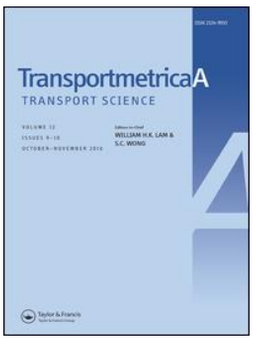

Taylor & Francis
Taylor & Francis Group

Transportmetrica A: Transport Science

ISSN: 2324-9935 (Print) 2324-9943 (Online) Journal homepage: https://www.tandfonline.com/loi/ttra21

Reduced gradient algorithm for user equilibrium traffic assignment problem

Abbas Babazadeh, Babak Javani, Guido Gentile & Michael Florian

To cite this article: Abbas Babazadeh, Babak Javani, Guido Gentile & Michael Florian (2020) Reduced gradient algorithm for user equilibrium traffic assignment problem, Transportmetrica A: Transport Science, 16:3, 1111-1135, DOI: 10.1080/23249935.2020.1722279

To link to this article: https://doi.org/10.1080/23249935.2020.1722279

Accepted author version posted online: 27 Jan 2020.
Published online: 19 Feb 2020.

TRANSPORTMETRICA A: TRANSPORT SCIENCE
2020, VOL. 16, NO. 3, 1111–1135
https://doi.org/10.1080/23249935.2020.1722279

Reduced gradient algorithm for user equilibrium traffic assignment problem

Abbas Babazadeh [a], Babak Javani[a], Guido Gentile [b] and Michael Florian[c]

[a]School of Civil Engineering, College of Engineering, University of Tehran, Tehran, Iran; [b]Dipartimento di Ingegneria Civile, Edile e Ambientale, Sapienza Universita di Roma, Roma, Italy; [c]CIRRELT, Université de Montréal, Montréal, Canada

ABSTRACT

A path-based algorithm is developed for the static traffic assignment problem (TAP). In each iteration, it decomposes the problem into origin-destination (OD) pairs and solves each subproblem separately using the Wolfe reduced gradient (RG) method. This method reduces the dimensions of each single-OD subproblem by selecting a basic path between the OD pair and reformulating the subproblem in terms of the nonbasic paths. A column generation technique is included to avoid path enumeration in large scale networks. Also, some speed-up techniques are designed to improve the computational efficiency. The algorithm shifts flows from costlier paths to cheaper paths; however, the amount of flow shifted from a costlier path is proportional to not only the travel time but also the flow on the path. It is applied to the Philadelphia and Chicago test problems, while different strategies for choosing the basic paths are examined. The RG algorithm shows an excellent convergence to relative gaps of the order of 1.0E-14 when compared against several reference TAP algorithms.

ARTICLE HISTORY
Received 2 August 2018
Accepted 2 December 2019

KEYWORDS
Traffic assignment problem; reduced gradient method; path-based algorithm; large scale network

Introduction

Assigning an origin-destination (OD) demand matrix to an urban network for determining link flows and travel times is known as the traffic assignment problem (TAP). The solution of TAP needs assumption(s) about how the users distribute among traveling paths/routes between the OD pairs. Wardrop's first principle (Wardrop 1952), called the user equilibrium (UE) condition, states: 'The journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route'. The UE condition is achieved when no traveler can improve his travel time by unilaterally changing routes. Based on the UE condition, Beckmann, McGuire, and Winsten (1956) transformed the TAP into an optimization problem that works in both forms of link-path and link-node representations of transportation networks. Since then, extensive research have been carried out on providing solution algorithms for Beckmann's optimization model. The same

CONTACT Abbas Babazadeh ababazadeh@ut.ac.ir School of Civil Engineering, College of Engineering, University of Tehran, P.O. Box 11155–4563, Tehran, Iran

problem was also equivalently formulated as a non-linear complementarity problem (NCP) and several algorithms were provided to solve it. In the following, 'UE-based TAP' is simply referred as 'TAP'.

The TAP is the heart of the transportation planning procedures like network design problem and congestion pricing, because it is required to be solved so many times in order to predict the flow pattern for combinations of alternatives. In practice, reducing the runtimes of the TAP algorithms, even a few seconds, may decrease the planning time for a few weeks or months. Despite the fact that the first algorithm of the TAP was path-based (Dafermos and Sparrow 1969), the link-based algorithms were in the center of attention before the 1990s because of their low memory requirements. LeBlanc, Morlok, and Pierskalla (1975) applied the method of convex combination of Frank and Wolfe (1956), which led to the most famous algorithm of the TAP. The Frank-Wolfe (FW) algorithm has been used in most of the software packages for traffic assignment because of its simple implementation. This is while it has a slow convergence rate and cannot gain precise solutions even after days. Its poor performance is attributed to the severe zigzag movement that occurs close to the optimal solution. Many researchers have attempted to enhance its performance by modifying its search direction (see Patriksson 1994, 102). Most recently, Mitradjieva and Lindberg (2013) improved the FW directions by introducing the bi-conjugate FW (BFW) method and Holmgren and Lindberg (2014) enhanced the FW performance by studying subproblem updating in the FW method.

Although the link-based algorithms are mostly easy to implement and have modest memory usage, they are not usually efficient and rapid enough. In addition, they do not store the information of the active paths (i.e. the paths used by the travelers), which are of considerable interest or importance to the transportation planners. On the other hand, recent advances in computer science and advent of random access memories (RAM) with larger storage capacities have opened the way for implementing path-based algorithms on real-life networks. The main problem for carrying out the path-based algorithms on large or even medium scale networks is that enumerating and storing the details of all the existing paths are almost impossible. The column generation technique is usually applied to overcome this problem by letting the algorithms work with the set of active paths rather than all paths existing in the network. Kumar, Peeta, and Nie (2012) investigated three active path set update strategies: origin-destination (OD) based, origin based and simultaneous strategy. They showed that the simultaneous strategy, which updates the active paths simultaneously for all OD pairs, is the most efficient strategy among them. In the path-based algorithms, the main problem is usually decomposed into subproblems in terms of OD pairs. Accordingly, there are three possible strategies for updating link flows: All-At-Once updates the link flows simultaneously after solving the subproblems of all OD pairs, One-Origin-At-A-Time updates link flows once the subproblems having the same origin are solved and One-OD-At-A-Time updates link flows once the subproblem of each OD pair is solved. Chen (2001) and Chen and Lee (1999) acclaimed that the last strategy is very effective and can extremely accelerate the convergence speed of the path-based algorithms.

Larsson and Patriksson (1992) presented the disaggregate simplicial decomposition (DSD) algorithm, where in each iteration the TAP is replaced by a restricted master problem (RMP) defined in terms of the current shortest paths and those generated in the last iterations. Alternately, the RMP is disaggregated by OD pairs, and the subproblem of each OD

pair is solved using a combination of the reduced gradient (RG) method and an approximate Newton method. Jayakrishnan et al. (1994) applied the gradient projection (GP) method of Bertsekas (1976) to devise another a path-based algorithm, which moves flows from the non-shortest paths to the shortest paths of the same OD pairs. They eliminated the demand constraints and reformulated the objective function based on the non-shortest paths. Once the non-shortest-path flows of an OD pair are found, the shortest-path flow of that OD pair can be computed by subtracting its demand from the sum of the non-shortest-path flows. Chen, Lee, and Jayakrishnan (2002) compared the performance of DSD and GP algorithms on several test networks, and claimed that the GP algorithm was superior. Lee, Nie, and Chen (2003) proposed a conjugate GP algorithm by calculating the Hessian matrix more accurately. The path-based version of FW algorithm, called OD- based FW (ODBFW), was developed by Chen, Jayakrishnan, and Tsai (2002). Utilizing the One-OD-At-A-Time flow update strategy, it decomposes the TAP in terms of OD pairs and solves each subproblem by the FW algorithm. Analogous to the algorithm proposed by Jayakrishnan et al. (1994), the ODBFW moves flows to the shortest paths from others. Florian, Constantin, and Florian (2009) developed the projected gradient (PG) algorithm utilizing Rosen's GP method (Rosen 1960) to shift flows of every OD pair from the paths with higher costs (i.e. higher travel times) than the average to those with lower costs. The components of the search direction are set equal to the difference between the path costs and the average cost of the same OD pair, while the step size is found by doing a usual line search. Kumar and Peeta (2010) developed the slope based multi-path algorithm, which is inspired by the works of Jayakrishnan et al. (1994) and Florian, Constantin, and Florian (2009); it shifts flows from the higher-cost to the lower-cost paths like the latter, and uses a constant step length like the former. Kumar and Peeta (2014a) introduced a slope-based algorithm, which shifts flows based on the sensitivity of the path travel times with respect to path flows. Di Lorenzo, Galligari, and Sciandrone (2015) proposed the inexact sequential minimal optimization algorithm by adopting a decomposition-based approach and a column generation strategy, which shifts the flow of the maximum cost path of each OD pair to one of their lower cost paths. Javani and Babazadeh (2017) developed an algorithm which finds descent directions by partially solving a sequence of quadratic programing (QP) subproblems in a truncated QP (TQP) framework. The algorithm, called OD-based FW TQP (ODFWTQP), decomposes each QP subproblem in OD pairs, and solves each subproblem by the FW method considering only the active paths. Most recently, a path equilibration algorithm was designed by Galligari and Sciandrone (2018), where in each iteration the flows of only two paths belonging to each OD pair are adjusted using an inexact line search and an adaptive column generation technique. Bedsides, Xie, Nie, and Liu (2018) presented a greedy (GREEDY) path-based algorithm in which a heuristic approach is utilized to solve the subproblem corresponding to each OD pair. The line search is one of the most essential factors in the overall efficiency of the TAP algorithms. Chen et al. (2013) invented a self-adaptive Armijo strategy to find acceptable step sizes in TAP algorithms and analyzed the convergence rate of FW, DSD and GP algorithms with this line search. Kumar and Peeta (2014b) proposed strategies to improve the performance of path-based algorithms by enhancing the computational efficiency of their shortest paths finding, path flows updating and link flows and travel times updating stages. The convergence behavior and the numerical stability of the path-based algorithms are investigated by Perederieieva et al. (2016) when precise solutions are required. Additionally, Babazadeh (2005) and Babazadeh and Aashtiani

(2005) developed a path-based algorithm to solve the more challenging equilibrium transit assignment problem for real-life networks.

