This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier’s archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright
A class of bush-based algorithms for the traffic assignment problem

Yu (Marco) Nie *
Department of Civil and Environmental Engineering, Northwestern University, Evanston, IL 60208, United States

ARTICLE INFO
Article history:
Received 2 March 2009
Received in revised form 12 June 2009
Accepted 12 June 2009

Keywords:
Bush-based algorithm
User equilibrium traffic assignment
Acyclicity

ABSTRACT
This paper studies a class of bush-based algorithms (BA) for the user equilibrium (UE) traffic assignment problem, which promise to produce highly precise solutions by exploiting acyclicity of UE flows. Each of the two building blocks of BA, namely the construction of acyclic sub-networks (bush) and the solution of restricted master problems (RMP), is examined and further developed. Four Newton-type algorithms for solving RMP, which can be broadly categorized as route flow and origin flow based, are presented, of which one is newly developed in this paper. Similarities and differences between these algorithms, as well as the relevant implementation issues are discussed in great details. A comprehensive numerical study is conducted using both real and randomly generated networks, which reveals that the relative performance of the algorithms is consistent with the analysis. In particular, the results suggest that swapping flows between shortest and longest route segments consistently outperforms other RMP solution techniques.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

Searching for efficient solution algorithms for the static user equilibrium (UE) traffic assignment problem (TAP) is one of the most studied subjects in transportation research (Patriksson, 1994; Florian and Hearn, 1995). In part, the continual enthusiasm for this over 50-year old problem (Wardrop, 1952; Beckmann et al., 1956) was driven by the inadequacy of existing algorithms in dealing with large-scale urban travel forecasting applications. For example, Boyce et al. (2004) showed that the Frank–Wolfe algorithm (Frank and Wolfe, 1956; Leblanc et al., 1975) which has been a standard TAP solver for over a quarter century, is unable to obtain an equilibrium solution precise enough to fulfill basic planning functions. Probably more important, TAP presents challenges for academic research owing to its rich modeling possibilities, special structure and great size in real-world applications (Patriksson, 1994).

Formulations and algorithms for TAP typically operate in the space of link flows (e.g. Leblanc et al., 1975; Hearn et al., 1985; Fukushima, 1985) or route flows (e.g. Gibert, 1968; Dafermos and Sparrow, 1969; Leventhal et al., 1973; Bertsekas, 1976; Bertsekas and Gafni, 1983; Larsson and Patriksson, 1992; Jayakrishnan et al., 1994). In his seminal paper on logit assignment, Dial (1971) showed that traffic assignment can be performed more efficiently in an acyclic network rooted at an origin (called bush), since flows from the origin to all destinations can be assigned in parallel. The similar concept was employed in solving minimum-delay routing problem in communication networks, which is in effect equivalent to traffic assignment problem (Gallager, 1977; Bertsekas et al., 1984). The bush concept is significant not only because it offers efficiency, but, more importantly, because it warrants optimality, in the sense that both UE and system-optimal (SO) flows can be represented by a bush (e.g. Newell, 1980). A number of recent advances have confirmed the great utility of the bush concept in designing efficient algorithms for solving transportation network problems. Dial (1999) invented a bush-based algorithm for finding minimum-revenue toll, explicitly taking the advantage of acyclicity of SO flows. Bar-Gera (2002) and Dial...
In general, BA is found to be more efficient compared to link-based or route-based counterparts, in that it requires modest computational time and memory to produce highly precise UE solutions.

According to the flow aggregation level in bush equilibration, BA can be classified as route flow or origin flow based. In the former class is Dial’s algorithm (2006), which performs traffic assignment by swapping flows between the longest and shortest routes. We note that Dafermos (1971) proposed a similar algorithm and proved its convergence, albeit her algorithm has been regarded infeasible for a long time because finding longest routes is NP-hard in a general network. Dial’s algorithm effectively operationalizes this idea by exploiting acyclicity of bushes. Along the same line of Gallager (1977) and Bertsekas et al. (1984), Bar-Gera’s origin-based algorithm (OBA) (1999, 2002) operates in the space of origin flows, which are represented by proportions of traffic arriving at each node from its predecessor links. Bar-Gera (2002) employed a projected quasi-Newton method to update origin flows, which has some similarities to Bertsekas (1976) but is innovative in several aspects. Recently, Nie (2007) showed, both theoretically and experimentally, that Bar-Gera’s original algorithm can be substantially improved using a different method to estimate the Hessian. Little is known about the relative performance of the two classes of BA, since they both are relatively new. Dial (2006) reported that his algorithm is much faster than Bar-Gera’s. Slavin et al. (2006) has made similar observations. However, both comparisons were made between executable codes (they both used the executable made available by Bar-Gera), and therefore maybe subject to uncontrolled implementation factors. In addition, these comparisons did not consider the aforementioned improvements to Bar-Gera’s algorithm (Nie, 2007).

This paper surveys and documents the recent developments of the bush-based traffic assignment algorithms. A coherent framework is proposed to organize the algorithms, to reveal their similarities and differences, and perhaps more importantly, to explain the observed computational performance which allows insights to be drawn for developing more efficient traffic assignment algorithms. The paper also contributes several new additions to the family of bush-based algorithms. First, it proposes new strategies for the construction of bushes, whose details, although critical to the performance of the algorithm, have not been fully revealed in the literature. Second, a new variant of bush-equilibration algorithm is proposed, which is origin-based and circumvents line search by decomposing assignment with respect to node (instead of origin). As to be shown by the numerical results, the performance of the new algorithm is consistently better than that of the existing variants of Bar-Gera’s algorithm, and in many cases is comparable to that of Dial’s. The relative performance of the new algorithm to existing ones helps identify the critical factors that affect computational performance. Last but not least, a number of implementations issues, notably the problem of sequence in the presence of multiple origins, are discussed in details.

Four bush-equilibration algorithms, including Bar-Gera’s original algorithm (2002), Dial’s algorithm (2006), improved Bar-Gera’s algorithm with corrected Hessian (Nie, 2007), and the aforementioned new algorithm, are implemented and tested in this paper. Although they are different in many ways, these algorithms are generally of Newton-type. Theoretical results that justify these algorithms are first synthesized and presented in a cohesive manner, which are followed by a comprehensive numerical study using a wide range of networks. The algorithms are implemented atop the same programming platform such that they share codes wherever possible, which ensures meaningful comparison.

For the remainder, the next section reviews the origin-based formulation for the UE traffic assignment problem. Section 3 discusses the construction of acyclic sub-networks. Section 4 presents four bush-based algorithms using either route flow or origin flow based transformations. Several important implementation issues are addressed in Section 5 and Section 6 numerical results. The last section summarizes the paper with remarks on the major findings and possible directions for further research.

2. The origin-based formulation

Bush-based algorithms can be derived by decomposing TAP with respect to either origins or destinations, at which bushes are rooted. Such a decomposed TAP formulation, termed origin-based in this paper, is in fact a variant of Beckmann’s seminal formulation (1956), which, unlike the path-link formulation made popular by Dafermos (1968), relies on the link–node flow conservation law. Beckmann’s formulation may be written using origin flows as follows (Bar-Gera, 2002):

\[
\begin{align*}
\min z(x) &= \sum_{ij} \int_0^{x_{ij}} t_{ij}(w) dw \\
\text{subject to : } \\
\sum_{l \in \partial S} x_{ij} - \sum_{r \in R} x_{ij} &= q^r_{ij}, \quad \forall j \in N, r \in R \\
\sum_{i} x_{ij} &= x_i, x_{ij} \geq 0, \quad \forall (i,j) \in A, r \in R,
\end{align*}
\]

1 To be precise, “Newton-type” here means that second-order derivatives are used to guide the search direction when solving decomposed sub-problems. The overall structure of these algorithms is however a Gauss–Seidel decomposition process whose convergence rate is linear (see e.g. Rheinboldt, 1998).
\[ q^r_j = \begin{cases} \sum d_{rs} & j = r \\ -d_{rs} & j = s \\ 0 & \text{otherwise} \end{cases} \quad \forall j \in A, r \in R, s \in S \] (4)

where

- \( A \) and \( N \) are sets of links and nodes, respectively.
- \( R \) and \( S \) are sets of origin and destination nodes, respectively.
- \( d_{rs} \) is the travel demand departing from origin \( r \) destining to \( s \).
- \( x^r_{ij} \) is the flow on link \( ij \) contributed by all O–D pairs \( rs \) originating at \( r \).
- \( t_{ij} \) is the total flow on link \( ij \).
- \( t_{ij}(\cdot) \) is a strictly positive and increasing function of \( x^r_{ij} \).
- \( O(i) \) and \( I(j) \) denote the set of successor(s) and predecessor(s) of \( j \), respectively.

