ORIGIN-BASED ALGORITHMS FOR TRAFFIC ASSIGNMENT: ALGORITHMIC STRUCTURE, COMPLEXITY ANALYSIS, AND CONVERGENCE PERFORMANCE

Jun Xie, Ph.D.
Postdoctoral Researcher
School of Naval Architecture, Ocean and Civil Engineering
Shanghai Jiaotong University
800 Dongchuan Rd.
Shanghai 200240, China
Phone: +86 (21) 3420-6196
Fax: +86 (21) 3420-6197
Email: j-xie@sjtu.edu.cn

Chi Xie, Ph.D.
Professor
School of Naval Architecture, Ocean and Civil Engineering
Shanghai Jiaotong University
800 Dongchuan Rd.
Shanghai 200240, China
Phone: +86 (21) 3420-8385
Fax: +86 (21) 3420-6197
Email: chi.xie@sjtu.edu.cn.

Text: 5,970 words
Figures and tables: 6
Total length: 7,470 words

Submitted for presentation at the 94th Transportation Research Board Annual Meeting, January 11-15, 2015, Washington, D.C. and publication in the Transportation Research Record
ABSTRACT

This paper presents an extensive analytical and numerical investigation on a class of origin-based algorithms for the user equilibrium-based traffic assignment problem. A total of nine known algorithms in this class are first clustered into four algorithmic structures, based on their structural differences and similarities in algorithm design. We further conduct a complexity analysis on these different algorithmic structures by calculating the frequency of executing node and link operations, which provides a simple analytical way to estimate their per-iteration computational costs. To deliver a comprehensive and fair comparison on their convergence performance, all these nine algorithms are then implemented on the same programming platform and run to solve a few representative large-scale traffic networks with different sizes and congestion levels. A close look on the convergence performance statistics further justifies the consistency of the complexity analysis and numerical evaluation results on the computational efficiency of these algorithms. Discussions on the degeneration of algorithm convergence efficiency with respect to network size and congestion level provide useful insights for the potential improvement of current origin-based algorithms or the proposition of new algorithms.

Keywords: traffic assignment; user equilibrium; origin-based algorithms; bushes; paired alternative segments; complexity analysis; convergence performance
INTRODUCTION

Traffic assignment has been widely used as the standard tool to predict network flow patterns in the final stage of travel demand forecasting. Although various traffic assignment models that can capture more realistic features of real traffic systems have been explored and studied, such as dynamic traffic assignment (1, 2) and stochastic traffic assignment models (3, 4), we see that the most frequently used traffic assignment model in practice is still the static deterministic model under the user equilibrium (UE) principle, which states that for every origin-destination (O-D) pair all used routes have equal travel costs and no unused route has a lower cost (5). Several superiorities of the UE model is easy to enumerate, including its intuitive assumptions, simple formulations and stable solutions with respect to inputs. To our knowledge, the most important advantage of the UE model in comparison to other models may attribute to its various kinds of solution methods that can solve large-scale network problems efficiently.

The first class of practical TAP algorithms operates in the space of link flows, known as link-based algorithms, such as the Frank-Wolfe (FW) algorithm and its variants (6-8). Easy implementation and fast convergence in early iterations allow these algorithms to emerge as the winner in the transportation planning practice over the past decades. However, the main shortcoming of the link-based algorithms is their inability to obtain high-precision solutions in most practical applications. To address this issue, path-based algorithms, which generate, store and equilibrate a relative small number of paths for each O-D pairs, have been considered in the literature(9-11). In general, path-based algorithms work well for small- and medium-size problems, yet their efficiency degrade significantly with the problem size increasing (12).

This paper studies and compares a class of origin-based algorithms for the traffic assignment problem (TAP), which promise to produce highly precise UE solutions. All algorithms involved in this paper are characterized by the “origin-based” feature, by which TAP is decomposed into a sequence of subproblems with respect to origins; each one can be solved to an origin-based UE solution representation restricted on a single-origin network, and then loops among all origins to equilibrate the entire network. Under the above relatively relaxed definition, a number of algorithms can be classified into this family, most of which were developed in the past decade.