The origin-based algorithms are another class of the TAP algorithms, first proposed by Bar-Gera (2002). Their memory usage is lower than the path-based algorithms and can have remarkable performances as shown so far. They are based on the fact that every user travels on a bush rooted at his origin. A bush rooted at a node is an acyclic subnetwork (of the network) that includes at least one path from the node to every other nodes. These algorithms iterate two main steps: bush reconstruction and equilibration (Nie 2010). Equilibrating a bush (i.e. finding UE flow on the bush) is performed by shifting flows between the links, as does in the origin-based (OBA) algorithm of Bar-Gera (2002), or the paths on the bush, as does in the B algorithm of Dial (2006). Nie (2010) performed comparisons among four variants of Newton-type origin-based algorithms. Bar-Gera (2010) developed an origin-based algorithm by shifting flows between paired alternative segments, which was named the traffic assignment by paired alternative segments (TAPAS) algorithm. Gentile (2014) proposed another origin-based algorithm, called the local user equilibrium (LUCE) algorithm, which partitions the problem in terms of destinations and finds an approximate solution for each destination by the linearization of the travel time functions. Zheng (2015) introduced another new algorithm to the family of origin-based algorithms. Also, Xie and Xie (2015) conducted an extensive numerical and analytical investigation on the origin-based algorithms. Inoue and Maruyama (2012) and Perederieieva et al. (2015) reviewed some of the TAP algorithms and compared their rate of convergence in several test networks. Finally, Gentile (2016) compared different gradient projection algorithms applying them also to the case of dynamic traffic assignment.

This paper is aimed to show the application of the RG method of Wolfe (1967) to the solution of the TAP. There are also other contributions in this direction. Nguyen (1974) devised an algorithm based on the link-node formulation of Beckmann's model using the convex simplex method of Zangwill (1969) that is a modification of the RG method. Florian and Nguyen (1974) used the RG method for finding the solution of the TAP with elastic demands. Larsson and Patriksson (1992) combined the RG method with an approximate second order method to design a solution approach for the link-path formulation of the TAP. Unfortunately, despite the excellent efficiency of the RG, its performance has never been investigated on large scale networks. In this connection, a path-based RG algorithm is developed by applying a column generation technique along with the One-OD-At-A-Time flow update strategy and the simultaneous active path set update strategy. The algorithm decomposes the problem in terms of OD pairs and solves each subproblem by the RG method.

The numerical convergence of the RG algorithm is directly compared to the conventional FW and the Bar-Gera's OBA algorithms; as well as the newer ODFWTQP (Javani and Babazadeh 2017) and GREEDY (Xie, Nie, and Liu 2018) algorithms. It is also compared with the BFW (Mitradjieva and Lindberg 2013) and PG (Florian, Constantin, and Florian 2009) algorithms, using the software package Emme 4 (INRO 2017); and with the LUCE algorithm (Gentile 2014), using the software package Visum 16 (PTV 2016). In addition, indirect comparison with algorithm B is performed based on the results given in Dial (2006). The networks of Philadelphia and Chicago are selected for the purposes of the research, and a sensitivity analysis is conducted for assessing how different choices of basic variables affect

the convergence rate of the algorithm. The numerical results demonstrate performance gain of the proposed algorithm.

The rest of this paper is organized as follows. In the next section, the TAP is formally stated and its main properties are noted. The RG method is outlined and adapted to the TAP in the subsequent section. Another section is devoted to present the numerical results, and the summary and conclusions are provided in the final section.

TAP model

Let graph $G = (N, A)$ represent the transportation network with N as the set of nodes and A as the set of links. The Beckmann's transformation of the UE condition (Beckmann, McGuire, and Winsten 1956) is a nonlinear optimization problem with linear constraints, as follows:

$$\begin{aligned} \underset{h}{Min} \quad & Z(x(h)) = \sum_{a \in A} \int_0^{x_a} t_a(v)dv & \\ \text{s.t.} \quad & \sum_{p \in P_i} h_p = D_i & \forall i \in I \\ & h_p \geq 0 & \forall p \in P_i, i \in I \\ & x_a = \sum_{i \in I} \sum_{p \in P_i} \delta_{ap} h_p & \forall a \in A \end{aligned} \tag{1}$$

where

$I =$ set of OD pairs with positive demands
$D_i =$ demand flow rate from the origin to the destination of OD pair i
$P_i =$ set of paths from the origin node to the destination node of OD pair i
$\delta_{ap} = 1$ if link a lies on path p, and 0 otherwise
$x_a =$ flow on link a
$h_p =$ flow on path p

and $t_a(x_a) : [0, \infty) \rightarrow [0, \infty)$ is the travel time function of link $a \in A$. Considering x and h to be the vectors of link and path flows, respectively, the last constraint of the problem can be regarded as the vector-valued function $x = x(h)$ with the components $x_a = x_a(h)$ for $a \in A$. The travel time on each path p is the sum of the travel times of the links lying on the path, which is written as the function $T_p(h) = \sum_{a \in A} t_a(x_a(h))\delta_{ap}$. The optimization model (1) assumes that the travel time of each link is a function of just the flow on that link, resulting that the Karush-Kuhn-Tucker (KKT) conditions of (1) are equivalent to the Wardrop's first principle, and so every KKT point is a UE solution. Moreover, assuming that the travel time functions $t_a(x_a)$ are positive, non-decreasing and continuously differentiable (the travel time function of a typical link on a real-life network has these features), the objective function of (1) is twice continuously differentiable and convex (Sheffi 1985). Interestingly, the TAP comprises a separable and convex objective function with linear constraints, which form a convex set of feasible solutions. Hence, convex optimization algorithms, like RG, are well applicable to the problem.

RG method

The method of RG was originally proposed by Wolfe (1967) to solve nonlinear programing problems with linear constraints. This method reduces the dimension of the problem by dividing the variable set into basic and nonbasic subsets, and reformulating the problem

based on the nonbasic variables. In this paper, we deal with problem (1), which is a minimization problem with both linear equality and nonnegativity constraints. In what follows, we shortly describe a variation of the RG method for this special problem. Throughout the paper, all vectors are column vectors, except that the gradient is considered to be a row vector.

Consider the following minimization problem:

$$\begin{aligned} \min_{y} \quad & f(y) \\ \text{s.t.} \quad & Ay = b \\ & y \geq 0 \end{aligned} \tag{2}$$

where f is a continuously differentiable function on R^n, A is an $m \times n$ matrix, and b is a vector of dimension m. Wolfe (1967) made the nondegeneracy assumption that matrix A has rank m; and every feasible solution to the problem has at least m strictly positive components and at most $n - m$ zero components. At any iteration of the method, A is partitioned into $[B, N]$ and y into (y_B, y_N), where B is an $m \times m$ invertible matrix, and y_B and y_N are the vector of basic and nonbasic variables with the dimensions of m and $n - m$, respectively. Accordingly, the original problem (2) can be expressed as

$$\begin{aligned} \min_{y} \quad & f(y_B, y_N) \\ \text{s.t.} \quad & By_B + Ny_N = b \\ & y_B \geq 0, \quad y_N \geq 0. \end{aligned} \tag{3}$$

The basic idea of the RG method is that for any specified y_N the equality constraint of problem (3) can be uniquely solved for y_B as $y_B = B^{-1}b - B^{-1}N y_N$, and then the objective function of the problem will be reduced to a function of y_N only, whose gradient at given point $\bar{y}$ equals

$$r^t = -\nabla_B f(\bar{y}) B^{-1} N + \nabla_N f(\bar{y}) \tag{4}$$

where ∇_B and ∇_N denote the gradients with respect to y_B and y_N, respectively, and r^t is the transpose of the $(n - m) \times 1$ vector $r = (r_j)$, which is called the reduced gradient at $\bar{y}$.

Let $\bar{y}$ be a feasible point of problem (3), and $d = (d_B, d_N)$ be the RG direction partitioned into vectors d_B and d_N corresponding to the basic and nonbasic variables, respectively. For vector d to be a feasible descent direction it is required that (i) $Ad = Bd_B + Nd_N = 0$, (ii) $d_j \geq 0$ for $\bar{y}_j = 0$, and (iii) $\nabla f(\bar{y})\, d < 0$. Condition (i) is equivalent to say that d should lie in the null space of A, and obviously, condition (i) remains true if for any d_N we let

$$d_B = -B^{-1} N d_N. \tag{5}$$

To satisfy conditions (ii) and (iii), $d_N = (d_{Nj})$ is set equal to the negative of r, except that the components d_{Nj} with $r_j > 0$ and $\bar{y}_j = 0$ are held at zero. Wolfe (1972) showed by an example that this method does not necessarily converge to a KKT point, but McCormick (1970) presented a modified version as

$$d_{Nj} = \begin{cases} -r_j & \text{if } r_j \leq 0 \\ -\bar{y}_j r_j & \text{if } r_j > 0 \end{cases} \tag{6}$$

which enables convergence (Bazaraa, Sherali, and Shetty 2006, 651). Notice that the direction d defined by (5) and (6) satisfies conditions (i) and (ii), and hence is a feasible direction.

Furthermore, to verify condition (iii), we can use (4) to write

$$\nabla f(\bar{y})d = \nabla_B f(\bar{y})d_B + \nabla_N f(\bar{y})d_N = (-\nabla_B f(\bar{y})B^{-1}N + \nabla_N f(\bar{y}))d_N = r^t d_N \tag{7}$$

which using (6) yields $\nabla f(\bar{y})\, d < 0$ if $d_N \neq 0$. This shows the RG direction d is a descent direction if it has at least a nonzero component. In addition, Bazaraa, Sherali, and Shetty (2006, 604) proved that $d = 0$ (or equivalently, $d_N = 0$) iff $\bar{y}$ is a KKT point of the problem (2).

After the direction d is found, an improving solution is achieved by moving from $\bar{y}$ according to $y = \bar{y} + \alpha\, d$, where α is the optimal step size along the direction. The step size is determined by solving a one-dimensional line search problem of the form

$$\begin{aligned} \min_{\alpha} \quad & f(\bar{y} + \alpha d) \\ \text{s.t.} \quad & \bar{y} + \alpha d \geq 0\,. \end{aligned} \tag{8}$$

Finally, a solution is obtained that minimize f with respect to the current set of basic variables. Subsequently, the procedure is restarted with the choice of a new set of m basic variables (requiring to be strictly positive), and iterates the same steps of direction finding and line search until convergence.