Problem (1)–(4) can be decomposed with respect to origins, in the spirit of the Gauss–Seidel method. Namely, one can separately consider the traffic assignment problem for each origin \( r \), while treating flows contributed by other origins as constant background traffic. Keeping this in mind, we suppress the origin index \( r \) and concentrate on a single-origin case for notational convenience. Although \( r \) no longer appears in the notation, we shall use it to refer to “the” origin of the problem whenever necessary. The above problem is rewritten as:

\[
\min z(x) = \sum_{j} \int \int_{0}^{q^r_j} t_{ij}(w)dw
\]

subject to:

\[
\sum_{i \in I(j)} x_{ij} - \sum_{l \in O(j)} x_{lj} = q_j, \quad \forall j \in N \tag{6}
\]

\[
x_{ij} \geq 0, \quad \forall (i, j) \in A \tag{7}
\]

where \( q_j \) is traffic input from the origin destined for \( j \). Let \( u_l \) be the multiplier associated with node \( j \) (Constraint (6)). The KKT conditions of the above formulation include the feasible Conditions (6), (7) plus the following complementary conditions:

\[
t_{ij}(x_j) + u_i - u_j \geq 0, \quad \forall (i, j) \in A \tag{8}
\]

\[
x_{ij} t_{ij}(x_j) + u_i - u_j = 0, \quad \forall (i, j) \in A \tag{9}
\]

Interpreting \( u_l \) as the minimum travel time from the origin \( r \) to \( j \), the optimality condition states that link \( ij \) is used only when it is on the shortest route leading to its head node \( j \). For any route between \( r \) and \( j \), summing the above condition over all links that belong to the route yields

\[
c^j_k - u_j \geq 0, f^j_k(c^j_k - u_j) = 0, \quad \forall j \tag{10}
\]

where \( f^j_k \) and \( c^j_k \) are the flow and travel time on route \( k \in K \). This is the well-known route-based user equilibrium (UE) condition (Wardrop, 1952). As suggested by (10), the UE condition actually holds not only for each designated O–D pair (as stated in Wardop’s first principle), but indeed for any node pair \( ij \) where \( j \neq r \). In addition, the UE flows for the single-origin problem satisfy the following important property.

**Proposition 1** (Acyclicity of user equilibrium). For the single-origin formulation (5)–(7), links that have positive flow at user equilibrium never form a direct cycle.\(^2\)

**Proof.** See Lemma 1–3 (Bar-Gera, 2002) or Lemma 3 (Dial, 2006). Also see Newell (1980, pp. 154–155). □

Consequently, origin-based UE flows form an acyclic sub-network. We call such a network *bush*, following Dial (1970, 1971, 1999) for its simplicity. In this paper, a bush is formally defined as below:

**Definition 1** (*Bush*). A directed network is called a bush rooted at \( r \) if it (1) is acyclic, (2) has at least one route from \( r \) to every other node.

It follows from the definition that the root node \( r \) is the only node in the bush that has no incoming link. Acyclicity of UE flows has important implications to the design of algorithms. For one thing, it is much more efficient to find shortest paths over a bush thanks to the acyclicity. Perhaps more importantly, acyclicity makes it equally efficient to find longest paths, which is NP hard for general networks. However, since a real transportation network is by no means a bush, one has to carefully construct one in order to take advantage of acyclicity. This is a non-trivial issue that requires some elaboration.

\(^2\) Whenever cycle is used hereafter, it always refers to direct cycle, unless otherwise specified.
3. Construction of bushes

A connected network $G(N,A)$ can be reduced to a bush $B(N,A_b)$ by removing a subset of its links to eliminate all cycles. A bush $B(N,A_b)$ is called a child bush of $G(N,A)$ if $A_b \subseteq A$. Let the single-origin traffic assignment problem (5)–(7) be defined on the network $G(N,A)$. Then, a restricted master problem (RMP) of the original problem can be defined on the child bush $B(N,A_b)$, by restricting traffic assignment to links in $A_b$. Whenever a RMP is solved, the restricted UE solution $x$ should satisfy constraints (6) and (7) as well as the complementarity conditions

$$t_i(x_i) + u_i - u_j \geq 0, \quad \forall (i,j) \in A_b$$

$$x_i(t_i(x_i) + u_i - u_j) = 0, \quad \forall (i,j) \in A_b$$

$$x_i = 0, \quad \forall (i,j) \in A/A_b$$

Since all links $ij \notin A_b$ must have zero flow, to ensure the restricted UE solution is also the solution to the original problem requires $t_i(x_i) + u_i - u_j \geq 0, \forall ij \notin A_b$. Let us define $P^1_i = \{ij|ij \in A/A_b, t_i(x_i) + u_i - u_j, \text{ where } x \text{ is the restricted UE solution corresponding to } B(N,A_b)\}$. The bush corresponding to the user equilibrium solutions for the original problem, called a UE bush, is characterized as follows:

**Definition 2 (UE bush).** A child bush $B(N,A_b)$ is a UE bush if $P^1_i = \emptyset$.

Because the UE bush is generally unknown a priori, it must be constructed iteratively in the solution process. The basic idea is to start from an initial child bush, and then gradually refine it until the graph converges to the UE bush. The procedure may be described as follows (for the proof of the convergence, refer to Bar-Gera (2002), Dial (2006)):

**Bush-based algorithm (BA)**

**Step 0** Initialize $B$ as a shortest path tree rooted at the origin. Assign all flows to links on the tree.

**Step 1** Expand $B$ by adding new links that have potential to reduce O–D travel times.

**Step 2** Solve RMP of Problem (5)–(7) corresponding to $B$.

**Step 3** Reduce $B$ by removing unused links.

**Step 4** Convergence test.

We focus on Steps 1 and 3 in this section and defer the discussion of Steps 2 and 4 to Sections 4 and 5, respectively. First note that unused links can be easily identified as they should carry zero flows. Thus, after solving RMP, links with zero flows can be deleted as long as in so doing Condition (2) in **Definition 1** is satisfied. Expanding network is a more complex issue since adding links arbitrarily could cause cycles. According to **Definition 2**, any link $ij \in P^1_i$ is a candidate for network expansion because adding such a link provides the potential to reduce travel time from $r$ to $j$. The following result ensures that a bush will maintain its acyclicity after adding these links.

**Proposition 2 (Acyclicity preservation I).** If the RMP corresponding to a child bush $B(N,A_b)$ is optimally solved, the graph with links $A_b \cup P^1_i$ remains to be a bush.

