direction will be clear from equations (14), (16), and (17) below.

Bertsekas et al. (1984) suggest using \( \alpha(n) = 1 \) as the stepsize for all iteration \( n \) and the second derivative information for an automatic scaling to bypass the need to determine a suitable stepsize. This scheme alleviates the difficulty of finding a proper range of the stepsize and has been found to be a very robust choice in earlier research with different network sizes and topologies (Jayakrishnan et al., 1994; Sun et al., 1996). Also, the demand conservation constraints are embedded into the objective function here. Hence, the projection only needs to ensure that the path flows are non-negative, and this can be easily performed by making the path flow zero if it results in negative flow.

In each iteration, \( f^*_{k} \) is partitioned into the shortest path flow \( f^*_{k_{rs}} \) and the non-shortest path flows \( f^*_{k} \) for each OD pair (r,s) with the path set \( K_{rs} \). The demand conservation constraints can be removed from the constraint set by expressing \( f^*_{k_{rs}} \) in terms of \( f^*_{k} \).

\[
f^*_{k_{rs}} = q_{rs} - \sum_{k \in K_{rs}, k \neq k_{rs}} f^*_{k}, \quad \forall \, k_{rs}, r \in R, s \in S
\]

(11)

where \( k_{rs} \) denotes the shortest path from \( r \) to \( s \). This leaves only the non-negativity constraints on the non-shortest paths. Substituting the shortest path flow \( f^*_{k_{rs}} \) for each OD pair into the objective function, we obtain the new optimization problem of the form

\[
\min \tilde{Z}(\tilde{f})
\]

subject to \( f^*_{k} \geq 0 \), \( \forall \, k \in K_{rs}, k \neq k_{rs}, r \in R, s \in S
\]

(12)

(13)

where \( \tilde{Z} \) is the transformed objective function in terms of the non-shortest path flows, \( \tilde{f} \), for all OD pairs. The first and second derivatives of the transformed objective function can be easily derived:

\[
\frac{\partial \tilde{Z}}{\partial f^*_{k}} = \frac{\partial Z}{\partial f^*_{k}} - \frac{\partial Z}{\partial f^*_{k_{rs}}}, \quad \forall \, k \in K_{rs}, k \neq k_{rs}, r \in R, s \in S
\]

(14)

\[
\frac{\partial^2 \tilde{Z}}{(\partial f^*_{k})^2} = \frac{\partial^2 Z}{(\partial f^*_{k})^2} - 2 \frac{\partial^2 Z}{\partial f^*_{k} \partial f^*_{k_{rs}}} + \frac{\partial^2 Z}{(\partial f^*_{k_{rs}})^2}, \forall \, k \in K_{rs}, k \neq k_{rs}, r \in R, s \in S
\]

(15)

where \( Z \) is the original objective function (equation 6 restated using equation 4) with all paths in the path set, including both the shortest and non-shortest paths. Each component of the gradient becomes the difference between the first derivatives with respect to a non-shortest path and the shortest path. Note that the first derivative of \( Z \) with respect to any path is simply the sum of the link costs on that path calculated at the current flow pattern.
\[
\frac{\partial Z}{\partial f_{ks}^r} = \sum_{a \in A} t_a(x_a) \delta_{ks}^r \\
\frac{\partial Z}{\partial f_{kn}^r} = \sum_{a \in A} t_a(x_a) \delta_{kn}^r
\]  

(16)  

(17)

Equations (16) and (17) explain the \((d_k^r(n) - d_{k,\bar{k}}^r(n))\) term in equation (9). The second derivatives of the transformed objective function given in equation (15) can also be written as

\[
\frac{\partial^2 Z}{(\partial f_{k}^r)^2} = \sum_{a \in A} t'_a(x_a) \left( \delta_{ka}^r - \delta_{k,a}^r \right)^2, \quad \forall \ k \in \mathcal{K}_r, \ k \neq \bar{k}_r, \ r \in \mathcal{R}, \ s \in \mathcal{S}
\]  

(18)

where \(t'_a(x_a)\) is the first derivative of the link cost function (i.e., same as the second derivative of the objective function \(Z\) in equation 15). Observe that a small increase in the flow on a path \(k\) results in an equal amount of reduction of flow on the corresponding shortest path \(\bar{k}_r\), and causes no change in the flow on the common part of the two paths. Thus, the second derivative are calculated using only links not common to \(k\) and \(\bar{k}_r\).

Denoting \(d_k^r\) and \(d_{k,\bar{k}}^r\) as the first derivative costs of path \(k\) and the shortest path \(\bar{k}_r\) given in equations (16) and (17), respectively, and \(x_k^r\) as the second derivative cost given in (18), the basic iterative (flow update) equations given in (9) and (10) are all known quantities, except \(\alpha\) which is a scalar stepsize modifier which may be chosen by different methods. Since the flow update equations employ automatic scaling based on the second derivative information, a constant stepsize of 1 for all iterations (which seems to work well regardless of the travel demand \(q_{rs}\)) can be used (Bertsekas et al., 1984).