Bazaraa, Sherali, and Shetty (2006, 610) proved that the above variant of the Wolfe's RG method converges to a KKT point whenever the m largest variables are chosen to be the basic variables (ties are broken arbitrarily). A convergence rate analysis was also established by Luenberger and Ye (2008, 387). They proved that the method converges linearly and the rate of convergence is strongly related to the choice of the basic variables.

Adaptation of RG method to TAP

The proposed algorithm follows a Gauss-Seidel decomposition scheme, in which the original problem (1) is decomposed in terms of OD pairs, and the decomposed subproblems are separately solved by the above-described RG method in a consecutive manner. To avoid path enumeration in large scale networks, the subproblem of each OD pair is formulated by considering a restricted set of active paths (i.e. the paths having the potential to be used), instead of requiring all paths joining the origin to the destination. This restricts each single-OD assignment to the links lying on its own active paths. The algorithm initializes the active path sets of the OD pairs, each comprising a single path, and updates them simultaneously per each major iteration. A column generation technique is applied to generate active paths, and the simultaneous strategy is applied to update the active path sets. Moreover, the algorithm utilizes the One-OD-At-A-Time strategy for updating link flows and travel times once a subproblem is solved.

Consider an OD pair i with active path set $P_i^+ \subset P_i$ comprising $n = |P_i^+|$ paths connecting the origin to the destination. Let $\bar{h}$ be a feasible solution to problem (1) where $\bar{h}_p = 0$ for $p \notin \cup_{i\in I} P_i^+$, and $x_a(\bar{h}) = \sum_{i\in I}\sum_{p\in P_i}\delta_{ap}\bar{h}_p$ be the current flow on link $a \in A$. The restricted

subproblem corresponding to active path set P_i^+ is rewritten as

$$\begin{aligned} \min_{(h_p:p\in P_i^+)} \quad & Z_i(h_p : p \in P_i^+) = \sum_{a\in A} \int_0^{x_a(\bar{h})+\sum_{p\in P_i^+}\delta_{ap}(h_p-\bar{h}_p)} t_a(v)dv \\ \text{s.t.} \quad & \sum_{p\in P_i^+} h_p = D_i \\ & h_p \geq 0 \quad \forall p \in P_i^+ \end{aligned} \tag{9}$$

which is much smaller in size than if it was written for all the paths in P_i. This subproblem can be stated in the general form of problem (2), considering that $f = Z_i, y = (h_p : p \in P_i^+)$, $A = [1, \ldots, 1]_{1\times n}$ and b is a positive scalar equal to D_i. The algorithm partitions y into a basic vector $y_B = (h_{p'})_{1\times 1}$ and the nonbasic vector $y_N = (h_p : p \neq p')_{(n-1)\times 1}$. In other words, an arbitrary path $p' \in P_i^+$ is made basic and the other paths in $P_i^+ - \{p'\}$ are declared nonbasic. Moreover, by dividing A into $B = [1]_{1\times 1}$ and $N = [1, \ldots, 1]_{1\times(n-1)}$, the subproblem can be expressed in the form of problem (3). Notice that A is a matrix of rank 1, hence invertible, and y has at least one component that is strictly positive. Consequently, Wolfe's assumptions hold true, and then the method of RG can be applied to obtain a feasible descent direction $d = (d_p : p \in P_i^+)$ after the reduced gradient $r = (r_p : p \in P_i^+ - \{p'\})$ is computed. It is to note that computing the reduced gradient is not costly in our applications because our results (see section 'Numerical experiments') show that the average number of active paths in real case problems is of the order of 1.3 to 1.4 paths per OD pair.

To compute the reduced gradient vector, we first note that the gradient of the objective function Z_i at point $\bar{h}$ equals the vector of the current travel times on the active paths, having the components

$$\frac{\partial Z_i(\bar{h})}{\partial h_p} = \sum_{a\in A} \frac{\partial Z_i(\bar{h})}{\partial x_a} \frac{\partial x_a(\bar{h})}{\partial h_p} = \sum_{a\in A} t_a(x_a(\bar{h}))\delta_{ap} = T_p(\bar{h}) = \bar{T}_p \quad \forall p \in P_i^+. \tag{10}$$

It follows immediately that the gradients with respect to the basic and nonbasic variables are

$$\nabla_B Z_i(\bar{h}) = (\bar{T}_{p'})_{1\times 1} \tag{11}$$

$$\nabla_N Z_i(\bar{h}) = (\bar{T}_p : p \neq p')_{1\times(n-1)} \tag{12}$$

and substituting these into (4) we obtain

$$r^t = -\nabla_B Z_i(\bar{h}) B^{-1} N + \nabla_N Z_i(\bar{h}) = (\bar{T}_p - \bar{T}_{p'} : p \neq p')_{1\times(n-1)}. \tag{13}$$

This shows the reduced gradient r is the vector of the travel times of the nonbasic paths minus the travel time of the basic path, having the components

$$r_p = \bar{T}_p - \bar{T}_{p'} \quad \forall p \in P_i^+ - \{p'\}. \tag{14}$$

Using this in (6), the components of the RG direction d corresponding to the nonbasic paths are determined by

$$d_p = \begin{cases} -(\bar{T}_p - \bar{T}_{p'}) & \text{if } (\bar{T}_p - \bar{T}_{p'}) \leq 0 \\ -\bar{h}_p(\bar{T}_p - \bar{T}_{p'}) & \text{if } (\bar{T}_p - \bar{T}_{p'}) > 0 \end{cases} \quad \forall p \in P_i^+ - \{p'\} \tag{15}$$

and then, by using (5), the component corresponding to the basic path is set equal to

$$d_{p'} = -\sum_{p \in P_i^+ - \{p'\}} d_p. \tag{16}$$

Once the RG direction d is found, the optimal step length $\alpha \geq 0$ is found by minimizing $Z_i(\bar{h}_p + \alpha d_p : p \in P_i^+)$ with respect to α provided that $h_p = \bar{h}_p + \alpha d_p$ remains nonnegative for all $p \in P_i^+$. Note that how to choose the basic path is realized in the next section.

The proposed algorithm makes use of two iteration counts: outer (or major) and the inner (or minor) iterations. At the start of each outer iteration, a shortest path based column generation method is used to update the active path set P_i^+ for each OD pair i. It is well-known that solving the shortest path problems are computationally burdensome and consumes most of the running times of the TAP algorithms. It therefore seems reasonable to bring all the restricted subproblems (9) into equilibrium (with keeping the current active path sets unchanged) before going to the next outer iteration. To do this, the algorithm performs a number of inner iterations within each outer iteration to get a suitable level of precision considering all OD pairs simultaneously. Each inner iteration involves scanning the OD list while each single-OD subproblem is solved by RG iterations to a desired level of precision (unless it is currently achieved). At the same time, the algorithm skips the OD pairs with only one path in their active path sets.

For each OD pair i, the precision of the corresponding subproblem at point $\bar{h}$ is measured using the *restricted gap* defined by

$$gap_i = \sum_{p \in P_i^+} \bar{h}_p \bar{T}_p - D_i \min_{p \in P_i^+} (\bar{T}_p) \tag{17}$$

which is the gap from the total travel time of OD pair i at the current solution to its desired value occurring as if each user experienced the current shortest path among the paths in the restricted set P_i^+. The proposed stopping criterion for each subproblem is that gap_i becomes smaller than a fraction γ (say 0.1) of the *average gap* achieved at the last outer iteration, which is defined as

$$AGap = \frac{\sum_{i \in I} Gap_i}{|I|} \tag{18}$$

where Gap_i equals the distance between the total travel time of the travelers of OD pair i and the total travel time of them as if they experienced the shortest path travel time on the whole network. This is stated as

$$Gap_i = \sum_{p \in P_i^+} h_p T_p - D_i u_i \quad \forall i \in I \tag{19}$$

where h is the solution from the last outer iteration, T_p is the travel time on path p at solution h, and u_i is the shortest path travel time of OD pair i as calculated at the beginning of the current outer iteration. Based on our experiments, the performance of the algorithm is markedly improved if the OD pairs with restricted gaps of less than, say, 10% of the average gap are skipped. After visiting all OD pairs, the overall convergence of the inner iterations

is checked using the relative average of the latest values of gap_i for all $i \in I$, yielding the *restricted relative gap*

$$rgap = \frac{\sum_{i\in I} gap_i}{\sum_{i\in I}\sum_{p\in P_i^+} \bar{h}_p \bar{T}_p}. \tag{20}$$

This is compared against the *relative gap*

$$RGap = \frac{\sum_{i\in I} Gap_i}{\sum_{i\in I}\sum_{p\in P_i^+} h_p T_p} \tag{21}$$

which is the relative average of the values of Gap_i, $i \in I$, as calculated above. Note that *RGap* is the same classical relative gap often used to measure the convergence rates of the TAP algorithms, which is always a non-negative value approaching zero as the path flows getting closer to the UE solution. Also, it is based on the shortest paths on the whole network and so differs from *rgap* that is restricted to the active path set. Our experiments confirm that the stopping criterion for inner iterations should become stricter as the algorithm proceeds. Therefore, the inner iteration loop terminates and a new outer iteration starts whenever *rgap* falls below *RGap* multiplied by a given small ϵ, say 0.1, and divided by the major iteration counter.

The steps of the proposed RG algorithm are as follows:

Step 0 (Initialization) Find an initial solution h by performing an all-or-nothing (AON) assignment. For each OD pair i, form the active path set P_i^+ containing the shortest path found for OD pair i through the AON assignment. Compute $x_a = \sum_{i\in I}\sum_{p\in P_i} + \delta_{ap} h_p$ for all $a \in A$. Set outer iteration counter $k = 0$.

Step 1 (Column generation) Update $t_a = t_a(x_a)$ for all $a \in A$, and calculate the shortest path trees rooted at origins. For each OD pair i: calculate $T_p = \sum_{a\in A} t_a(x_a)\,\delta_{ap}, \forall p \in P_i^+$; store the shortest path travel time in u_i; add the corresponding shortest path to P_i^+ (if not included).

Step 2 (Outer iteration termination). Calculate Gap_i using (19), $\forall i \in I$, and then *RGap* using (21). If the target relative gap is achieved, stop. Otherwise, set $k = k + 1$, and calculate *AGap* using (18) before going to Step 3.