**Proof.** Suppose that a cycle $\{k, k+1, \ldots, k+l, k\}$ forms after the addition. Let $C$ be the set of links in the cycle. For any link $ij \in C \cap A_b$ that has a positive flow, $u_i + t_j = u_j$ from the definition of equilibrium (cf. Eq. (9)). Since $t_j > 0$ by definition, we have $u_i < u_j$. If link $ij \in C \cap A_b$ but carries zero flow, we still have $u_i + t_j = u_j$. To see this, note that only one path should exist between the root node to $j$ which must use link $ij$. (If this is not the case, link $ij$ should have been deleted.) For any newly added links $t_j + u_i < u_j$. To summarize, along the cycle $u_k \leq u_{k+1} \leq \cdots \leq u_l$. Note that the cycle must have at least one newly added link, otherwise $A_b$ is not a bush (because it contains a cycle). Thus we must have $u_k < u_l$. A contradiction. \[\Box\]

Under mild assumptions, links in $P^1_i$ guarantees the strict reduction of objective function (5) in the subsequent iteration (cf. Lemmas 2 and 4, Dial, 2006). Consequently, the construction of UE bush will converge finitely. The finite convergence does not rule out the possibility of reentries of links in the process. However, the strict reduction of objective function from one iteration to the next prevents such unavoidable reentries from developing into endless loops.

**Remark 1.** Dial (2006) suggested expanding bushes using links with negative “reduced costs”. Because the reduced cost defined therein equals $u_i + t_j - u_j$ in our paper, Dial’s approach effectively expands a bush using the set $P^1_i$. Also, instead of explicitly removing zero-flow links, the mirror of a newly added link is deleted.\(^3\)

Strictly speaking, the preservation of acyclicity is valid only if the restricted UE solution is exact. Unfortunately, achieving such a “perfect” UE solution is practically impossible. A highly converged UE solution (with a relative gap very close to zero) may effectively reduce (possibly eliminate) the risk of introducing cycles in the above network expansion strategy. However, even in a very highly converged solution, there could be near zero flows (be tiny they may be) on links that would create cycles. It is therefore hard to predetermine against which precision level RMPs should be solved in order to avoid the

\[^3\] Dial’s algorithm thus requires that every link should have a mirror. However, we note that this requirement is not restrictive and imposed only for providing a tight justification of the algorithm.
violation of acyclicity. Moreover, pursuing a highly converged restricted solution is expensive and may not be a worthwhile effort during the early iterations. Therefore, relying on Proposition 2 to preserve acyclicity may not be the best choice. In presence of an imperfect restricted UE solution, \( P_k^1 \) is rewritten as \( P_k^i \). Additional criteria have to be imposed to guarantee acyclicity in this case. One possibility is to make use of the topological distance in a bush, which is defined as follows.

**Definition 3 (Topological distance).** Let the length of every link in a bush be 1. The topological distance of a node \( j \), denoted as \( \pi_j \), is the maximum distance from the origin to \( j \).

Nodes in a bush can be ordered according to their topological distances. Visiting nodes following this order is called a pass and is the key to efficient operations (such as shortest path search) promised by a bush.

**Definition 4 (Ascending (descending) pass).** An ascending (descending) pass is a sequential visit to each node of a bush following the increasing (decreasing) order of topological distance.

To maintain acyclicity, it is sufficient to require a newly added link \( ij \) to satisfy the following condition

\[
\pi_i < \pi_j
\]

where \( \pi_i \) is the topological distance of node \( i \). To see this, note that the existence of any cycle would contradict the fact that \( \pi_i < \pi_j \) for all links along the cycle. However, since topological distance does not take into account actual travel cost, using Condition (14) to select links seems an arbitrary choice. A better alternative is to use the maximum travel cost from the origin to node \( i \) (denoted as \( U_i \)) to replace the topological distance in (14). Accordingly, we define

\[
P_k^2 = \{ij | \pi_i < \pi_j, U_i + t_{ij} < U_j, \text{where } x \text{ is an inexact restricted UE solution corresponding to } B(N, A_b)\}.
\]

**Proposition 3 (Acyclicity preservation II).** For any bush \( B(N, A_b) \), the graph with links \( A_b \cup P_k^2 \) is always a bush.

**Proof.** First, for any link on the existing bush, \( U_i + t_{ij} < U_j \) by definition (note that if \( U_i + t_{ij} > U_j, U_j \) would not be the maximum cost from the origin to \( j \)). Thus, if a cycle occurs after adding a link in \( P_k^2 \), \( U_i < U_j \) for all links \( ij \) along it. This is clearly impossible. ☐

**Remark 2.** Adding links defined in \( P_k^2 \) promises to reduce the maximum travel cost from the origin to \( j \) in the subsequent iteration, which helps move the solution toward UE because the maximum and minimum costs from the origin to any node should be identical at UE. Bar-Gera (2002) employs a slightly different version of \( P_k^2 \): namely, he proposes adding all links that satisfy \( U_i < U_j \) in network expansion, although a formal justification of the strategy is not offered. Comparing to those included in \( P_k^2 \), Bar-Gera’s approach will add more links, of which some may not necessarily reduce maximum cost.

Recognizing the difficulty of obtaining an exact restricted UE solution, a bush is expanded in this paper using the intersection of \( P_k^1 \) and \( P_k^2 \). This strategy not only offers a descent direction as discussed before, but also maintains acyclicity of the expanded network. In the unlikely event where \( P_k^1 \cap P_k^2 = \emptyset \), links in \( P_k^2 \) will be used to avoid breakdown.

4. Algorithms for the restricted master problem

Having demonstrated how bushes may be iteratively constructed, we now turn to the issue of equilibrating flows on these bushes, namely, solving the restricted master problem. Although many algorithms exist for this problem, the focus of this paper is on Newton-type algorithms that are able to make good use of the property of acyclicity. These include algorithms based on either route or origin flows.

4.1. Route flow transformation

Problem (5)–(7) can be transformed using route flows as follows:

\[
\min z(f) = \sum_{mn} \int_0^{x_{mn}} t_{mn}(w)dw
\]

subject to:

\[
\sum_k f^k_i = \eta_j, \quad \forall j \in N
\]

\[
x_{mn} = x^0_{mn} + \sum_k \delta_{mnk}f^k, \quad \forall k \in K, \quad \forall mn \in A
\]

\[
\eta_j = \sum_{i \in O(j)} x_{ij} + q_j
\]

\[
f^j_k \geq 0, \quad \forall j, k
\]

where \( \eta_j \) is the total flows arriving at node \( j \), \( f^k_i \) is the flow on path \( k \) connecting the origin and \( j \); \( \delta_{mnk} = 1 \) if path \( k \in K \) (\( K \) is a set of routes from the origin to \( j \)) uses link \( mn \) and 0 otherwise; and \( x^0_{mn} \) is the portion of flow on link \( mn \) that does not arrive at \( j \).
The above transformation differs from the well-known link-route formulation (Dafermos, 1968; Sheffi, 1985) in two aspects. First, the flow conservation Condition (16) is imposed at all nodes instead of only destinations. Accordingly, \( \eta_j \) is defined as the sum of the demand for \( j \) (\( q_j \)) and the flow passing through \( j \) to head for downstream nodes. It is worth noting that \( \eta_j \) is well defined only in acyclic networks. Second, the formulation is a decomposition for each node \( j \) which restricts traffic assignment only to links that belong to at least one route \( k \in K_j \). Consequently one can solve the problem for each \( j \) sequentially, following a descending pass.