With the basic flow update equations stated, we can give the complete algorithmic steps as follows:

**Initialization:** Generate the first path for each OD pair at free-flow travel time and perform an all-or-nothing (AON) assignment.

1. Set \(x_a(0) = 0, \ t_a = t_a[x_a(0)], \ \forall \ a, \ \text{and} \ K_r = \emptyset \).
2. Set iteration counter \(\Rightarrow n = 1\).
3. Solve the shortest path problem \(\Rightarrow \bar{k}_r(n), \ K_r = \bar{k}_r(n) \cup K_r\)
4. Perform AON assignment \(\Rightarrow f_{\bar{k}_r(n)}^r(n) = q_{rs}, \ \forall \ r, s\).
5. Assign path flows to links \(\Rightarrow x_a(n) = \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}} \sum_{k \in K_r} f_k^r(n) \delta_{ka}^r, \ \forall \ a\).

**GP Solver:** Generate paths based on current link travel times, augment the set of generated paths, and solve the basic flow update equations.
6. Increment iteration counter \( n = n + 1 \).
7. Update link travel time \( t_a(n) = t_a(x_{rs}(n - 1)), \forall a \).
8. Solve shortest path problem \( \tilde{k}_{rs}(n), \forall r, s \).
9. Determine whether \( \tilde{k}_{rs}(n) \) exists in the path set \( K_{rs} \) or not \( \Rightarrow \):
   - If \( \tilde{k}_{rs}(n) \notin K_{rs} \), then \( K_{rs} = \tilde{k}_{rs}(n) \cup K_{rs} \).
   - Otherwise, tag the shortest path among the paths in \( K_{rs} \) as \( \tilde{k}_{rs}(n) \).
10. Compute first and second derivative costs \( \Rightarrow d_k^{rs}(n), d_k^{rs}(n), \text{ and } s_k^{rs}(n) \).
11. Update non-shortest path flows \( f_k^{rs}(n + 1) = \max \{ ([f_k^{rs}(n) - \alpha(n) (s_k^{rs}(n))]^{-1} (d_k^{rs}(n) - d_k^{rs}(n)), 0) \}, \forall k \neq \tilde{k}_{rs}(n), r, s \).
12. If \( f_k^{rs}(n + 1) = 0 \), then drop path \( k \Rightarrow K_{rs} = K_{rs} \setminus k \).
13. Update shortest path flow \( f_{\tilde{k}_{rs}(n)}^{rs}(n + 1) = q_{rs} - \sum_{k \in K_{rs}} f_k^{rs}(n + 1), \forall r, s \).
14. Update link flows \( x_a(n + 1) = \sum_{r \in R} \sum_{s \in S} \sum_{k \in K_{rs}} f_k^{rs}(n + 1) \delta_{ka}^{rs}, \forall a \).
15. Determine maximum path length (travel time) deviation of all OD pairs \( \Rightarrow \):
   \[ E = \max_{rs} \sum_{k \in K_{rs}} \left( \frac{d_k^{rs}(n) - d_k^{rs}(n)}{q_{rs}} \right) \left( f_k^{rs}(n) \left( d_k^{rs}(n) - d_k^{rs}(n) \right) \right) \]
16. Check convergence \( \Rightarrow \) If \( E \leq \varepsilon \), then stop. Otherwise, go to 6.

In Step 11, a unity stepsize is chosen for \( \alpha(n) \) for all iteration \( n \) since \( [s_k^{rs}(n)]^{-1} \) is used for scaling. It is well known in nonlinear programming analysis that methods which employ diagonal scaling based on second derivative information work well with a unity stepsize. It can be shown that given any starting set of path flows there exists an \( \bar{\alpha} > 0 \) such that if \( \alpha \in (0, \bar{\alpha}) \) the sequence generated by this algorithm converges to the optimal (Bertsekas and Gallager, 1987).

3. Flow Update Policies

For the multi-commodity flow problem (i.e., more than one OD pair), it is possible to implement the path-based gradient projection algorithm in at least two different ways. One policy is the “all-at-once” flow update. In each iteration, the path flows for all the OD pairs are updated simultaneously. This policy is equivalent to the Jacobi algorithm used in the classical iterative technique for solving systems of linear equations (Bertsekas and Tsitsiklis, 1989). In this method, the total link-flow pattern (hence, the link-cost pattern) is adjusted only after all the path flows from all OD pairs have been assigned to the network. The alternate policy is the “one-at-a-time” flow update. In each
iteration, the assignment is carried out sequentially one OD pair (or one origin) at a time in a cyclic manner with re-adjustment of link-flow and link-cost patterns after each OD pair (or origin). This policy can be considered as a variation of the cyclic coordinate method in nonlinear programming (Bazarra and Shetty, 1979), or as a special case of the Gauss-Seidel algorithm in the general iterative technique (Bertsekas and Tsitsiklis, 1989). It converges faster than the “all-at-once” flow-update policy, because it incorporates the latest available information resulting from the previous OD pair (or origin) to update the current OD pair (or origin). Reasons for this speedy convergence are that the “one-at-a-time” flow update accounts for the interaction between the flows of different OD pairs, approximates the Taylor series expansion better because of less flow variations (i.e., the basic iterative equations are obtained from a second-order Taylor approximation of the original problem), and increases the likelihood of using a unity stepsize without inducing divergence.