Step 3 (Decomposition and subproblem solution) For each OD pair i: calculate $\bar{T}_p = \sum_{a\in A} t_a(x_a)\,\delta_{ap}$ and store h_p in $\bar{h}_p$ for all $p \in P_i^+$; calculate gap_i using (17); if $|P_i^+| > 1$, repeat the following steps (at least once) until $gap_i < \gamma \cdot AGap$:

Step 3.1 (Direction finding) Choose a path $p' \in P_i^+$ as the basic path. Construct direction vector $d = (d_p : p \in P_i^+)$ according to (15) and (16).

Step 3.2 (Step size finding) Find α that solves

$$\begin{cases} \min\limits_{\alpha} & Z_i(\bar{h}_p + \alpha d : p \in P_i^+) \\ \text{s.t.} & 0 \le \alpha \le \min\{-\bar{h}_p/d_p : d_p < 0, p \in P_i^+\} \end{cases}$$

Step 3.3 (Movement and updating) For each path $p \in P_i^+$, update $h_p = \bar{h}_p + \alpha d$, and $x_a = x_a + (h_p - \bar{h}_p)\,\delta_{ap}$ for all $a \in A$ lying on path p. Eliminate zero-flow paths from P_i^+. Calculate $\bar{T}_p = \sum_{a\in A} t_a(x_a)\,\delta_{ap}$ and store h_p in $\bar{h}_p$ for all $p \in P_i^+$. Calculate gap_i using (17).

Step 4 (Inner iteration termination) Calculate *rgap* using (20). If $rgap < \varepsilon \cdot RGap/k$ go to Step 1; otherwise, repeat Step 3. ■

Compared to the other state-of-the-art alternatives, the algorithm above offers computational advantages for the following reasons.

First, at each inner iteration (Step 3), the RG method is applied to solve a sequence of OD-based subproblems in the form of (9). This is a nonlinear problem with $n = |P_i^+|$ variables and $n + 1$ constraints (i.e. one linear equality constraint and n nonnegativity constraints). The main idea is to transform this subproblem into an equivalent problem with only nonnegativity constraints, so that the theory and method of RG for such problem can then be applied. To this end, a basic path $p' \in P_i^+$ is chosen among the active paths connecting OD pair i and variable $h_{p'}$ is expressed in terms of the nonbasic variables using $h_{p'} = D_i - \sum_{p \neq p'} h_p$. Substituting this into the objective function Z_i leads to the equivalent problem

$$\begin{aligned} \min_{(h_p : p \neq p')} \quad & \phi_i(h_p : p \neq p') = \sum_{a \in A} \int_0^{x_a(\bar{h}) + \sum_{p \neq p'} \delta_{ap}(h_p - \bar{h}_p) + \delta_{ap'}(D_i - \sum_{p \neq p'} h_p - \bar{h}_{p'})} t_a(v)dv \\ \text{s.t.} \quad & h_p \geq 0 \quad \forall p \in P_i^+ - \{p'\} \end{aligned} \tag{22}$$

The function ϕ_i is a restriction of Z_i onto the feasible region of the equality constraint of subproblem (9), and then is a function of $n - 1$ variables. Not only has the main subproblem replaced by an equivalent problem with one less variable, but also the number of constraints has been reduced by one as well. The latter is a crucial factor, because our experiments (presented in the subsequent section) demonstrate that in practice the number of constraints of subproblem (9) is on average in the range of 2.1 to 2.2, and therefore removing one of them may have a desirable effect on the performance of the algorithm (notice that this range is relevant to the OD pairs with two or more active paths, because those with only one active path are skipped during the RG iterations). In addition, the reduced gradient of Z_i at $\bar{h}$ can be viewed as the gradient of the reduced function ϕ_i evaluated at $(h_p : p \neq p')$. This is because using the chain rule gives the derivatives

$$\frac{\partial \phi_i}{\partial h_p} = \sum_{a \in A} t_a(x_a(\bar{h}))\delta_{ap} - \sum_{a \in A} t_a(x_a(\bar{h}))\delta_{ap'} = \bar{T}_p - \bar{T}_{p'} \quad \forall p \in P_i^+ - \{p'\} \tag{23}$$

which are identical to those given in (14).

Second, many known TAP algorithms generate search directions based on the idea of shifting flow from paths with higher travel times (i.e. costlier paths) to those with the lower travel times (i.e. cheaper paths). These search directions are usually proportional to either the path travel times or path flows; for instance, the search direction of the PG algorithm (Florian, Constantin, and Florian 2009) is linked only to the path travel times, and that of the ODBFW (Chen, Jayakrishnan, and Tsai 2002) only to the path flows. However, the RG method described above benefits from both path travel times and path flows in its direction finding process. Considering the components of the direction computed in Step 3.1, using (15) and (16), we can easily see that the RG algorithm yet shifts flow from costlier paths (than the basic path p') to cheaper paths; however, the rate of shifting from a costlier path is proportional to not only the travel time but also the flow on the path. It seems reasonable from a behavioral perspective because the tendency of a traveler to move from a costlier

path will increase (decrease) as there are more (less) travelers on the path. This can also be seen as a natural consequence of the UE condition of subproblem (9) which includes

$$h_p[T_p(h) - T_{p'}(h)] = 0 \quad \text{if } T_p(h) \geq T_{p'}(h) \tag{24}$$

(see Patriksson 1994, 126). In other words, the dependence of the shifting rate from a costlier path p to the basic path p' upon both $\bar{h}_p$ and $\bar{T}_p - \bar{T}_{p'}$ can naturally speed up the movement of solution toward satisfying the equilibrium condition.

Third, in Step 3.2, the algorithm attempts to find the step length α so as to minimize Z_i in direction d. The restriction $\alpha \geq 0$ is applied because d is a descent direction. However, if $d_p < 0$ for some active path p, a positive step along d may violate the nonnegativity constraints. Therefore, the maximum step length for which $h_p = \bar{h}_p + \alpha\, d_p$ remains feasible for all p is obtained as

$$\alpha_{\max} = \min\left\{-\frac{\bar{h}_p}{d_p} : d_p < 0, p \in P_i^+\right\} \tag{25}$$

and combining this with (15) yields

$$\alpha_{\max} \leq \min\left\{-\frac{1}{\bar{T}_p - \bar{T}_{p'}} : d_p = \bar{h}_p(\bar{T}_p - \bar{T}_{p'}) < 0, p \in P_i^+ - \{p'\}\right\} \tag{26}$$

As a result, the step size α is prevented to be unduly small when $\bar{h}_p$ is small for some nonbasic path p (see Bazaraa, Sherali, and Shetty 2006, 603), especially at the first iterations of the algorithm where $\bar{T}_p - \bar{T}_{p'}$ is somewhat large. In addition, this feature prohibits the line search interval to be unnecessarily large when $\bar{h}_p$ is large and $\bar{T}_p - \bar{T}_{p'}$ is very small. This happens at the last iterations where the solution gets closer to the optimum. It is worth mentioning that unnecessary small step sizes will deteriorate the overall convergence of the algorithm and excessive large search intervals will raise the computational burden, especially for interval reduction methods like bisection.

It also should be noted that the RG direction is very easy to compute in Step 3.1 of the proposed algorithm, because the information on the active paths are stored while they are being processed during execution.

Numerical experiments

In this section, the RG algorithm is tested on the real-life networks of Philadelphia and Chicago, and its performance is compared to some other competitive TAP algorithms. Table 1 lists a summary of the main characteristics of these networks, including the number of zones, nodes, links, and OD pairs, and the total number of trips. The detailed information about the networks is provided from Bar-Gera's website (Bar-Gera 2015) for the purposes of the research.

A sensitivity analysis is performed to determine the effects of alternative choices of basic path on the convergence rate of the RG algorithm. Besides, a direct comparison between the RG algorithm and the algorithms of FW (LeBlanc, Morlok, and Pierskalla 1975), OBA (Bar-Gera 2002), ODFWTQP (Javani and Babazadeh 2017) and GREEDY (Xie, Nie, and Liu 2018) is made by looking at their relative performance. The RG, FW, ODFWTQP and GREEDY algorithms are coded by the authors in C++ language (in Microsoft Visual Studio 2013) using

the same programing framework where the algorithms share subroutines as many as possible. The executable code of the OBA algorithm is obtained from the webpage of the Open Channel Foundation (2014), and compared to the mentioned algorithms for the same networks. All of these experiments are conducted on a PC with a 3.4 GHZ Core i7 processor and 8 GB of RAM.

In addition, to have a better evaluation of the RG algorithm, it is directly compared with the BFW algorithm (Mitradjieva and Lindberg 2013) and the PG algorithm (Florian, Constantin, and Florian 2009) using the SOLA and the path-based traffic assignment modules included in the Emme 4 software (INRO 2017); and with LUCE algorithm (Gentile 2014) using the equilibrium assignment LUCE procedure developed in the Visum 16 software (PTV 2016). These comparisons are made for the same test networks and using the same computer as above. Furthermore, an indirect comparison between RG and B algorithms is carried out on the Chicago network. In this experiment, the OBA algorithm is used as the benchmark, because its performance against B algorithm is reported in Dial (2006).

The bisection (Bolzano) method is used to compute the step size in Step 3.2 of the RG algorithm. It initiates from the interval [0, $\alpha_{\max}$], where $\alpha_{\max}$ is determined using (25), and iteratively reduces the current interval by cutting it in half at each iteration. The method will stop after 35 cuts or whenever the following stopping criterion is satisfied:

$$|\nabla Z_i(\bar{h} + \alpha d)d| \leq \eta |\nabla Z_i(\bar{h})d| \tag{27}$$

where η is set to half the relative gap attained at the last outer iteration. Hence, more precise line searches are performed as the algorithm iterates. The above criterion states that the linesearch will be terminated if the absolute value of the gradient of Z_i with respect to α at current α is smaller than or equal to the absolute value of the gradient of Z_i with respect to α at $\alpha = 0$ multiplied by the small value of η.

It is noteworthy to mention that both the parameters γ and ε used in Steps 3 and 4 of the algorithm are set to 0.1. In addition, the iterative procedure of solving the problem at each major iteration will be truncated after 100 minor iterations in case the stopping criterion in Step 4 is not met. Also, in Step 3, each single-OD subproblem will be solved at most 2 times if the corresponding stopping criterion is not satisfied.