Let \( k \in K_j \) be the shortest route. The constraint (16) can be rewritten as

\[
f_k^j = \eta_j - \sum_{k \neq k} f_k^j
\]

Replacing \( f_k^j \) in (17) with this new constraint, the route-based formulation becomes

\[
\min \sum_{mn} \int_0^{d_{mn}} \sum_{k \in K_j} \left[ \sum_{t \in k} f_k^j \left( \sum_{t \in k} \eta_j - \sum_{k \neq k} f_k^j \right) \right] t_{mn}(w)dw,
\]

subject to: \( f_k^j \geq 0, \ k \neq k \)

The simple constraint structure of the above problem makes projected Newton methods an ideal choice. In such a method, the flows on non-shortest routes are updated in iteration \( p + 1 \) by:

\[
f_k^j(p + 1) = \left[ f_k^j(p) - \frac{g_k^j}{h_k^j} \right]^+, \ \forall k \in K_j, p \neq k
\]

where \( \lambda \) is a step size and \( [a]^+ = a \) if \( a \geq 0 \) and \( 0 \) otherwise. Furthermore,

\[
g_k = \frac{\partial z(f)}{\partial f_k^j} = \sum_{mn} \delta_{k}^{mn} t_{mn} - \sum_{mn} \delta_{k}^{mn} t_{mn} = c_k^j - c_k^j
\]

\[
h_k = \frac{\partial^2 z(f)}{\partial f_k^j^2} = \sum_{mn} \left( \delta_{k}^{mn} - \delta_{k}^{mn} \right) t_{mn}(\delta_{k}^{mn} - \delta_{k}^{mn})
\]

Note that the above algorithm only employs the diagonal elements of the Hessian, and therefore is a quasi-Newton method.

It is obvious that a route-based algorithm needs to avoid priori route enumeration to be practically useful. In most previous studies, this is achieved by column generation, which iteratively generates routes only when they become required to optimally solve the problem (Gibert, 1968; Bertsekas, 1976; Jayakrishnan et al., 1994). Nonetheless, even storing and manipulating such a subset of routes is expensive for today’s practical networks, which may easily contain millions of O–D pairs. Dial’s algorithm (2006) overcomes this difficulty by only tracking the shortest and longest routes and shifting flows between them using (22). By taking advantage of acyclicity in a bush, this algorithm even obviates the storage of shortest and longest routes. Note that longest and shortest route trees can be calculated simultaneously in a single ascending pass of a bush, and all route relevant information (e.g. \( g_k \) and \( h_k \) in (22)) can be obtained in this pass, without reading any information from storage. A description of Dial’s algorithm now follows.

Dial’s algorithm

**Input:** a feasible link flow vector \( x_{ij} \), parameter \( \epsilon \)

**Step 0** Initialization: update link travel time \( t_{ij} \).

**Step 1** Ascending pass: Calculate the shortest and longest route trees. For each node \( i \), store the shortest and longest travel times from the origin to node \( i \) in \( u_i \) and \( U_i \), and the shortest and longest routes in \( p_i \) and \( P_i \).

**Step 2** Convergence test: If \( \max(U_i - u_i), \ \forall i < \epsilon \), stop; otherwise, goto Step 2.

**Step 3** Descending pass: Repeat the following steps for every node \( j \) before returning to Step 1.

1. Following the descending order from \( j \), find the first node \( i \) such that \( p_i = P_i \). If \( i = j \), go to next \( j \), otherwise, denote the set of links between \( j \) and the \( i \) that are on shortest and longest routes as \( S_j \) and \( L_j \), respectively.
2. Calculate \( g = (u_j - u_i) - (U_j - U_i) \), \( h = \sum_{mn \in S_j \cup L_j} t_{mn}^j \). Set \( dx = \min(\|g\| / h, \min \{x_{mn} | mn \in L_j \}) \).
3. **Update flows.** set

\[
x_{mn} + dx \quad mn \in S_j
\]

\[
x_{mn} - dx \quad mn \in L_j
\]

4. **Update** \( t_{mn}, mn \in S_j \cup L_j \).
The step size \( \lambda \) in Step 3.1 is usually taken as a constant 1.0. As in most Newton-type methods, \( \lambda \) may be reduced when it causes oscillations (Jayakrishnan et al., 1994). Following Step 3.3, it may be useful to update shortest and longest paths for the affected portion of the network. While this operation keeps \( u_i, U_i, P_i \) up-to-date in the subsequent operations, it does incur extra computational overhead. We did not implement this extra operation as it is not explicitly mentioned in Dial (2006).

4.2. Origin flow transformation

Problem (5)–(7) can also be transformed by using routing variables to represent origin flows. This transformation was first studied by Gallager (1977), and revisited in Bar-Gera (2002). Let \( \eta_j \) be the total flow arriving at node \( j \), and the routing variable \( \phi_{ij} \) denote the proportion of \( \eta_j \) that use link \( ij \). The problem is rewritten as

\[
\min z(\phi) = \sum_y \int_{0}^{\eta_y} t_i(w)dw
\]

subject to:

\[
x_y = \eta_y \phi_{ij}, \quad \forall j
\]

\[
\eta_j = q_j + \sum_{l \in \partial(j)} \eta_l \phi_{lj}, \quad \forall j
\]

\[
\sum_{l \in \partial(j)} \phi_{lj} = 1, \quad \forall j; \phi_{ij} \geq 0, \quad \forall ij
\]

(25)–(28)

With the help of acyclicity, the gradient of \( z(\phi) \) with respect to \( \phi \) can be calculated recursively using the following formula (Bar-Gera, 2002):4

\[
\frac{\partial z(\phi)}{\partial \phi_{lm}} = \eta_m \left( t_{lm} + \frac{\partial z(\phi)}{\partial q_l} \right) \cdot \frac{\partial z(\phi)}{\partial q_l} = \sum_i \phi_{il} \left( t_{il} + \frac{\partial z(\phi)}{\partial q_l} \right) \cdot \frac{\partial z(\phi)}{\partial q_l} = 0
\]

(29)

To obtain a sufficient UE condition, define \( v_i = \frac{\partial z(\phi)}{\partial \phi_{lm}} \) and \( v_{lm} = t_{lm} + v_i \).

Theorem 1 (A sufficient UE condition). A feasible solution of the optimization problem (25)–(28) is user-equilibrium if the following condition is satisfied.

\[
v_{lm} - u_{lm} \geq 0; \phi_{lm}(v_{lm} - u_{lm}) = 0, \quad \forall l, m \neq r
\]

(30)

where \( u_{lm} = \min\{v_{lm}, \forall l \in I(m)\} \).

Proof. cf. Theorem 3 (Gallager, 1977), also Lemma 5 (Bar-Gera, 2002). □

To evaluate \( v_{lm} \) and \( v_i \) requires an ascending pass of the bush, using a slightly different recursive relationship (Bar-Gera, 2002):

\[
v_{lm} = t_{lm} + v_i; \quad v_i = \sum_{l \in I(i)} \phi_{il} v_{il}; \quad v_r = 0
\]

(31)

It is generally difficult to derive an exact closed-form for the second-order derivatives required by Newton-type algorithms. The upper and lower bounds, however, are available for these derivatives.

Theorem 2. In the optimization problem (25)–(28), the second-order derivative of \( z(\phi) \) with respect to \( \phi \) satisfies the following conditions:

\[
\frac{\partial^2 z(\phi)}{\partial \phi_{lm}^2} = \eta_m \left( t_{lm} + \frac{\partial^2 z(\phi)}{\partial^2 q_l} \right)
\]

(32)

\[
\sum_{l \in I(i)} \phi_{il} \left( t_{il} + \frac{\partial^2 z(\phi)}{\partial^2 q_l} \right) \leq \frac{\partial^2 z(\phi)}{\partial^2 q_l} \leq \sum_{l \in I(i)} \phi_{il}^2 t_{il} + \left( \sum_{l \in I(i)} \phi_{il} \sqrt{\frac{\partial^2 z(\phi)}{\partial^2 q_l}} \right)^2
\]

(33)

\[
\frac{\partial^2 z(\phi)}{\partial^2 q_r} = 0
\]

(34)


Note that \( q_i \) is treated as an auxiliary free variable in order to establish the recursive relationship. For details the reader is refer to Gallager (1977) and Nie (2007).