To implement the “one-at-a-time” flow update version of the path-based gradient projection algorithm, it is relatively easy to modify the algorithm given in the previous section. The required changes are:

(1) **Initialization:** Before building the shortest path tree for the next origin, update the link travel times after the AON assignment of the OD pairs of the current origin.

(2) **GP Solver:** Same as the modifications in the Initialization. That is, find the shortest paths as a tree for a given origin as the root node, with the shortest paths to multiple destinations found simultaneously. The OD pairs are then picked in order, one at a time. The flows on all paths of an OD pair are updated. When the next OD pair is picked for flow update, the same path trees are used, but with updated link flows and costs. Thus, the algorithm finds a shortest path tree for one origin at a time, followed by flow update on all paths belonging to each individual OD pair for all the destinations of this origin. This is followed by finding the shortest path tree for the next origin until all origins are completed.

4. **Applicability of Implementing the Two Flow Update Policies in Other Traffic Assignment Algorithms**

All the conventional link-flow based traffic assignment algorithms operate with the “all-at-once” flow-update policy. These include the Frank-Wolfe algorithm introduced by LeBlanc et al. (1975), the PARTAN algorithm presented by LeBlanc et al. (1985) and subsequently improved by Florian et al. (1987) and Arezki and Van Vliet (1990), the modified Frank-Wolfe algorithm by Fukushima (1984), and the accelerated Frank-Wolfe algorithm by Weintraub et al. (1985). These algorithms store only the link-flow solutions, and consequently cannot be implemented in the “one-at-a-time” flow update mode. The reason for this is that the line search step requires feasible auxiliary link flows (i.e., total link flows from all OD pairs rather than a single OD pair). Without explicitly storing path-flow solution, it is not possible to remove the background traffic from the
current link-flow solution and add them to the one OD auxiliary link-flow solution to form a feasible auxiliary link-flow solution. However, if the paths are retained in each iteration, it opens up the possibility of "one-at-a-time" flow update implementation. Recently, Chen et al. (1996) develop a path-based version of the Frank-Wolfe algorithm with "one-at-a-time" flow update policy and show that faster convergence can be achieved compared to the conventional "all-at-once" flow update implementation of the FW-type algorithms.

Another group of algorithms which have been found to be more efficient than the above FW-type algorithms is the simplicial decomposition method. Algorithms that belong to this general class include the restricted simplicial decomposition algorithm by Hearn et al. (1987), Schittenhelm’s algorithm (1990), improved Schittenhelm’s algorithms by Lee (1994), and the disaggregate simplicial decomposition algorithm by Larsson and Patriksson (1992). All these algorithms have been studied only under the "all-at-once" flow update policy even though paths are retained at each iteration for the latter three algorithms. No previous attempt was made to examine the effects of "one-at-a-time" flow update policy. Since paths are stored in the algorithms developed by Schittenhelm, Lee, and Larsson and Patriksson, it is simple to implement the "one-at-a-time" flow update in these simplicial decomposition algorithms. In the restricted simplicial decomposition algorithm, it may be difficult to update flows "one-at-a-time" since only the auxiliary link-flow vectors are kept in each iteration.

5. Restricted Path Equilibration Scheme

This section discusses the rationale behind the restricted path equilibration scheme and proposes a way to implement this balancing procedure within the path-based gradient projection algorithm.

The amount of flow to divert from the non-shortest paths to the shortest path, given in equation (9), is determined by \( \alpha(n), s_k^n(n), (d_k^n(n) - d_k^{rs}(n)) \). If the product of these factors is larger than its existing path flow \( f_k^n(n) \), then all flow on the non-shortest path will be shifted to the shortest path. This activates the projection operator which effectively drops the non-shortest path from the path set since it no longer carries positive flow. However, the same non-shortest path may join the path set at a later iteration if it becomes more attractive. It is possible that the same path exits and enters the path set several times during the iterations. To minimize this cycling effect, one can use a restricted path equilibration scheme suggested by Larsson and Patriksson (1992) to fully equilibrate the set of paths generated so far prior to returning to the column generation phase to create new paths. This has the effect of, at each iteration, bringing in only profitable paths that are more probable to cause a reduction in the objective value, and at the same time, retaining the existing paths in the path set.