According to section 'RG method', the overall convergence rate of the RG algorithm highly depends on the choice of the basic path among the active paths (between each OD pair) in the direction finding step of the RG algorithm. Although the nonlinear programing texts (e.g. Bazaraa, Sherali, and Shetty 2006, 605) suggest the basic variables should be the largest ones, a sensitivity analysis is conducted by us to realize whether or not the path with the highest flow tends to work better than the paths with less flows. There are a number of candidate strategies that may be considered for choosing the basic path. It is well-known that the TAP algorithms usually shift flows from paths with higher travel times

Table 1. Specifications of test networks.

Network	No. of Zones	No. of Nodes	No. of Links	No. of OD Pairs	Total No. of Trips
Philadelphia	1525	13,389	40,003	1,151,166	18,503,872
Chicago	1790	12,982	39,018	3,136,441	1,360,428

to those with lower travel times. Hence, the most sensible strategies would be choosing a path whose travel time is the highest among the positive-flow paths or the lowest among all paths. The former moves flows from the costliest path to other paths, while the latter shifts flows to the cheapest path from others. Another strategy is choosing the path whose travel time is closest to the average of the path travel times. This is inspired by the earlier work of Florian, Constantin, and Florian (2009), and is based on the idea of shifting flows from the paths with higher costs than the average to those with lower costs. Accordingly, the RG algorithm is implemented on the test networks for each of the following path choice strategies:

- Choosing a path with the largest flow (RG-hmax)
- Choosing a positive-flow path with the maximum travel time (RG-Tmax)
- Choosing a path with the minimum travel time (RG-Tmin)
- Choosing a positive-flow path with the travel time closest to the average (RG-Tmean).

Figures 1 and 2 compare the convergence rate of the RG algorithm by using each of the above path choice strategies for the Philadelphia and Chicago test networks, respectively. These figures tabulate the evolution of the relative gap with respect to the CPU time until an *RGap* of 1.0E-14 is achieved. For Philadelphia, Figure 1 shows the RG-Tmin algorithm can achieve an *RGap* of 1.0E-14 in 11.4 minutes. This significantly outperforms the RG-hmax and RG-Tmax algorithms, which consumed respectively 15.9 and 17.3 minutes, and the RG-Tmean algorithm, which has the worst convergence rate. For the Chicago network, based on Figure 2, the RG-Tmin algorithm performs the best for relative gaps down to 1.0E-9, while at the smaller relative gaps the RG-hmax algorithm tends to dominate. However, their ultimate performance are not markedly different, as they approach to *RGap* of 1.0E-14 in 24.1 and 25.7 minutes, respectively. In comparison, less satisfaction is gained from the two other algorithms, of which the RG-Tmax demonstrates the worst performance with about 33 minutes CPU time needed to reach the target relative gap. Based on this sensitivity analysis, the convergence rate of the RG algorithm is mainly related to how the basic path is chosen; and it is overall the best when the lowest travel time path is selected. In effect, we choose the path with the minimum travel time as the basic path in Step 3.1 of the RG algorithm hereinafter.

In our implementation of the RG algorithm, the subproblem of each OD pair *i* is solved (in Step 3) for 2 times, or until it reaches a restricted gap less than a fraction γ of the last average gap that was computed (i.e. $gap_i < \gamma \cdot AGap$). As a result, the smaller the parameter γ, the more the subproblems are solved and the more precise the solutions are. A sensitivity analysis is performed to explore the effect of changing the value of parameter γ on the performance of the algorithm. Figure 3 illustrates the CPU time spent by the RG algorithm, as a function of γ, to reach *RGap* of 1.0E-14 for the Philadelphia and Chicago test problems. As can be seen, in both experiments, the CPU time fluctuates widely when γ varies in the range of almost 0 to 1, hitting a low at $\gamma = 0.1$, but it rises sharply as γ increases beyond 1. Therefore, the best performance is achieved when γ is set equal to 0.1.

Table 2 is prepared to show the computational details of the RG algorithm for the test networks. To obtain an *RGap* of 1.0E-14, it performs 20 iterations in 683.5 seconds (11.4 minutes) for Philadelphia and 30 iterations in 1540 seconds (25.7 minutes) for Chicago. In this table the rows '*CPU time of finding the shortest paths*' and '*CPU time of updating the active*

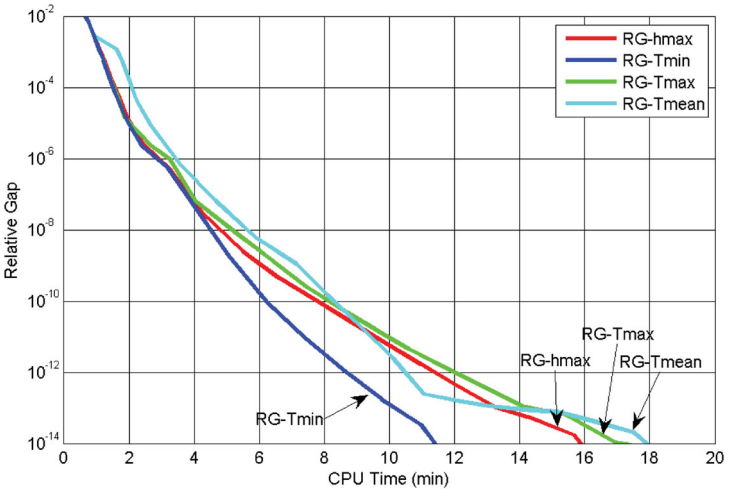

Figure 1. Convergence rates of the RG algorithm with different choices of basic path for Philadelphia.

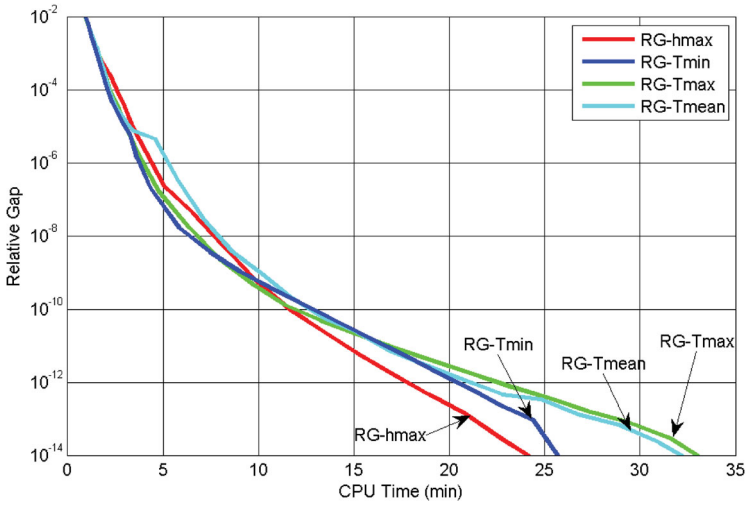

Figure 2. Convergence rates of the RG algorithm with different choices of basic path for Chicago.

path sets' are corresponding to Step 1 of the RG algorithm. The former represents the CPU time spent on solving the shortest path problems, and the latter presents the CPU time spent on adding the shortest paths to the active path sets. The rows '*CPU time of finding the feasible directions*' and '*CPU time of performing the line searches*' show the CPU times corresponding to Steps 3.1 and 3.2 of the algorithm, respectively. The former in addition contains the CPU time to calculate path travel times throughout Step 3 regarded as necessary for Step 3.1. Table 2 also illustrates the total memory used for storing path and link structures, maximum number of active paths per OD pair, maximum number of arcs per active path, the total number of active paths and the average number of active paths per OD pair in the final solutions.

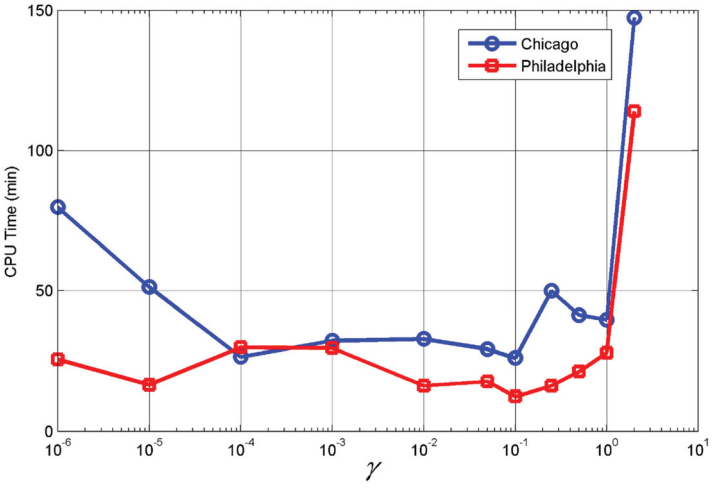

Figure 3. Sensitivity to parameter γ on convergence of the RG algorithm to relative gap of 1.0E-14.

Table 2. Computational details of the RG algorithm to relative gap of 1.0E-14.

	Network	
Computational details	Philadelphia	Chicago
Total no. of iterations	20	30
Total CPU time (sec)	683.5	1540
CPU time of finding the shortest paths (sec)	120.7	135.1
CPU time of updating the active path sets (sec)	18.4	44.4
CPU time of finding the feasible directions (sec)	406.7	927.9
CPU time of performing the line searches (sec)	155.6	350.2
Total memory usage (MB)	491	868
Maximum no. of active paths per OD pair	11	11
Maximum no. of arcs per active path	234	183
Total no. of active paths in the final solution	1,591,567	2,904,338
Average no. of active paths per OD pair in the final solution	1.38	1.26
Average no. of active paths per OD pair in the final solution, excluding OD pairs with just one active path	2.18	2.12

According to Table 2, the CPU time to solve the subproblems (consisting mainly of direction finding and line search) is about 82 percent of the total CPU time for Philadelphia, and about 83 percent of that for Chicago. The CPU times of finding the RG directions and performing the line searches for Chicago are about 2.3 times the Philadelphia's. Table 2 also reveals that the maximum number of arcs per path for Philadelphia is greater than Chicago, providing an insight about the lengths of the longest paths on these networks. In effect, solving the shortest path problems for Philadelphia is more burdensome than Chicago. As can be seen in the table, the total CPU time required to solve the shortest path problems for Philadelphia is about 0.9 times that for Chicago, however, the average CPU time for finding the shortest paths per iteration is 6.04 seconds for Philadelphia and 4.5 seconds for Chicago. Table 2 also shows the maximum number of active paths per OD pair (observed during running the code) equals 11 for both Philadelphia and Chicago. Moreover, the total number of active paths in the final solution of Chicago is about twice that of Philadelphia. More active paths causes larger memory usage and longer CPU time for storing and updating the path

Table 3. Details of the line searches used by the RG algorithm to relative gap of 1.0E-14.