---

4 Note that \( q_i \) is treated as an auxiliary free variable in order to establish the recursive relationship. For details the reader is refer to Gallager (1977) and Nie (2007).
To be consistent with the sufficient condition given in Theorem 1, define \( s_l = \frac{\partial^2 z(\eta)}{\partial \eta_l^2}, \quad s_lm = \frac{\partial^2 z(\eta)}{\partial \eta_m^2} \), and \( s_l = t'_l + s_l \). Using the lower bound defined in (33), \( s_{lm} \) can be calculated using
\[
s_{lm} = s_{lm} = t'_l + \tilde{s}_l; \quad \tilde{s}_l = \sum_i \phi_i \tilde{s}_i; \quad \tilde{s}_l = 0. \tag{35}
\]

If the upper bound is applied, the recursive formula reads:
\[
s_{lm} = s_{lm} = t'_l + \tilde{s}_l; \quad \tilde{s}_l = \sum_i \phi_i \tilde{s}_i + \sum_j \phi_{ij} \sqrt{\tilde{s}_j}; \quad \tilde{s}_l = 0 \tag{36}
\]

The reader is referred to Nie (2007) for an elaborated derivation of Eqs. (25)–(36). Before discussing Bar-Gera’s algorithm (Bar-Gera, 2002), we need to first define the last pseudo origin (LPO).5

**Definition 5. (Last pseudo origin)**

For any given node \( j \) in a bush, a node \( i \) is a pseudo origin of \( j \) if and only if all routes leading to \( j \) come through node \( i \). The last pseudo origin (LPO) is the pseudo origin that has the minimum topological difference \( \pi_j - \pi_i \).

**Bar-Gera’s algorithm**

**Input:** a feasible link flow vector \( x_{ij}, \) parameters \( \epsilon, \epsilon_1, \beta \)

**Step 0** Initialization: update link travel time \( t_{ij} \) and its derivative \( t'_{ij} \) based on the current \( x_{ij} \).

**Step 1** Ascending pass: Calculate \( u_i, U_i \) if \( \max(U_i - u_i) < \epsilon \), stop; otherwise, calculate \( \tilde{v}_i, \tilde{v}_y \) using (31), and \( \tilde{s}_i, \tilde{s}_y \) using (35).

Set step size \( \lambda = 1.0 \), auxiliary link flows \( y_{ij} = 0 \) for all \( ij \) on the current bush, and auxiliary node flows \( \eta_{il} = 0 \) for all node \( i \).

**Step 2** Descending pass: Repeat the following for every node \( j \)

1. **Find the last pseudo origin** \( l \) of \( j \) and \( m = \arg\min_i (t_{ij} | i \in I(j)) \), Update \( y_{ij} = y_{ij} + \eta_{ij} \phi_{ij} \) for all \( i \in I(j) \).

2. **For each link** \( ij \), \( i \neq m \): calculate
\[
\sigma = \frac{dx}{dz} = \sum_{ij} y_{ij} t_{ij} (x_{ij} + y_{ij})
\]

if \( \sigma < 0, \) set \( x_{ij} = x_{ij} + \frac{y_{ij}}{\sigma_i}, \forall ij \), go to Step 1; otherwise, set \( \lambda = \beta \lambda \), go to Step 2.

In Step 2.1, a quasi-Newton method is employed to shift flows from links associated with non-shortest routing variables to the one corresponding to the shortest routing variable. Note that \( g \) and \( h \) represent, respectively, the first- and second-order derivatives of \( z(\phi) \) with respect to non-shortest routing variables (where the shortest routing variable \( \phi_{lm} \) is replaced with \( 1 - \sum_j = \epsilon \phi_{ij} \)) using a similar concept as in Eq. (21)). Also, \( h \) is an approximation of the Hessian’s diagonal elements because 1) Eq. (35) provides only a lower bound for \( \lambda_s \) and \( s_{lm} \), and 2) the overlapping between \( \phi_{ij} \) and \( \phi_{lm} \) may not be fully captured by (37). Step 3 checks whether or not the proposed move \( y_{ij} \) would strictly reduce the objective function value (Lemma 7, Bar-Gera, 2002). Such a reduction is guaranteed as long as the step size is small enough (Lemma 8, Bar-Gera, 2002). Finally, the value of \( \beta \) recommended by Bar-Gera is 0.5.

Eq. (37) bounds \( h \) from below using a small positive number. This bound is necessary to avoid numerical breakdown because \( t'_{ij} \) (hence \( h \)) could become zero when \( x_{ij} = 0 \). However, Nie (2007) showed that, as a result of using lower bound approximation (35), \( h \) given by (37) might be negative even when all \( t'_{ij} > 0 \). Consequently, the original Bar-Gera’s algorithm ends up using \( \epsilon_1 \) to estimate second-order derivatives more frequently than intended, which often negatively impact the overall convergence performance. Nie (2007) also showed that the upper bound approximation (36) ensures \( s_{ij} + s_{mj} - 2 \tilde{s}_l > 0 \) under mild assumptions. A modified Bar-Gera’s algorithm, in which the calculation of \( \tilde{s}_i, \tilde{s}_y \) in the ascending pass is replaced with (36), is implemented in Nie (2007) and demonstrates improved convergence performance.

Bar-Gera’s algorithm employs an Armijo-type line search (Armijo, 1966) to determine a step size. This extra effort is deemed as necessary in order to guarantee a negative social pressure \( \sigma \) (see Step 3, Bar-Gera’s algorithm), and in turn the strict reduction of objective function value. However, it is worth investigating whether or not the effort of line search is paid off from an efficiency point of point, for several reasons. First, the requirement of negative social pressure is sufficient yet not necessary for strict descent. Therefore, forcing it through line search may be overly conservative. Secondly, when flow update is restricted only to the portion of the bush between a node and its LPO and therefore can be preformed sequentially and

5 It is is called last common node in Bar-Gera (2002).
more frequently, exact line search is often regarded unnecessarily expensive (e.g., Dial’s algorithm). Finally, line search in Bar-Gera’s algorithm is particularly time-consuming because whenever a step size evaluation is performed for an origin, a descending pass is required. These observations suggest that better performance may be achieved by forsaking line search in Bar-Gera’s algorithm and fit it into a similar framework as Dial’s (2006). We describe such a revised procedure in the following.

A revised Bar-Gera’s algorithm that circumvents line search.

**Inputs:** a feasible link flow vector \( x_{ij} \), parameters \( \epsilon \)

**Step 0** Initialization: Update link travel time \( t_{ij} \) and its derivative \( t_{ij}' \) based on the current \( x_{ij} \).

**Step 1** Ascending pass: Calculate \( u_i, U_i \), if \( \max(U_i - u_i) < \epsilon \), stop; otherwise, calculate \( v_i, v_j \) using (31), and \( s_i, s_j \) using (35).

**Step 2** Descending pass: Repeat the following for every node \( j \)

1. Find the last pseudo origin \( l \) of \( j \) and \( m = \text{argmin}_i (v_{ij} | i \in I(j)) \).
2. For each link \( ij \in I(j) \) and \( i \neq m \): calculate \( g \) and \( h \) using (37). Set \( dx = \min(\lambda g/h, x_{ij}) \) (where \( \lambda \) is a step size similar to Dial’s algorithm), update \( x_{ij} = x_{ij} - dx, x_{mj} = x_{mj} + dx, \eta_i = \eta_i - dx, \eta_l = \eta_l + dx \).
3. Following a descending pass, for every node \( p \) between nodes \( j \) and \( l \) (not including \( j \) and \( l \)), set \( x_{ip} = x_{ip} + g_{ij} \phi_{ip}, \eta_i = \eta_i + h_{ip} \phi_{ip}, \forall i \in I(p) \).
4. Update travel time for all links between \( i \) and \( l \), and \( j \) (not including \( j \) and \( l \)), update \( v_i, v_j \) using (31), and \( s_i, s_j \) using (35).

The revised algorithm circumvents the line search, and update link flows \( x_{ij} \) concurrently in the descending pass using a quasi-Newton method. In each update, the algorithm passes the portion between the current node and its last pseudo origin twice, one in the descending order to shift flows, the other in the ascending order to update the gradient and second-order derivatives.