This restricted path equilibration scheme can be easily implemented in the gradient projection algorithm by adding a procedure to equilibrate the path flows until the path
costs are balanced. At Step 15 of *GP Solver*, if convergence is met, then stop.
Otherwise, apply the restricted path equilibration procedure until the path costs are
reasonably balanced, increment iteration counter \( n = n + 1 \) and go to Step 6.

**Restricted Path Equilibration (RPE) Procedure:** balance the path costs by adjusting the
flows on a restricted set of paths generated thus far.

1. Set equilibration counter \( m = 1 \),
\[
\bar{x}_a(m) = x_a(n + 1), \quad \forall \ a,
\]
and
\[
\bar{f}_k^r(m) = f_k^r(n + 1), \quad \forall \ k, r, s.
\]

2. Update link travel time \( => t_a(m) = t_a(x_a(m)), \quad \forall \ a. \)

3. Compute the first derivative costs \( => \)
\[
d_k^r(m), \quad \forall \ k, r, s.
\]

4. Determine the minimum first derivative cost \( => \)
\[
d_{k, r}(m) = \arg \min_{k \in K_a} d_k^r(m), \quad \forall \ r, s.
\]

5. Compute the second derivative costs \( => \)
\[
s_k^r(m), \quad \forall \ k \neq k_a(m), r, s.
\]

6. Update non-shortest path flows \( => \)
\[
\bar{f}_k^r(m + 1) = \max \left\{ \left[ \frac{\bar{f}_k^r(m) - \alpha(m) \left( s_k^r(m) \right)^{-1} (d_k^r(m) - d_{k_a(m)}^r) \right]}{0} \right\},
\quad \forall \ k \neq k_a(m), r, s.
\]

7. Update shortest path flow \( => \)
\[
\bar{f}_{k_a(m)}^r(m + 1) = q_{rs} - \sum_{k \in K_a} \bar{f}_k^r(m + 1), \quad \forall \ r, s.
\]

8. Update link flows \( => \)
\[
\bar{x}_a(m + 1) = \sum_{r \in R} \sum_{s \in S} \sum_{k \in K_a} \bar{f}_k^r(m + 1) \delta_{ba}, \quad \forall \ a.
\]

9. Check whether the paths in \( K_a \) have equilibrated or not.
If yes, go to Step 6 of *GP Solver*; else increment equilibration counter \( m = m + 1 \) and go to Step 2 of the RPE procedure.

Again, \( \alpha(m) \) in Step 7 is chosen in similar fashion as the *GP Solver*. Also, column
dropping is not used in the RPE procedure since the goal is to retain the paths within the
path sets. It is also possible to remove the column dropping (Step 11 of the *GP Solver*) if
the RPE procedure is used in the gradient projection algorithm.

6. Numerical Results

In this section, we report the results of the path-based gradient projection (GP) algorithm
with and without the restricted path equilibration (RPE) scheme under two flow update
policies. This results in four different implementations:
1. AAO-GP (all-at-once GP without RPE),
2. AAO-GP-RPE (all-at-once GP with RPE),
3. OAAT-GP (one-at-a-time GP without RPE), and
4. OAAT-GP-RPE (one-at-a-time GP with RPE).

We test these four implementations on several networks that have been used in the transportation literature. These networks include the 9-node network from Hearn and Kim (1995), the 13-node network from Nguyen and Dupius\(^1\) (1985), the well-known Sioux Falls network (LeBlanc et al., 1975), and the Anaheim network (Jayakrishnan et al., 1994). Tables 1 and 2 summarize the computational results of the above four networks at levels of accuracy of 1.0 \(\times 10^{-3}\) and 1.0 \(\times 10^{-4}\), respectively. The size of each network is given as a triplet: number of nodes/number of arcs/number of origin-destination (OD) pairs with positive travel demand. Columns three to six report the computational performance (i.e., final objective value, number of iterations, number of equilibrations\(^2\), and computational times) of the four algorithms. Columns seven and eight provide two path statistics (generated paths and active paths\(^3\)) to examine the question regarding the minimum number of paths to define an (approximate) equilibrium solution. The last column of Table 1 provides the reason for termination of the numerical run. It is based on one of the following three criteria: (1) maximum number of iterations allowed (MaxIter), (2) maximum path length (cost) deviation from the shortest path is within a prespecified tolerance (PLD)\(^4\), and (3) five consecutive iterations could not decrease the objective value (ObjVal).