The best known algorithm of this class is the Origin-Based Algorithm (OBA) developed by Bar-Gera (13). Nie (14, 15) revised Bar-Gera’s OBA by fixing an error in computing the second-order derivative, and circumventing the line search in solving the restricted master problem (RMP) defined on each bush rooted at the origin. In this paper, we call Nie’s revised algorithm QBA following his original paper. Gentile (16) proposed to use the greedy method to solve the sequence of approximated quadratic programs, and named the resulting algorithm the Local User Cost Equilibrium (LUCE) algorithm. Dial (17) proposed the so-called Algorithm B by solving the RMP directly by swapping flows from the longest path to the shortest path for each node on the bush, taking advantage of the acyclicity of a bush. Zheng and Peeta (18) proposed a variant
of Algorithm B by successively refining an endogenously determined $\varepsilon$ value to solve the RMP into $\varepsilon$-optimal flows. The above five algorithms all recognize the fact that links with positive flows under UE will never form a directed cycle, and choose to maintain a bush for each origin and operate flows on bush links only. In this regard, these algorithms are also referred to as bush-based algorithms.

Another highly efficient algorithmic structure widely used in the design of origin-based algorithms is paired alternative segments (PAS). The concept of PAS was recently proposed by Bar-Gera (19), though the use of it for TAP actually has a much longer history. To our best knowledge, the Network Simplex (NS) method adapted by Nguyen (20) was the first algorithm that identifies and uses PASs for TAP. It maintains a spanning tree for each origin such that a PAS can be detected by augmenting an out-of-tree link into the tree. This idea is recently revisited by Zheng (21), who fixed a pivot rule error of NS and proposed a generalization of this algorithm by allowing the spanning tree to contain links that could either emanate from or point toward the origin. For discussion convenience, this revised NS algorithm is abbreviated as RNS in this paper. His numerical results testified the high efficiency of using PASs in solving large-size problems. It is somewhat surprising that the NS algorithm has not received much attention, though it has great potential to achieve high performance and it was applied for TAP earlier than the FW algorithm. Another algorithm that uses PASs is Dial’s Algorithm B (17), in which a bush is preserved for each origin instead. Thanks to the acyclicity of a bush, a PAS can be identified by following the shortest and longest path segments ending at the same node. Use of PASs for TAP was more systematically explored by Bar-Gera in developing his famous TAPAS algorithm (19), where a breadth first search method that can directly identify PASs from the shortest path tree rooted at each origin is applied. Xie and Xie recently (22) improved TAPAS by proposing a new PAS identification method that continuously identifies the higher cost segment with the maximum flow link in the backward set of the nodes and relates only one origin to each PAS. Their numerical experiments show that the improved TAPAS (termed iTAPAS) can be two times faster than TAPAS in solving large networks. The above five algorithms all identify and operate PASs for a TAP formulated at one origin. Hence we can call them PAS-based algorithms. Note that Algorithm B can be categorized into both the bush-based and PAS-based algorithms as it uses bushes and PASs simultaneously.

In this paper, our main research purpose is to make a fair and systematic comparison on the class of origin-based algorithms. To this end, we investigated all known origin-based algorithms and their variants and compared them in the following two aspects. First, we provided four algorithmic structures to accommodate the nine algorithms of interest. For algorithms that utilize the same structure, we compare them by examining their similarities and differences in algorithmic details. For algorithms that belong to different structures, we try to make a comparison on their computation costs of each iteration by estimating the frequency of scanning the set of nodes and links for each structure. (Hereafter in this paper, such an evaluation of the computation costs pertaining to these different algorithmic structures is termed *structural*...
complexity in this paper, which is different from the widely used definition of computational complexity. The latter typically takes into account very basic algorithmic operations, such as assignment, arithmetic and logical operations, in the worst case of problem instances.) Second, we implemented all nine algorithms on the same programming platform and let different algorithms share as many subroutines as possible. Extensive numerical experiments were conducted on both medium- and large-scale test problems to provide an overall comparison in terms of convergence efficiency. Several per-iteration convergence statistics are examined as well to provide insights and explanations for the convergence behavior of different algorithms.