5. Implementation issues

We first discuss how the solution of restricted master problem is coordinated in the multi-origin case. Fig. 1 illustrates two possible approaches. In the first, each single-origin RMP is equilibrated (to a desirable precision) before moving to the next origin. After all origins have been visited, the overall convergence criterion is checked and a new iteration starts if needed. The second approach, which was used in Bar-Gera (2002), scans the origin list while performing one RMP iteration per origin. Essentially, it attempts to bring all RMPs to equilibrium simultaneously. According to our experiments, this all-at-once approach provides better performance in general, and thus is adopted in our implementation.

In the bush expansion step (see Step 2 in procedure BA), Bar-Gera (2002) suggests updating link flows (by performing a RMP iteration) immediately following the expansion of each bush. The impact of this strategy on the construction of UE bushes is unclear. On the one hand, flow update could violate UE conditions on bushes that have not yet been inspected for expansion (essentially it “breaks up” the equilibrium state inherited from the previous iteration), and consequently make it difficult to identify right links to add. On the other hand, the immediate flow update allows subsequent bush expansion to take into consideration the latest change of network topology, which may expedite convergence. Our experiments suggest that Bar-Gera’s strategy generally affect convergence performance positively, although the actual effect is largely problem-specific. Accordingly, our implementation updates link flows and costs at once after a bush is expanded.

We measure algorithm convergence using relative gap, which is defined in this paper as

\[
g_r = 1 - \frac{\sum_{r \neq l} u_{rs} d_{rs}}{\sum_{r \neq l} t_{rs}} \tag{38}\]

where \( u_{rs} \) is the minimum travel time between O–D pair \( rs \) based on the current link travel time \( t_{ij} \), \( \forall ij \). In our implementation, the relative gap controls the overall convergence in Step 4 of the procedure BA. However, using the relative gap to measure the convergence of RMPs seems unnecessarily expensive. Instead, an approximate convergence indicator is employed, which measures the maximum difference between longest and shortest travel times at any node on the bush. In the presence of multiple origins, the maximum difference can be averaged over all origins, weighted by total demands from each origin.

Finally, four algorithms are implemented and tested. These algorithms all adopt the BA framework, and differ from each other only on how RMP is solved (Step 3). One motivation of the paper is to demonstrate the difference of these algorithms, which include: (1) Dial’s algorithm (DBA) (Dial, 2006), (2) Bar-Gera’s algorithm (BBA) (Bar-Gera, 2002), (3) Revised Bar-Gera’s algorithm using upper bound (NBA) (Nie, 2007), and (4) Revised Bar-Gera’s algorithm that circumvents line search (QBA). The above algorithms are implemented using C++ on TNM (Toolkit of Network Modeling), a suite of reusable C++ classes for general network applications (Nie, 2006). While compromising some efficiency, this uniform object-oriented programming framework ensures that the algorithms share codes wherever possible. Recognizing that computation speed
is affected by many hardware and software factors, the goal of this study is not to develop the “fastest” code. Rather, we expect that the study will generate useful insights about the relative performance of these algorithms which is independent of specific software and hardware environments.

### 6. Numerical results

This section is organized into three parts. The first part intends to set the performance benchmark using several small real networks which have been tested in previous studies. The second part aims to demonstrate the sensitivity of algorithm performance to network size, density and demand level, using randomly generated grid networks. The last part tests the algorithms on larger real networks.

All numerical results reported in this section were produced on a Windows XP x64 Workstation with two Xeno 3.0 GHz CPUs and 8 GB RAM. Unless otherwise stated, the convergence criterion is that the relative gap $g_r \leq 10^{-12}$. The maximum numbers of main and RMP iterations are 500 and 20, respectively. In algorithms DBA and QBA, the step size $\lambda$ is reduced by a factor of 0.8 whenever the relative gap obtained from the current main iteration is 10% higher than that from the last iteration. Finally, the BPR-function is used to model the link travel time function $t_{ij}(x)$, where $t_{ij}(x) = t_0(1 + 0.15(x/C)^{4})$, where $t_0$ is the free flow travel time and $C$ is the link capacity.

#### 6.1. Performance on benchmark problems

The details of the tested networks are reported in Tables 1 and 2. In this section we focus only on the four small-scale problems. The convergence performance of the four bush-based algorithms, as well as the classic Frank–Wolfe (FW) algorithm, are compared in Fig. 2.

The FW algorithm demonstrated its well-known inability to achieve highly precise equilibrium solution, in that it never attained a relative gap of $10^{-6}$ in these experiments, while all its bush-based counterparts easily accomplished such a goal. Nevertheless, in all of the four problems FW obtained relative gaps of $10^{-3}$–$10^{-4}$ with the amount of CPU time comparable to those of the best bush-based algorithms, suggesting that FW could be competitive if solutions of low precision are to be sought.

**Fig. 2** shows that in all cases DBA and QBA attained the same level of precision with roughly 1/3 of CPU time required by BBA and NBA. It seems that the extra CPU times consumed by BBA and NBA largely came from their line search procedure. As revealed in Fig. 3, the CPU time per iteration (which consists of times required for solving RMP and constructing bush) is about three times higher in BBA and NBA. In addition, DBA and QBA also required fewer main iterations to achieve the same relative gap. Apparently, more frequent flow update helps sharpen convergence. Note that DBA and QBA update flows once for each node visited in the backward pass. In contrast, BBA and NBA do not update flow until they have finished visiting all nodes and confirmed that the proposed move yields a negative social pressure.

---

6 Parallel techniques such as multi-threading, for instance, may be used to take advantage of Dual- or Quad-core CPUs now available on inexpensive PCs.
Table 1
Detail of test networks.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Network</th>
<th>Node</th>
<th>Link</th>
<th>Zone</th>
<th>Trip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Anaheim</td>
<td>416</td>
<td>914</td>
<td>38</td>
<td>104,694</td>
</tr>
<tr>
<td></td>
<td>Barcelona</td>
<td>1020</td>
<td>2522</td>
<td>97</td>
<td>184,680</td>
</tr>
<tr>
<td></td>
<td>Winnipeg</td>
<td>1052</td>
<td>2836</td>
<td>135</td>
<td>64,784</td>
</tr>
<tr>
<td></td>
<td>Chicago</td>
<td>933</td>
<td>2950</td>
<td>386</td>
<td>1,260,910</td>
</tr>
<tr>
<td>Large</td>
<td>Seattle</td>
<td>6189</td>
<td>13,933</td>
<td>956</td>
<td>1,324,610</td>
</tr>
<tr>
<td></td>
<td>ChicagoRegional</td>
<td>12,982</td>
<td>39,018</td>
<td>1771</td>
<td>1,360,430</td>
</tr>
</tbody>
</table>

Source: Seattle network is converted from a VISUM version file provided by Robert Shull at PTV America. All other network data are obtained from http://www.bgu.ac.il/bargera/tntp/.

Table 2
Details of random grid networks.

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Node</th>
<th>Link</th>
<th>Zone</th>
<th>Trip</th>
<th>Set 2</th>
<th>Node</th>
<th>Link</th>
<th>Zone</th>
<th>Trip</th>
<th>Set 3</th>
<th>Node</th>
<th>Link</th>
<th>Zone</th>
<th>Trip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>360</td>
<td>12</td>
<td>47,621</td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>191,400</td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>114,840</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>1520</td>
<td>50</td>
<td>92,890</td>
<td>1600</td>
<td>9440</td>
<td>200</td>
<td>293,336</td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>153,120</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>900</td>
<td>3480</td>
<td>112</td>
<td>144,288</td>
<td>1600</td>
<td>12,640</td>
<td>200</td>
<td>387,933</td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>191,400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>191,400</td>
<td>1600</td>
<td>15,832</td>
<td>200</td>
<td>479,411</td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>229,680</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2500</td>
<td>9800</td>
<td>312</td>
<td>235,374</td>
<td>1600</td>
<td>19,008</td>
<td>200</td>
<td>588,400</td>
<td>1600</td>
<td>6240</td>
<td>200</td>
<td>267,960</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. Convergence performance on small real networks.
The performance of QBA and DBA is very similar in the test problems. However, DBA’s expense per iteration is slightly lower (see Fig. 3), because DBA restricts each flow update only to shortest and longest paths while QBA may have to update flows on the entire portion between the current node and its LPO. On the other hand, NBA consistently outperforms BBA, although the two algorithms generally demonstrated similar convergence behavior in this experiment. As noted before, NBA is expected to improve convergence performance because it avoids negative second-order derivatives by using the upper bound approximation.