### 6.1 Effects of the “One-At-A-Time” Flow Update Policy

In all the test networks, the “one-at-a-time” flow update policy, with and without the restricted path equilibration scheme, converges faster in terms of both the number of iterations and computational times than the “all-at-once” flow update policy. For the two smaller networks, the two flow update policies have comparable computational times, but the “one-at-a-time” flow update policy requires fewer iterations to satisfy the path length deviation stopping criterion. This criterion measures the degree of violation of the Wardrop’s equal travel time principle. Though the “all-at-once” algorithms were able to get to the same objective value, they cannot pass the path length deviation stopping

---

\(^1\) This network uses linear link cost functions, as opposed to the BPR functions.

\(^2\) For the “all-at-once” flow update policy, equilibration is applied to all OD pairs at once, whereas for the “one-at-a-time” flow update policy, equilibration is performed one OD pair at a time. Therefore, the number of equilibrations for the “all-at-once” case should be multiplied by the number of OD pairs in order to have a fair comparison with the “one-at-a-time” case.

\(^3\) An active path is a path with positive flow.

\(^4\) This stopping criterion can be expressed as:

\[
E = \max_{r,s} e^{rs} = \sum_{k \in K_r, k \neq \hat{k}_r} \frac{f_k^{rs}}{q_k^{rs}} \left( \frac{d_k^{rs} - d_{\hat{k}_r}^{rs}}{d_k^{rs}} \right) \leq \varepsilon.
\]
criterion. For the two larger networks, the “one-at-a-time” flow update policy shows not only substantial computational advantage, but also significantly fewer iterations compared to the “all-at-once” flow update policy. This is particularly useful, because the path storage, which is the main memory requirement in a path-based traffic assignment algorithm, depends on the number of iterations to reach convergence. Using a predecessor arc list to store the shortest path trees, the required memory amounts to \( N_i \times N \), where \( N_i \) is the number of iterations it takes to satisfy the stopping criterion, \( N_0 \) is the number of origins, and \( N \) is the number of nodes in the network. Both \( N \) and \( N_0 \) are readily fixed by the network topology, but \( N_i \) depends on the performance of the algorithm. Hence, it is crucial that the path-based gradient projection algorithm can achieve fast convergence for it to be of practical use. It is also worthy to mention that the unity stepsize works well with the “one-at-a-time” flow update policy for all four networks, but diverges under the “all-at-once” flow update policy for the two larger networks. The results shown in Tables 1 and 2 use a much smaller stepsize (obtained experimentally) to ensure convergence.

6.2 Effects of the Restricted Path Equilibration Scheme

The restricted path equilibration scheme, in each iteration, tries to equalize the travel time of each path within the path set generated thus far by redistributing the flow among them. In general, the repetition of this equilibration scheme can help to reduce the number of iterations to reach convergence and also the number of shortest path calculations. This reduction is achieved by adding extra computational effort to repeatedly solve the equilibration procedure. From Tables 1 and 2, it seems that the “all-at-once” flow update policy can gain some benefits by re-equilibrating the path costs among the restricted set of paths generated in each iteration. The number of iterations till termination is reduced quite a bit. This also cuts down the computational time spent on the shortest path calculations, but this is compensated by adding extra computational time to balance the path costs. As for the “one-at-a-time” flow update policy, the restricted path equilibration scheme has no effects on decreasing either the computational time or the number of iterations. In fact, no more than one equilibration is needed to satisfy the termination criterion in the restricted path equilibration procedure if the flow update is adjusted one OD at a time. The reason is that the flow adjustment has already corrected some of the unbalanced path costs.

6.3 Effects of Column Dropping in the Restricted Path Equilibration Scheme

As mentioned before in Section 5, column dropping may be removed if the gradient projection algorithm is implemented with the restricted path equilibration procedure. In this section, we provide two sets of results (\( 1.0 \times 10^{-3} \) and \( 1.0 \times 10^{-4} \) accuracy levels) in Tables 3 and 4 on the effects of dropping the inactive paths before applying the restricted path equilibration procedure and compare with the option of keeping all the generated paths for equilibration. Dropping the unused paths prior to equilibration, especially under
the “all-at-once” flow update policy, increases the number of iterations and the number of equilibrations (except the Anaheim network which has not fully converged), but reduces the computational times if the number of active paths is considerably less than the number of generated paths. This helps to keep the number of flow carrying paths small and also reduces the amount of calculation and bookkeeping needed in each iteration.

An interesting observation between the two flow update policies is that both the numbers of generated and active paths in the “one-at-a-time” policy are less than the “all-at-once” policy. This suggests that by adjusting the link flows one OD at a time has the effect of keeping the path set small. The results of the four networks consistently show that the “one-at-a-time” flow update algorithms generate and retain fewer paths compared to the “all-at-once” flow update algorithms. The difference is more apparent for the larger size networks. This result is significant, because in the past it is generally believed that the minimum number of paths defining an equilibrium solution can only be obtained through the method of repeating the equilibration procedure to the restricted set of paths generated in each iteration. This result shows that the “one-at-a-time” flow update policy can achieve similar effect with even less paths.