Network	Outer iteration(s)	Total number of line searches	Average step size	Percentage of time maximum step size is optimum
Philadelphia	First	140,238	0.38	98.89
	Last	2,532,216	4.13E6	0.0002
	All	14,571,560	7.43E5	10.34
Chicago	First	149,771	0.42	99.53
	Last	3,142,543	2.47E8	0.0012
	All	36,737,155	2.39E7	7.63

information. Finally, the results of Table 2 indicate that the average number of active paths in the final solutions of the test networks is of the order of 1.3 to 1.4 paths per OD pair, and is about 2.1 to 2.2 paths per OD pair when the OD pairs with more than one path are considered. It is to note that in Step 3.3 of the RG algorithm, the paths which carry positive flows smaller than 1.0E-13 are eliminated from the active path sets.

In another experiment, we investigate how large optimal step lengths found by the algorithm (in Step 3.2) are and how often the maximum step lengths calculated by (25) are also the optimal ones. Table 3 provides additional results of the line searches performed on the test problems. As can be seen, there are a total of over 14 million line searches carried out in the case of Philadelphia and over 36 million in the case of Chicago. Comparing the first and the last outer iterations, in both instances, the number of line searches per iteration grows sharply as the algorithm comes close to the solution. For Philadelphia, there are 140,238 cases solved in the first iteration and 2,532,216 solved in the last iteration; and for Chicago, there are 149,771 and 3,142,543 of such cases, respectively. The average of the step sizes computed in the first iteration is 0.38 for Philadelphia and 0.42 for Chicago. However, the step size values substantially increase as the iteration number rises, because the travel times of the active paths become closer and so the norms of search directions decrease. Moreover, we know that the bisection method searches an interval between zero and the maximum allowed step size to find an optimal step size for each subproblem. Based on the results given in the last column of Table 3, in both networks, the maximum step sizes are also the optimal ones above 99% of the time in the first iteration; but this is not the case almost all the time in the last iteration.

The performances of RG, FW, OBA, ODFWTQP and GREEDY algorithms for Philadelphia are compared in Figure 4, verifying the superiority of the RG algorithm over all others. While FW is not capable of reaching an *RGap* of 1.0E-5 even after two hours (and about 1100 iterations) and OBA cannot attain an *RGap* of 1.0E-7 in the same duration, the RG algorithm reaches an *RGap* of 1.0E-14 in 11.4 minutes. It also outperforms the ODFWTQP algorithm, especially seeing that ODFWTQP cannot gain an *RGap* of 1.0E-9 after a running time of two hours (and about 1000 iterations). The figure also demonstrates that the RG algorithm is superior to the GREEDY algorithm for relative gaps down to 1.0E-14.

Figure 5 shows the evolution of *RGap* with respect to CPU time during the iterations of the algorithms RG, FW, OBA, ODFWTQP and GREEDY for Chicago. The RG algorithm strictly outperforms the other ones and is able to achieve an *RGap* of 1.0E-14 in 25.7 minutes. The FW algorithm cannot obtain an *RGap* of 1.0E-5 even after three hours (and 2500 iterations), and the OBA algorithm cannot reach an *RGap* of 1.0E-7 in the same length of time. Also, analogous to the results given above for Philadelphia, the asymptotic behavior of the ODFWTQP

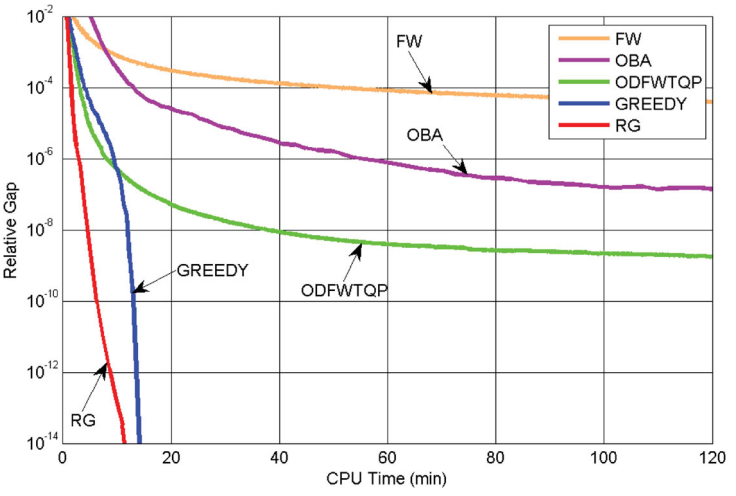

Figure 4. Convergence rates of algorithms FW, OBA, ODFWTQP, GREEDY and RG for Philadelphia.

algorithm occurs for relative gaps less than 1.0E-8, so that it is not capable of gaining an *RGap* of 1.0E-10 even after three hours (and about 1500 iterations). The figure also reveals the ability of the GREEDY algorithm to attain the target relative gap in 37.3 minutes, whereas the RG algorithm makes the same progress about 12 minutes earlier. The RG and GREEDY algorithms are similar as they both apply the column generation technique, the One-OD-At-A-Time flow update strategy and the simultaneous active path set update strategy; and both decompose the problem in terms of OD pairs. However, the major difference between these two algorithms lies in how they solve the decomposed subproblems. The GREEDY algorithm approximates each subproblem by a quadratic problem using the second-order Taylor expansion, and then solves the KKT conditions of each quadratic approximation in a greedy way, without any line search. This is obviously unlike the RG algorithm, which relies on the first-order approximation and performs the line search as well. It is worth mentioning that Xie, Nie, and Liu (2018) claimed the GREEDY algorithm outperforms the bush-based algorithms such as TAPAS by a broad margin for Philadelphia and Chicago. Consequently, given our numerical results, RG is at least comparable to the state-of-the-art bush-based algorithms like TAPAS.

An experiment is conducted here to investigate the performance of the RG algorithm against other recently proposed TAP algorithms coded within well-known software packages. With regard to this, the SOLA[1] and the path-based traffic assignment modules of Emme 4 software (INRO 2017) are used to apply the BFW and the PG algorithms, respectively, to solve the Philadelphia and Chicago test problems. Further, to make a comparison with an origin-based algorithm, the equilibrium assignment LUCE procedure of Visum 16 software (PTV 2016) is applied to the test networks. In this experiment, again, the relative gap of 1.0E-14 is considered as the target precision, although the assignment algorithms embedded in Emme cannot attain relative gaps below 1.0E-7.[2]

Figure 6 compares the convergence rates of the BFW (Emme 4), PG (Emme 4), LUCE (Visum 16) and RG algorithms for the Philadelphia test problem. The figure demonstrates that RG is marginally more efficient than LUCE for relative gaps larger than 1.0E-8, whereas

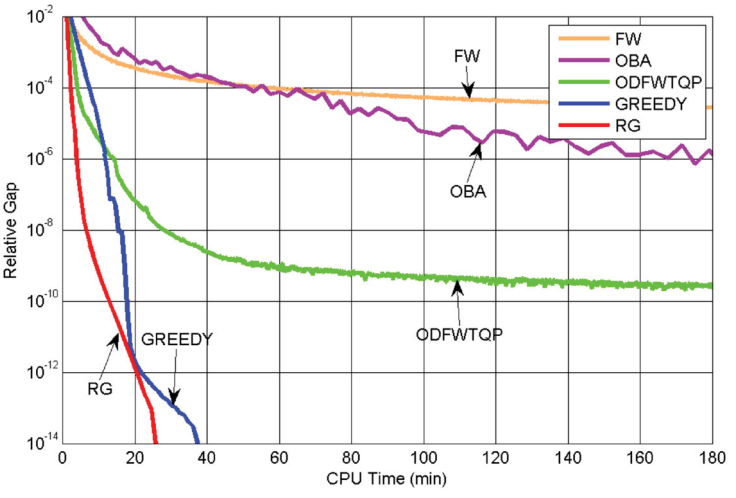

Figure 5. Convergence rates of algorithms FW, OBA, ODFWTQP, GREEDY and RG for Chicago.

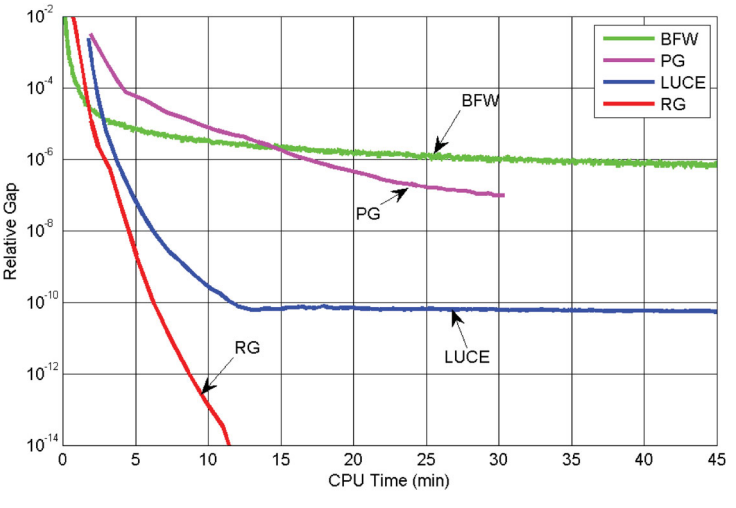

Figure 6. Convergence rates of algorithms BFW, PG, LUCE and RG for Philadelphia.

is strictly more efficient than it for smaller relative gaps. The RG algorithm gains an *RGap* of 1.0E-14 in about 11.4 minutes, whereas the asymptotic behavior of LUCE occurring after the relative gap of 1.0E-10 makes it unable to reach an *RGap* of 1.0E-11 even after 45 minutes and 223 iterations. Figure 6 also shows that RG clearly wins PG for relative gaps down to 1.0E-7 in the Philadelphia problem. According to this figure, RG achieves an *RGap* of 1.0E-7 in 3.73 minutes and after 14 iterations, while PG does the same in 30.38 minutes and after 245 iterations and BFW is unable to do the same even after 45 minutes and 6000 iterations. The results imply that, to reach an *RGap* of 1.0E-7, RG requires about 88 percent less CPU time (i.e. works about 8 times faster) than does PG for Philadelphia.