6.2. Sensitivity analysis

The sensitivity analysis is conducted with three sets of randomly generated grid networks, as shown in Table 3. Set 1 includes five grid networks whose size increases from 10 by 10 to 50 by 50, with a zone ratio of one eighth. All five networks in Set 2 are 40 by 40 grid, with the link-to-node ratio increased from 3.9 to 11.88 (by adding extra links into the grid networks). Note that in both sets, trips are set such that the congestion level within each set is comparable. Set 3 also uses the 40 by 40 grid network from Set 1, but multiplies its O–D demands by a factor of 0.6, 0.8, 1.0, 1.2 and 1.4.

6.2.1. Network size

Table 3 compares CPU times (as well as the number of iterations) required by each algorithm in order to achieve various levels of precision. As shown in the table, DBA is the front runner in almost all cases, not only outperforming BBA and NBA, but also leading its closest competitor QBA with significant margins (in some cases DBA is more than two times faster than QBA). A close look reveals that DBA outperforms QBA mainly because its cost per iteration is significantly lower than QBA when solving these grid networks. In fact, DBA and QBA required roughly the same number of iterations to achieve almost all levels of precision. In the 2500-node problem, DBA took 42 iterations, or 197.28 s CPU time, to achieve the relative gap of $10^{-12}$, while QBA spent 40 iterations but 361.83 s CPU time to achieve the same gap.

![Fig. 3. Iteration vs. CPU time for the four algorithms.](image-url)
The table also shows that BBA is always the slowest algorithm and the only one that had failed to achieve the desirable gap \(10^{-12}/C_0\) for 1600- and 2500-node networks. This problematic performance can be explained by the wrong search direction resulted from lower bound approximation (Nie, 2007). NBA seems to be less sensitive to network size than DBA and QBA. For the 900-node network, the CPU time required by NBA to achieve a relative gap of \(10^{-12}/C_0\) is about 6.6 and 2.3 times

Table 3

| Node | Algorithm | \(10^{-2}\) CPU time in seconds (number of iterations)* | \(10^{-4}\) | \(10^{-6}\) | \(10^{-8}\) | \(10^{-10}\) | \(10^{-12}\) |
|------|-----------|--------------------------------|olvers|olvers|olvers|olvers|olvers|
| 100  | DBA       | 0.02(1) 0.03(4) 0.03(5) 0.05(7) 0.06(10) 0.06(12) | | | | | |
|      | NBA       | 0.03(1) 0.06(3) 0.09(5) 0.19(10) 0.28(16) 0.41(64) | | | | | |
|      | BBA       | 0.01(1) 0.06(3) 0.09(5) 0.19(10) 0.30(16) 0.44(62) | | | | | |
|      | QBA       | 0.01(1) 0.03(3) 0.05(3) 0.06(7) 0.06(9) 0.08(11) | | | | | |
| 400  | DBA       | 0.09(1) 0.42(5) 0.50(6) 0.66(8) 0.80(10) 0.88(11) | | | | | |
|      | NBA       | 0.31(1) 1.13(4) 1.66(6) 2.45(9) 3.22(12) 3.98(27) | | | | | |
|      | BBA       | 0.36(1) 1.58(5) 2.22(7) 11.53(35) 37.17(113) 56.94(402) | | | | | |
|      | QBA       | 0.17(1) 0.58(5) 0.80(7) 1.02(9) 1.22(11) 1.42(13) | | | | | |
| 900  | DBA       | 0.66(1) 2.75(5) 4.86(9) 6.44(12) 8.52(16) 10.59(20) | | | | | |
|      | NBA       | 3.39(2) 7.94(5) 14.14(9) 24.97(16) 41.81(27) 63.58(66) | | | | | |
|      | BBA       | 3.94(2) 9.11(5) 17.89(10) 30.36(17) 49.92(28) 74.19(70) | | | | | |
|      | QBA       | 1.30(1) 4.06(5) 7.14(9) 11.19(14) 18.70(23) 26.23(32) | | | | | |
| 1600 | DBA       | 3.83(2) 9.03(5) 17.98(10) 27.08(15) 43.55(24) 60.03(33) | | | | | |
|      | NBA       | 11.05(2) 26.03(5) 46.36(9) 103.00(20) 165.06(32) 237.4(88) | | | | | |
|      | BBA       | 12.69(2) 30.38(5) 62.72(10) 259.09(38) 934.24(131) -1.00(-1) | | | | | |
|      | QBA       | 8.00(2) 15.00(5) 25.41(9) 51.52(17) 79.27(25) 110.55(34) | | | | | |
| 2500 | DBA       | 10.02(2) 28.00(6) 50.88(11) 83.84(18) 140.68(30) 197.28(42) | | | | | |
|      | NBA       | 28.33(2) 79.88(6) 145.67(11) 252.67(19) 386.20(29) 537.61(48) | | | | | |
|      | BBA       | 32.50(2) 92.50(6) 174.06(11) 983.01(54) -1.00(-1) -1.00(-1) | | | | | |
|      | QBA       | 22.97(2) 47.72(6) 107.17(14) 154.38(19) 262.80(30) 361.83(40) | | | | | |

*The CPU time and number of iterations are both set to -1 when the required relative gap could not be achieved by the algorithm.

The table also shows that BBA is always the slowest algorithm and the only one that had failed to achieve the desirable gap \(10^{-12}\) for 1600- and 2500-node networks. This problematic performance can be explained by the wrong search direction resulted from lower bound approximation (Nie, 2007). NBA seems to be less sensitive to network size than DBA and QBA. For the 900-node network, the CPU time required by NBA to achieve a relative gap of \(10^{-12}\) is about 6.6 and 2.3 times

Note: negative CPU time means the algorithm had never achieved the required gap.

Fig. 4. Results for Set 2 (1600-node network): the sensitivity to link–node ratio.
of those required by DBA and QBA, respectively. These ratios become 2.7 (for DBA) and 1.5 (for QBA) for the 2500-node network.

In theory, the computational time per iteration (unit CPU time) of bush-based algorithms is bounded by $O(mo)$, where $m$ is the number of links and $o$ is the number of origins. The results shown in Table 3 reasonably agree with the prediction of this theoretical bound. For instance, the actual unit CPU times of DBA read from the table are 0.006, 0.08, 0.54, 1.8 and 4.6 s for 100-, 400-, 900-, 1600- and 2500-node networks, respectively. Using the above bound, one can predict that the unit CPU time required for the 1600-node network should be about $3\cdot \frac{112}{3480} \approx 3.3$ times of that required for the 900-node network, which is very close to the actual ratio $1.8/0.54 \approx 3.3$.

### 6.2.2. Link–node ratio

Fig. 4 shows that the computational time of each algorithm generally increases with the link–node ratio. A few exceptions, however, are noticed from the bottom panel of the figure, where most algorithms spent less CPU time (and actually less iterations) to equilibrate the 12,640-link network than they did for other networks, including those with fewer links. Conceivably, network topology could have a complicated impact on convergence behavior which may explain the above abnormality. DBA is the fastest algorithm in all five tests, followed by QBA and NBA. Noticeably, QBA was affected by the link–node ratio more than others; its required CPU times for both precision levels grew with the number of links at the fastest pace. This result is not a surprise when one recalls that QBA may need to scan the entire portion between the present node (in the backward pass) and its LPO for flow update. Thus, the denser the network becomes, the more computation time each flow update demands.