7. Conclusions

This paper has examined the effects of equilibration with a restricted set of paths generated in each iteration in two flow update policies. The “all-at-once” policy simultaneously updates the path flows for all OD pairs in each iteration. The “one-at-a-time” policy updates the path flows of one OD pair at a time in a cyclic manner with intermediate link flow adjustments to reflect the changes in the path flows of each OD pair. This gives rise to four different implementations of the path-based gradient projection algorithm. The results show that repeatedly solving the restricted path equilibration procedure can help in the “all-at-once” flow update policy, but unnecessary under “one-at-a-time” flow update policy. The results also reveal that the “one-at-a-time” flow update policy generates and retains less number of paths compared to the “all-at-once” flow update policy with or without the restricted path equilibration scheme. Finally, the “one-at-a-time” policy increases the robustness of using a unity stepsize in the path-based gradient projection algorithm without inducing divergence.
References


<table>
<thead>
<tr>
<th>Network Descriptions</th>
<th>Algorithm</th>
<th>Final Obj.</th>
<th># of Iter.</th>
<th># of Equil.</th>
<th>CPU (sec.)</th>
<th># of Paths Generated</th>
<th># of Paths Active</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nine-Node (9/18/4) Hearn and Kim (1995)</td>
<td>AAO-GP</td>
<td>1453.1533</td>
<td>15</td>
<td>0</td>
<td>0.0532</td>
<td>26</td>
<td>18</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE</td>
<td>1453.1525</td>
<td>11</td>
<td>43</td>
<td>0.0678</td>
<td>27</td>
<td>21</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>1453.1692</td>
<td>10</td>
<td>0</td>
<td>0.0483</td>
<td>20</td>
<td>18</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>1453.1542</td>
<td>10</td>
<td>1</td>
<td>0.0478</td>
<td>26</td>
<td>21</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td>Thirteen-Node (13/19/4) Nguyen and Dupius (1984)</td>
<td>AAO-GP</td>
<td>85028.08</td>
<td>15</td>
<td>0</td>
<td>0.0506</td>
<td>12</td>
<td>8</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE</td>
<td>85028.88</td>
<td>9</td>
<td>11</td>
<td>0.0460</td>
<td>10</td>
<td>8</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>85028.09</td>
<td>11</td>
<td>0</td>
<td>0.0459</td>
<td>10</td>
<td>8</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>85028.08</td>
<td>11</td>
<td>1</td>
<td>0.0469</td>
<td>10</td>
<td>9</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td>Sioux Falls (24/76/528) LeBlanc et al. (1975)</td>
<td>AAO-GP</td>
<td>42.4500</td>
<td>50</td>
<td>0</td>
<td>1.9633</td>
<td>1500</td>
<td>700</td>
<td>MaxIter</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE</td>
<td>42.3140</td>
<td>39</td>
<td>61</td>
<td>2.0512</td>
<td>1225</td>
<td>704</td>
<td>ObjVal</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>42.3134</td>
<td>16</td>
<td>0</td>
<td>0.5998</td>
<td>992</td>
<td>647</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>42.3134</td>
<td>25</td>
<td>1</td>
<td>1.0340</td>
<td>1020</td>
<td>666</td>
<td>ObjVal</td>
</tr>
<tr>
<td>Anaheim (416/914/1406) Jayakrishnan et al (1994)</td>
<td>AAO-GP</td>
<td>1205757.50</td>
<td>50</td>
<td>0</td>
<td>13.5164</td>
<td>2756</td>
<td>1809</td>
<td>MaxIter</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE</td>
<td>1205643.00</td>
<td>27</td>
<td>53</td>
<td>13.1491</td>
<td>3059</td>
<td>1905</td>
<td>ObjVal</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>1205625.50</td>
<td>8</td>
<td>0</td>
<td>2.3539</td>
<td>2198</td>
<td>1572</td>
<td>PLD&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>1205625.50</td>
<td>8</td>
<td>0</td>
<td>2.4819</td>
<td>2198</td>
<td>1572</td>
<td>PLD&lt;0.001</td>
</tr>
</tbody>
</table>