Figure 7 is prepared to compare the computational efforts required by the algorithms BFW, PG, LUCE and RG to reach *RGap* of 1.0E-14 for the Chicago problem. According to

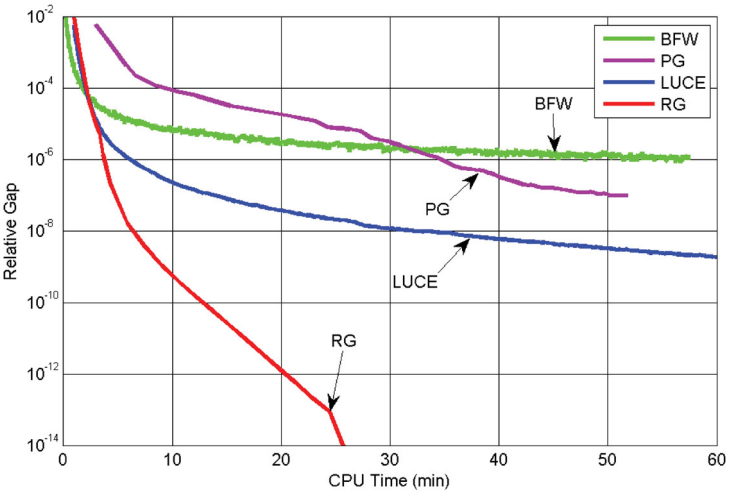

Figure 7. Convergence rates of algorithms BFW, PG, LUCE and RG for Chicago.

Table 4. Relative CPU times of algorithms OBA, B and RG to relative gap of 1.0E-4 for Chicago.

OBA	B	RG
1	0.171	0.041

this figure, BFW and LUCE are slightly faster than RG for relative gaps greater than 1.0E-4; however, RG absolutely outperforms them for smaller relative gaps. The RG algorithm gains the target relative gap in 25.7 minutes, but LUCE cannot reach even an *RGap* of 1.0E-9 after one hour and 230 iterations. As also illustrated in Figure 7, the algorithms RG and PG reach an *RGap* of 1.0E-7 in 4.75 minutes (and after 18 iterations) and in 51.80 minutes (and after 270 iterations), respectively, while the BFW cannot attain the same relative gap in one hour (and after 6000 iterations). Based on the results, the RG algorithm needs about 91 percent less CPU time (i.e. is about 11 times faster) than PG to gain an *RGap* of 1.0E-7 for Chicago.

In addition, an indirect comparison with the algorithm B is performed for the Chicago problem, based on the results given in Dial (2006). He compared the CPU times needed by the algorithms B and OBA on the same computer for Chicago, showing that B achieved an *RGap* of 1.0E-4 in 30 minutes while OBA reached it in 175 minutes. These results can help us compare RG with algorithm B using the OBA code available on Bar-Gera's website (Bar-Gera 2015) as the benchmark. On our computer (a 3.4 GHZ Core i7 processor and 8 GB of RAM), the algorithms RG and OBA gain an *RGap* of 1.0E-4 in 2.08 and 51.1 minutes, respectively. Table 4 represents the relative CPU times of the algorithms B and RG versus the OBA, after the CPU times of OBA for our run and Dial's were normalized to 1. According to this table, the proposed RG algorithm works about 4.2 times faster than algorithm B.

The above experiments disclose that the RG method can successfully deal with the OD-based subproblems (9) within Step 3 of the proposed RG algorithm (our experiments involved subproblems with up to 11 path variables). However, readers may wish to know how fast the RG method is in comparison to commercial solvers. It is to note that linking an

Table 5. CPU times of RG method and GAMS/CONOPT for single-OD subproblems.

Network	Subproblem no.	No. of active paths	Target restricted gap	CPU time over 10,000 runs (sec)	
				RG method	GAMS/CONOPT
Philadelphia	1	2	1.9E-4	0.003	53.9
	2	5	8.8E-6	0.008	146.1
	3	10	5.0E-8	0.106	235.2
Chicago	4	2	2.3E-3	0.002	52.6
	5	5	4.8E-5	0.009	151.3
	6	10	3.1E-8	0.063	217.8

external solver with the main C++ code of the proposed algorithm is the most crucial issue for applying such a solver, because extra subroutines should be coded in order to exchange data between the solver and the code. Instead, we compare the computational efficiency of the RG method for solving the subproblems only (instead of the entire problem) against the CONOPT solver of GAMS optimization package (GAMS 2018), a powerful commercial solver for nonlinear programing problems. To this end, three subproblems involving 2, 5 and 10 active paths are selected randomly among all those encountered during the previous runs of the RG algorithm for Philadelphia, and the same is performed for the case of Chicago. Each subproblem is solved by RG iterations (i.e. iterating Steps 3.1, 3.2 and 3.3 without eliminating the zero-flow paths) as well as by GAMS/CONOPT (using GAMS C++ API) to a target restricted gap gap_i set to the maximal precision that GAMS can achieve,[3] and each run is repeated 10,000 times in order to ease the comparison of the running times (the times spent on constructing GAMS input files are excluded). Table 5 reports the performances of the RG method and GAMS/CONOPT on the six selected subproblems, run on the same computer as before. It is readily noticeable in the table that the RG method outperforms GAMS significantly.

Summary and conclusions

A variant of the reduced gradient (RG) method of Wolfe was used to develop a fast convergence path-based algorithm for the static traffic assignment problem (TAP). The algorithm decomposes the problem in terms of origin-destination (OD) pairs, and solves each subproblem by the RG method after applying a column generation technique and the One-OD-At-A-Time flow update strategy. The real-life networks of Philadelphia and Chicago were selected for the purposes of the research, and a sensitivity analysis was conducted on them to realize the best strategy for choosing a basic path for each OD pair in the direction finding step of the algorithm. The results show that better performance is achieved when using the path with the minimum travel time as the basic path.

The performance of the suggested RG algorithm was investigated on the test networks by showing that it can reach a relative gap of 1.0E-14 in reasonable CPU times. The algorithm was directly compared to the conventional Frank and Wolfe (FW) and the well-known origin-based (OBA) algorithms; the commercial versions of the local user cost equilibrium (LUCE), the bi-conjugate FW (BFW) and the projected gradient (PG) algorithms; the more recent OD-based truncated quadratic programing (ODFWTQP) algorithm; and a newly developed greedy (GREEDY) algorithm; and indirectly compared to algorithm B

as well. The results of these comparisons elucidated that the algorithm is highly efficient, especially when very precise solutions are required.

Over the past two decades, more than 20 papers have been published on the comparison of the performance of the existing TAP algorithms (see the papers mentioned in section 'Introduction'). Although the results of these studies cannot be used to draw a definitive conclusion on the fastest or the most efficient algorithm (mostly because of using different programing skills and computers), each of which gives us valuable insights into this area. The test problems of Philadelphia and Chicago have been often used in these experiments, and the *RGap* is the most used measure for assessing the convergence of the algorithms. A review of these studies indicates our results are consistent with theirs in light of the following insights. First of all, it can be seen that the FW algorithm is still of interest as a benchmark for comparison purposes, although it is much slower than the newer TAP algorithms. It has been also frequently reported that OBA is faster than FW and can reach *RGap* of 1.0E-7 in reasonable CPU times for Philadelphia and Chicago. The performance of BFW, PG, GP and LUCE have been examined by a number of researchers who reported that these algorithms outperform OBA and are capable of reaching the *RGap* values of about1.0E-8 for large scale networks. Moreover, the algorithms B, TAPAS and GREEDY have been reported able to achieve very high levels of precision, i.e. the *RGap* values of 1.0E-12 to 1.0E-14, for Philadelphia and Chicago in acceptable CPU times.

Another point that should be addressed is what differences exist between the developed RG algorithm in this paper and the DSD algorithm of Larsson and Patriksson (1992). The DSD algorithm applies a scaled version of the same RG method we applied but to an extreme point formulation of subproblems (9), although the directions produced are the same when normalized (provided that the same basic paths are chosen). However, unlike the RG algorithm, the DSD algorithm does not include dropping the active paths with zero flows, and also chooses path with the maximum flow (out of the paths belonging to each OD pair) as the basic path. Moreover, at each main iteration, it performs a line search simultaneously for all OD pairs over total link flow variables, which is well-known to be inefficient compared to the line searches performed individually for each OD pair. Furthermore, Larsson and Patriksson (1992) reported that their DSD algorithm cannot gain precise solutions unless a second order method is used in place of the RG after the convergence rate of the algorithm has become very small. The network of Barcelona (consisting of 1020 nodes, 2525 arcs and 7029 OD pairs) is the largest-size problem they solved, resulted in a solution with a relative error of about 1.0E-2. After all, our suggested RG algorithm controls the convergence of the subproblems, both individually and as a whole, using speed-up techniques.

It is well-known that TAP algorithms that can reach higher precise solutions in less CPU times make the transportation planning results more reliable and stable. Many of these algorithms (like the suggested RG) make use of the first derivatives (or an approximation of the Hessian matrix) of the objective function of subproblems (9) in their direction finding procedure, providing at most a linear rate of convergence. This is while a superlinear rate of convergence is obtainable by using the second order information. An attempt on this issue is the ODFWTQP algorithm by Javani and Babazadeh (2017), where the search direction is obtained by partially solving the quadratic programing (QP) approximations of the subproblems using the FW algorithm. They suggested that the overall convergence rate

of their algorithm is related to the efficiency of the method used to solve the QP subproblems, and will be expectedly enhanced by using more powerful methods than FW. In this regard, the authors plan to apply the suggested RG algorithm within the QP algorithmic framework.

Notes

1. We run SOLA with 8 threads available to us on our PC. Using more threads (e.g. 20 threads) may speed up the algorithm but will use more memory.
2. If a smaller value is specified in Emme's SOLA and path-based traffic assignment modules, it will be converted automatically to 1.0E-7.
3. GAMS only prints up to 8 decimals on the output file.