### 6.2.3. Congestion level

Fig. 5 demonstrates the sensitivity of the algorithms to different levels of congestion. As expected, when the network becomes more congested with additional trips, the computational efforts required to achieve the same level of precision increase across all algorithms. However, the degree to which the algorithms are affected by congestion level varies. QBA is probably the most sensitive algorithm, whose CPU time could jump as much as six times with a mere 20% of increase of trips (see the bottom panel in Fig. 5). The relative performance of QBA thus degrades for more congested networks. In particular, when high precision is sought, QBA needs three times more CPU time than DBA does: this is comparable to the performance of NBA.
6.3. Large real networks

This section tests the algorithms on large networks that have been constructed for the purpose of urban travel forecasting. The details of Seattle and Chicago (regional) networks can be found in Table 1. BBA is excluded from the comparison in this section because of its problematic convergence demonstrated in previous tests. The required convergence criterion is reset to $10^{-10}$.

Test results for Seattle network are reported in Fig. 6. Fig. 6a shows that DBA is the fastest algorithm to achieve almost all levels of precision. To converge to $10^{-10}$, DBA took about 1 h, whereas QBA and NBA spent roughly 2 and 3 h, respectively. Fig. 6b reveals that the three algorithms need roughly the same number of iterations to converge, and that it is the unit CPU time per iteration that has made a difference in the total computational cost. This observation is consistent with previous results. Another noteworthy phenomenon from Fig. 6a is the sharp and frequent oscillations on QBA's convergence curve. Corresponding to the oscillations, Fig. 6 shows that the step size $\lambda$ used in QBA was gradually reduced from 1.0 to 0.05 (a pre-determined lower bound). Although in theory any quasi-Newton algorithm without exact line search may be subject to such oscillations, the problem is much more severe in QBA than in DBA. For one thing, the Hessian employed in QBA ignores off-diagonal elements, which does not affect DBA thanks to its restriction of assignment to only two routes. Moreover, even the diagonal elements in QBA's Hessian are obtained through various approximations. Consequently, a smaller step size must be used to avoid oscillations when the approximated Newton direction deviates substantially from the true one.

The results for Chicago regional network (Fig. 7) tell basically the same story: DBA is the front runner, followed by QBA and NBA. In this case, DBA spent about 1.5, 2.5 and 8 h to reach the relative gap of $10^{-6}$, $10^{-8}$, and $10^{-10}$, respectively. This performance is comparable to the results reported in (Dial, 2006), suggesting that our implementation is reasonably efficient. Both QBA and NBA did not attain the required relative gap $10^{-10}$ within 12 h, largely because they completed fewer number of

---

7 Dial reported that his implementation attained $10^{-6}$ and $10^{-8}$ on a slightly different version of Chicago regional network within 150 and 400 min, respectively. We note that, however, Dial's tests were performed on a PC with 3.0 GHz Pentium CPU, which may be inferior to the machine used in this study.
iterations with the same amount of time (Fig. 7b), QBA was subject to similar yet less severe oscillations, which has reduced the step size to about 0.2 (Fig. 7c).

7. Concluding remarks

This paper examines a class of bush-based algorithms (BA) which promise to produce highly precise solutions for the traffic assignment problem. In order to exploit acyclicity of user equilibrium (UE) origin-based flows, these algorithms decompose the solution process into two steps: constructing bushes and finding UE flows on the bushes. The paper proposed new strategies for bush expansion which overcome the shortcomings of those employed in the previous studies. Within a uniform algorithmic framework, four quasi-Newton methods are discussed in details, which can be broadly categorized as route flow based and origin flow based. The former category contains only Dial’s algorithm (DBA), and the latter includes three variants of Bar-Gera’s algorithm, namely the original version (BBA), the improved version with corrected Hessian (NBA), and a newly proposed variant that obviates line search (QBA).

Our numerical results suggest that DBA is consistently faster and more stable than the three variants of Bar-Gera’s algorithm. The success of DBA may be attributed to three reasons.

1. DBA decomposes assignment for each node and thereby update link flows and travel times more frequently. As a comparison, BBA and NBA aggregates assign flows for each origin and rely on line search to ensure strict descent. According to our experiments, frequent flow update not only is a key to sharp convergence, but also makes line search an unnecessarily expensive operation.

2. The underlying assignment mechanism of Dial’s algorithm involves only links on either longest or shortest routes, which is computationally inexpensive. The origin flow based counterparts such as QBA need to visit all links downstream of the last pseudo origin, which is potentially a much larger set than the union of longest and shortest routes.

3. The Hessian employed in DBA is not subject to approximation errors. According to Eqs. (22)–(24), the Hessian of the route flow based problem is a one by one matrix if only two routes are considered. Consequently, there is no off-diagonal element involved in the Hessian. In contrast, obtaining the Hessian for the origin flow based transformation is remarkably more difficult. First, the diagonal elements of the Hessian can only be approximated by lower or upper bounds. The quality of such approximation, unfortunately, appears to be problem-specific. Moreover, off-diagonal elements have to be ignored because it is difficult to get even a closed-form bound for their calculation.

Inspired essentially by the first item listed above, the newly proposed algorithm QBA gained some limited success. On the one hand, it effectively sharpens the convergence performance and reduces the amount time required to complete each single iteration. In fact, QBA consistently outperformed BBA and NBA, and in some cases was as good as DBA. On the other hand, the performance of QBA degrades as the network becomes denser and/or more congested. Generally, as revealed in the experiments, QBA is more sensitive to problem properties than others. Probably the more aggressive flow update strategy adopted in QBA exacerbated the errors in estimating the Hessian. Meanwhile, the abandon of line search make it harder to mitigate the oscillations that come with the estimation errors. It is possible that through more careful implementation the above problems can be alleviated to some degree. However, the approximation error implied by item 3 is an inherited shortcoming, which is hard to completely overcome unless better closed-form bounds are developed.

Finally, the performance of BA can be easily distorted by a number of implementation factors in addition to the RMP solver explored here. Some of these effects are hard to quantify or predict. Updating origin-based flows (i.e., solving the RMP for one iteration) immediately following each bush expansion is one example. Contrary to the initial expectation that this operation is not essential, our experiments indicated that ignoring this step may adversely affect the overall convergence, and that the actual impacts are problem- and algorithm-specific. Another factor that has drastic impacts on convergence is network expansion strategies. To develop “most efficient” code for solving the traffic assignment problem calls for a careful examination of these issues, which may be of interest to further investigation.

Acknowledgements

Dr. Robert Dial, in a comment on an earlier draft of the paper, convinced me that his and Bar-Gera’s algorithms are better described based on the graph object in which they operate on, namely bush. Indeed, I came to realize that the term originally employed, origin-based, does not reflect the fact that Dial’s algorithm in effect operates route flows instead origin-based flows. I am very grateful to Dr. Dial’s many other comments that have substantially improved the quality of the manuscript. Dr. Hillel Bar-Gera has offered generous assistance to improve the author’s understanding on the origin-based algorithm initially proposed in his Ph.D. thesis. His OBA source code, which he kindly shared in 2000 through Dr. Der-Horng Lee, was of enormous help to my own implementation. I owe many thanks to Professor David Boyce for his continual support and guidance in this endeavor. Thanks also go to Professor Michael Florian from University of Montreal, who suggested adding the footnote 1 and the reference cited therein. Last but not least, constructive comments and generous editorial assistance offered by two anonymous reviewers are greatly appreciated. The views expressed in this paper are the author’s alone.
References