Table 1. Gradient Projection computational results for four test networks (PLD = 0.001)
<table>
<thead>
<tr>
<th>Network Descriptions</th>
<th>Algorithm</th>
<th>Final Obj.</th>
<th># of Iter.</th>
<th># of Equil.</th>
<th>CPU (sec.)</th>
<th># of Paths Generated</th>
<th># of Paths Active</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nine-Node (9/18/4)</td>
<td>AAO-GP</td>
<td>1453.1521</td>
<td>25</td>
<td>0</td>
<td>0.0690</td>
<td>28</td>
<td>21</td>
<td>ObjVal</td>
</tr>
<tr>
<td>Hearn and Kim (1995)</td>
<td>AAO-GP-RPE</td>
<td>1453.1519</td>
<td>19</td>
<td>43</td>
<td>0.0835</td>
<td>29</td>
<td>23</td>
<td>ObjVal</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>1453.1525</td>
<td>14</td>
<td>0</td>
<td>0.0575</td>
<td>22</td>
<td>20</td>
<td>PLD&lt;1E-4</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>1453.1519</td>
<td>15</td>
<td>1</td>
<td>0.0567</td>
<td>27</td>
<td>21</td>
<td>PLD&lt;1E-4</td>
</tr>
<tr>
<td>Thirteen-Node (13/19/4)</td>
<td>AAO-GP</td>
<td>85028.06</td>
<td>26</td>
<td>0</td>
<td>0.0638</td>
<td>12</td>
<td>8</td>
<td>ObjVal</td>
</tr>
<tr>
<td>Nguyen and Dupius (1984)</td>
<td>AAO-GP-RPE</td>
<td>85028.82</td>
<td>21</td>
<td>11</td>
<td>0.0645</td>
<td>10</td>
<td>8</td>
<td>ObjVal</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>85028.07</td>
<td>14</td>
<td>0</td>
<td>0.0506</td>
<td>10</td>
<td>8</td>
<td>PLD&lt;1E-4</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>85028.09</td>
<td>13</td>
<td>1</td>
<td>0.0509</td>
<td>10</td>
<td>9</td>
<td>PLD&lt;1E-4</td>
</tr>
<tr>
<td>Sioux Falls (24/76/528)</td>
<td>AAO-GP</td>
<td>42.4500</td>
<td>50</td>
<td>0</td>
<td>1.9633</td>
<td>1500</td>
<td>700</td>
<td>MaxIter</td>
</tr>
<tr>
<td>LeBlanc et al. (1975)</td>
<td>AAO-GP-RPE</td>
<td>42.3140</td>
<td>39</td>
<td>61</td>
<td>2.0512</td>
<td>1225</td>
<td>704</td>
<td>ObjVal</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>42.3134</td>
<td>25</td>
<td>0</td>
<td>0.9609</td>
<td>1012</td>
<td>664</td>
<td>ObjVal</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>42.3134</td>
<td>25</td>
<td>1</td>
<td>1.0340</td>
<td>1020</td>
<td>666</td>
<td>ObjVal</td>
</tr>
<tr>
<td>Anaheim (416/914/1406)</td>
<td>AAO-GP</td>
<td>1205757.50</td>
<td>50</td>
<td>0</td>
<td>13.5164</td>
<td>2756</td>
<td>1809</td>
<td>MaxIter</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP</td>
<td>1205625.50</td>
<td>9</td>
<td>0</td>
<td>3.0033</td>
<td>2216</td>
<td>1594</td>
<td>PLD&lt;1E-4</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE</td>
<td>1205625.50</td>
<td>9</td>
<td>0</td>
<td>3.1723</td>
<td>2216</td>
<td>1594</td>
<td>PLD&lt;1E-4</td>
</tr>
</tbody>
</table>

Table 2. Gradient Projection computational results for four test networks (PLD = 0.0001)
<table>
<thead>
<tr>
<th>Network Descriptions</th>
<th>Algorithm</th>
<th>Final Obj.</th>
<th># of Iter.</th>
<th># of Equil.</th>
<th>CPU (sec.)</th>
<th># of Proj.</th>
<th># of Paths Generated</th>
<th># of Paths Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nine-Node (9/18/4)</td>
<td>AAO-GP-RPE1</td>
<td>1453.1522</td>
<td>11</td>
<td>38</td>
<td>0.0653</td>
<td>9</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>1453.1525</td>
<td>11</td>
<td>43</td>
<td>0.0678</td>
<td>7</td>
<td>27</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>1453.1542</td>
<td>10</td>
<td>1</td>
<td>0.0475</td>
<td>6</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>1453.1542</td>
<td>10</td>
<td>1</td>
<td>0.0478</td>
<td>5</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>Thirteen-Node (13/19/4)</td>
<td>AAO-GP-RPE1</td>
<td>85028.10</td>
<td>9</td>
<td>10</td>
<td>0.0467</td>
<td>3</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>85028.88</td>
<td>9</td>
<td>11</td>
<td>0.0460</td>
<td>3</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>85028.08</td>
<td>11</td>
<td>1</td>
<td>0.0472</td>
<td>1</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>85028.08</td>
<td>11</td>
<td>1</td>
<td>0.0469</td>
<td>1</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Sioux Falls (24/76/528)</td>
<td>AAO-GP-RPE1</td>
<td>42.3141</td>
<td>33</td>
<td>61</td>
<td>2.8977</td>
<td>313</td>
<td>1164</td>
<td>714</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>42.3140</td>
<td>39</td>
<td>61</td>
<td>2.0512</td>
<td>301</td>
<td>1255</td>
<td>704</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>42.3134</td>
<td>25</td>
<td>1</td>
<td>1.2107</td>
<td>362</td>
<td>960</td>
<td>665</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>42.3134</td>
<td>25</td>
<td>1</td>
<td>1.0340</td>
<td>321</td>
<td>1020</td>
<td>666</td>
</tr>
<tr>
<td>Anaheim (416/914/1406)</td>
<td>AAO-GP-RPE1</td>
<td>1205637.00</td>
<td>32</td>
<td>57</td>
<td>19.6842</td>
<td>2744</td>
<td>3699</td>
<td>1923</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>1205643.00</td>
<td>27</td>
<td>53</td>
<td>13.1491</td>
<td>2799</td>
<td>3059</td>
<td>1905</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>1205625.50</td>
<td>8</td>
<td>0</td>
<td>2.5265</td>
<td>582</td>
<td>2153</td>
<td>1572</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>1205625.50</td>
<td>8</td>
<td>0</td>
<td>2.4819</td>
<td>515</td>
<td>2198</td>
<td>1572</td>
</tr>
</tbody>
</table>