Acknowledgement

The authors greatly appreciate the thoughtful comments and constructive suggestions of the editor and four anonymous reviewers.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Abbas Babazadeh http://orcid.org/0000-0002-3838-5220
Guido Gentile http://orcid.org/0000-0002-1523-1640

References

Babazadeh, A. 2005. "Equilibrium Transit Assignment Problem in Congested Networks: Formulation and Solution Algorithm." Ph.D. diss., Sharif University of Technology, Tehran, Iran.

Babazadeh A., and H. Z. Aashtiani. 2005. "Algorithm for Equilibrium Transit Assignment Problem." *Transportation Research Record: Journal of the Transportation Research board* 1923: 227–235.

Bar-Gera, H. 2002. "Origin-Based Algorithm for the Traffic Assignment Problem." *Transportation Science* 36 (4): 398–417.

Bar-Gera, H. 2010. "Traffic Assignment by Paired Alternative Segments." *Transportation Research Part B: Methodological* 44 (8): 1022–1046.

Bar-Gera, H. 2015. Transportation Test Networks. Accessed June, 2015. http://www.bgu.ac.il/ barg-era/tntp.

Bazaraa, M. S., H. D. Sherali, and C. M. Shetty. 2006. *Nonlinear Programming: Theory and Algorithms*. 3rd ed. Hoboken: John Wiley & Sons.

Beckmann, M., C. B. McGuire, and C. B. Winsten. 1956. *Studies in the Economics of Transportation*. New Haven, Connecticut: Yale University Press.

Bertsekas, D. 1976. "On the Goldstein-Levitin-Polyak Gradient Projection Method." *IEEE Transactions on Automatic Control* 21 (2): 174–184.

Chen, A. 2001. "Effects of Flow Update Strategies on Implementation of the Frank-Wolfe Algorithm for the Traffic Assignment Problem." *Transportation Research Record: Journal of the Transportation Research Board* 1771: 132–139.

Chen, A., R. Jayakrishnan, and W. K. Tsai. 2002. "Faster Frank-Wolfe Traffic Assignment with New Flow Update Scheme." *Journal of Transportation Engineering* 128 (1): 31–39.

Chen, A., and D. H. Lee. 1999. "Path-Based Algorithms for Large Scale Traffic Equilibrium Problems: A Comparison Between DSD and GP." In *78th Annual Meeting of the Transportation Research Board*, Washington, DC.

Chen, A., D. H. Lee, and R. Jayakrishnan. 2002. "Computational Study of State-of-the-Art Path-Based Traffic Assignment Algorithms." *Mathematics and Computers in Simulation* 59 (6): 509–518.
Chen, A., X. Xu, S. Ryu, and Z. Zhou. 2013. "A Self-Adaptive Armijo Stepsize Strategy with Application to Traffic Assignment Models and Algorithms." *Transportmetrica A: Transport Science* 9 (8): 695–712.
Dafermos, S. C., and F. T. Sparrow. 1969. "The Traffic Assignment Problem for a General Network." *Journal of Research of the National Bureau of Standards B* 73 (2): 91–118.
Dial, R. B. 2006. "A Path-Based User-Equilibrium Traffic Assignment Algorithm That Obviates Path Storage and Enumeration." *Transportation Research Part B: Methodological* 40 (10): 917–936.
Di Lorenzo, D., A. Galligari, and M. Sciandrone. 2015. "A Convergent and Efficient Decomposition Method for the Traffic Assignment Problem." *Computational Optimization and Applications* 60 (1): 151–170.
Emme release 4.3.5. 2017. INRO Consultants, Inc., Montreal, Canada.
Florian, M., I. Constantin, and D. Florian. 2009. "A New Look at Projected Gradient Method for Equilibrium Assignment." *Transportation Research Record: Journal of the Transportation Research Board* 2090: 10–16.
Florian, M., and S. Nguyen. 1974. "A Method for Computing Network Equilibrium with Elastic Demands." *Transportation Science* 8 (4): 321–332.
Frank, M., and P. Wolfe. 1956. "An Algorithm for Quadratic Programming." *Naval Research Logistics Quarterly* 3 (1-2): 95–110.
Galligari, A., and M. Sciandrone. 2018. "A Convergent and Fast Path Equilibration Algorithm for the Traffic Assignment Problem." *Optimization Methods and Software* 33 (2): 354–371.
General Algebraic Modeling System (GAMS) release 25.1.2. 2018. GAMS Development Corporation, Fairfax, VA, USA.
Gentile, G. 2014. "Local User Cost Equilibrium: a Bush-Based Algorithm for Traffic Assignment." *Transportmetrica A: Transport Science* 10 (1): 15–54.
Gentile, G. 2016. "Solving a Dynamic User Equilibrium model based on splitting rates with Gradient Projection algorithms." *Transportation Research Part B: Methodological* 92: 120–147.
Holmgren, J., and P. O. Lindberg. 2014. "Upright Stiff: Subproblem Updating in the FW Method for Traffic Assignment." *EURO Journal on Transportation and Logistics* 3 (3-4): 205–225.
Inoue, S. I., and T. Maruyama. 2012. "Computational Experience on Advanced Algorithms for User Equilibrium Traffic Assignment Problem and its Convergence Error." *Procedia-Social and Behavioral Sciences* 43: 445–456.
Javani, B., and A. Babazadeh. 2017. "Origin-Destination-Based Truncated Quadratic Programming Algorithm for Traffic Assignment Problem." *Transportation Letters* 9 (3): 166–176.
Jayakrishnan, R., W. T. Tsai, J. N. Prashker, and S. Rajadhyaksha. 1994. "A Faster Path-Based Algorithm for Traffic Assignment." *Transportation Research Record: Journal of the Transportation Research Board* 1443: 75–83.
Kumar, A., and S. Peeta. 2010. "Slope-Based Multipath Flow Update Algorithm for Static User Equilibrium Traffic Assignment Problem." *Transportation Research Record: Journal of the Transportation Research Board* 2196: 1–10.
Kumar, A., and S. Peeta. 2014a. "Slope-Based Path Shift Propensity Algorithm for the Static Traffic Assignment Problem." *International Journal for Traffic and Transport Engineering* 4 (3): 297–319.
Kumar, A., and S. Peeta. 2014b. "Strategies to Enhance the Performance of Path-Based Static Traffic Assignment Algorithms." *Computer-Aided Civil and Infrastructure Engineering* 29 (5): 330–341.
Kumar, A., S. Peeta, and Y. Nie. 2012. "Update Strategies for Restricted Master Problems for User Equilibrium Traffic Assignment Problem: Computational Study." *Transportation Research Record: Journal of the Transportation Research Board* 2283: 131–142.
Larsson, T., and M. Patriksson. 1992. "Simplicial Decomposition with Disaggregated Representation for the Traffic Assignment Problem." *Transportation Science* 26 (1): 4–17.
LeBlanc, L. J., E. K. Morlok, and W. P. Pierskalla. 1975. "An Efficient Approach to Solving the Road Network Equilibrium Traffic Assignment Problem." *Transportation Research* 9 (5): 309–318.
Lee, D. H., Y. Nie, and A. Chen. 2003. "A Conjugate Gradient Projection Algorithm for the Traffic Assignment Problem." *Mathematical and Computer Modelling* 37 (7-8): 863–878.

Luenberger, D. G., and Y. Ye. 2008. *Linear and Nonlinear Programming*. 3rd ed. Volume 116 of International Series in Operations Research & Management Science. New York: Springer.
McCormick, G. P. 1970. "The Variable Reduction Method for Nonlinear Programming." *Management Science* 17 (3): 146–160.
Mitradjieva, M., and P. O. Lindberg. 2013. "The Stiff is Moving—Conjugate Direction Frank-Wolfe Methods with Applications to Traffic Assignment." *Transportation Science* 47 (2): 280–293.
Nguyen, S. 1974. "An Algorithm for the Traffic Assignment Problem." *Transportation Science* 8 (3): 203–216.
Nie, Y. M. 2010. "A Class of Bush-Based Algorithms for the Traffic Assignment Problem." *Transportation Research Part B: Methodological* 44 (1): 73–89.
Open Channel Foundation. 2014. Accessed October, 2014. http://www.openchannelfoundation.org.
Patriksson, M. 1994. *The Traffic Assignment Problem: Models and Methods*. Utrecht, Netherlands: VSP.
Perederieieva, O., M. Ehrgott, A. Raith, and J. Y. Wang. 2015. "A Framework for and Empirical Study of Algorithms for Traffic Assignment." *Computers & Operations Research* 54: 90–107.
Perederieieva, O., M. Ehrgott, A. Raith, and J. Y. Wang. 2016. "Numerical Stability of Path-Based Algorithms for Traffic Assignment." *Optimization Methods and Software* 31 (1): 53–67.
Rosen, J. B. 1960. "The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints." *Journal of the Society for Industrial and Applied Mathematics* 8 (1): 181–217.
Sheffi, Y. 1985. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Englewood Cliffs, NJ: Prentice Hall.
Visum release 16.01-12. 2016. PTV Group., Karlsruhe, Germany.
Wardrop, J. G. 1952. "Road Paper. Some Theoretical Aspects of Road Traffic Research." *Proceedings of the Institution of Civil Engineers* 1 (3): 325–362.
Wolfe, P. 1967. "Methods of Nonlinear Programming. Chapter 6 of Nonlinear Programming." In *Interscience*, edited by J. Abadie, 97–131. New York: John Wiley.
Wolfe, P. 1972. "On the Convergence of Gradient Methods Under Constraint." *IBM Journal of Research and Development* 16 (4): 407–411.
Xie, J., Y. Nie, and X. Liu. 2018. "A Greedy Path-Based Algorithm for Traffic Assignment." In *Transportation Research Board 97th Annual Meeting*, Washington DC, USA, January 7–11. Transportation Research Board.
Xie, J., and C. Xie. 2015. "Origin-Based Algorithms for Traffic Assignment: Algorithmic Structure, Complexity Analysis, and Convergence Performance." *Transportation Research Record: Journal of the Transportation Research Board* 2498: 46–55.
Zangwill, W. I. 1969. *Nonlinear Programming: A Unified Approach*. Englewood Cliffs, NJ: Prentice-Hall.
Zheng, H. 2015. "Adaptation of Network Simplex for the Traffic Assignment Problem." *Transportation Science* 49 (3): 543–558.