Table 3. Computational performance of the restricted path equilibration scheme with and without column dropping (PLD = 0.001). RPE1 retains all paths generated to perform the restricted path equilibration scheme, while RPE2 (same as the one shown in Table 1) drops the inactive paths prior to equilibration.
<table>
<thead>
<tr>
<th>Network Descriptions</th>
<th>Algorithm</th>
<th>Final Obj.</th>
<th># of Iter.</th>
<th># of Equil.</th>
<th>CPU (sec.)</th>
<th># of Proj.</th>
<th># of Paths Generated</th>
<th># of Paths Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nine-Node (9/18/4)</td>
<td>AAO-GP-RPE1</td>
<td>1453.1521</td>
<td>14</td>
<td>38</td>
<td>0.0729</td>
<td>9</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>1453.1519</td>
<td>19</td>
<td>43</td>
<td>0.0835</td>
<td>7</td>
<td>29</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>1453.1519</td>
<td>15</td>
<td>1</td>
<td>0.0566</td>
<td>8</td>
<td>25</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>1453.1519</td>
<td>15</td>
<td>1</td>
<td>0.0567</td>
<td>6</td>
<td>27</td>
<td>21</td>
</tr>
<tr>
<td>Thirteen-Node (13/19/4)</td>
<td>AAO-GP-RPE1</td>
<td>85028.09</td>
<td>16</td>
<td>10</td>
<td>0.0577</td>
<td>3</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>85028.82</td>
<td>21</td>
<td>11</td>
<td>0.0645</td>
<td>3</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>85028.09</td>
<td>13</td>
<td>1</td>
<td>0.0505</td>
<td>1</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>85028.09</td>
<td>13</td>
<td>1</td>
<td>0.0509</td>
<td>1</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Sioux Falls (24/76/528)</td>
<td>AAO-GP-RPE1</td>
<td>42.3141</td>
<td>33</td>
<td>61</td>
<td>2.8977</td>
<td>313</td>
<td>1164</td>
<td>714</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>42.3140</td>
<td>39</td>
<td>61</td>
<td>2.0512</td>
<td>301</td>
<td>1255</td>
<td>704</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>42.3134</td>
<td>25</td>
<td>1</td>
<td>1.2107</td>
<td>362</td>
<td>960</td>
<td>655</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>42.3134</td>
<td>25</td>
<td>1</td>
<td>1.0340</td>
<td>321</td>
<td>1020</td>
<td>666</td>
</tr>
<tr>
<td>Anaheim (416/914/1406)</td>
<td>AAO-GP-RPE1</td>
<td>1205637.00</td>
<td>32</td>
<td>57</td>
<td>19.6842</td>
<td>2744</td>
<td>3699</td>
<td>1923</td>
</tr>
<tr>
<td></td>
<td>AAO-GP-RPE2</td>
<td>1205643.00</td>
<td>27</td>
<td>53</td>
<td>13.1491</td>
<td>2799</td>
<td>3059</td>
<td>1905</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE1</td>
<td>1205625.50</td>
<td>9</td>
<td>0</td>
<td>3.2540</td>
<td>586</td>
<td>2166</td>
<td>1594</td>
</tr>
<tr>
<td></td>
<td>OAAT-GP-RPE2</td>
<td>1205625.50</td>
<td>9</td>
<td>0</td>
<td>3.1723</td>
<td>519</td>
<td>2216</td>
<td>1594</td>
</tr>
</tbody>
</table>

Table 4. Computational performance of the restricted path equilibration scheme with and without column dropping (PLD = 0.0001). RPE1 retains all paths generated to perform the restricted path equilibration scheme, while RPE2 (same as the one shown in Table 2) drops the inactive paths prior to equilibration.