Extensive and systematic comparisons of the latest UE algorithms are important and essential to transportation planning practitioners and software developers, since they need to know which algorithms will satisfy their needs. We note that Inoue and Maruyama (12) conducted a very good comparison on the traditional link-based, path-based, and some of origin-based algorithms. We advance their work by adding several new algorithms and place our focus only on the class of state-of-the-art origin-based algorithms. We also contribute the literature through this paper by estimating the structural complexity of these algorithms, which provides an analytical way to assess and compare the computation costs of the algorithms.

The rest of this paper is organized as follows. Section 2 describes the notation and algorithmic terms that are used in this paper. In Section 3, we briefly review and compare the nine algorithms considered in this paper by categorizing them into four algorithmic structures and estimating their structural complexity. The convergence performance of the nine algorithms are then compared and analyzed in Section 4. The last section concludes the paper with a few remarks on the major findings and future research.

NOTATION AND ALGORITHMIC TERMS

Consider a directed traffic network $G(N,A)$ where $N$ and $A$ denote the sets of nodes and links, respectively. Suppose $G$ is strongly connected, by which we mean that there is at least one path connecting any two nodes in $G$. $N$ consists of a set of nodes, some of which are origins denoted by $R$ and some others of which are destinations denoted by $S$. The travel demand departing from an origin $r \in R$ to an destinations $s \in S$ is denoted by $d_{rs}$. Link $(i,j)$ is directed from node $i$ to node $j$. Let $x_{ij}$ be the traffic flow on link $(i,j) \in A$ and $t_{ij}(\cdot)$ be the travel time of traversing link $(i,j)$, which is a strictly positive, increasing and convex function of $x_{ij}$.

Given the above network, the UE-based TAP can be formulated as a convex optimization problem with linear constraints. Existing algorithms usually decompose a TAP into a series of subproblems and solve them iteratively. Central to the development of origin-based algorithms is the decomposition of traffic assignment with respect to origins (or destinations). More specifically, a TAP is decomposed into a sequence of single-origin TAPs, each one being defined on a subnetwork.
Definition 1 (Subnetwork). A subnetwork is a duplicate of network $G$ but rooted at a single origin $r$. Only the demand departing from origin $r$ can be assigned to links of the subnetwork. We let $G_r(N,A_r)$ denote a subnetwork.

Maintaining a subnetwork for each origin is essential for origin-based algorithms. Link flows need to be disassembled into the origin-specific link level and to be stored on the links of the subnetwork, denoted by $x^r_{ij}$. Origin-based link flows on the subnetworks contain much detailed information that can be used to generate precise descent directions. Subnetworks usually only provide a platform to decompose a TAP into multiple single-origin TAPs. They do not have the potential to force a feasible flow pattern to be optimal. Next, several other algorithmic devices that provide key blocks of methods used for solving these single-origin TAPs are given below.

Definition 2 (Bush). A subnetwork is called a bush if it is acyclic. It is denoted by $B(N,A_B)$.

Some network operations (such as a shortest path search) become more efficient in a bush by making use of topological distance, 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 node $j$ (14).

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 (14).

On a bush, the shortest path and the longest path trees can be built quickly and simultaneously by scanning each node and link once.

Definition 5 (Spanning tree). A bush is a spanning tree if it contains a minimal set of links that connect all nodes.

Definition 6 (Segment). A segment is defined as a sequence of distinct nodes and there is only one directed link between every pair of nodes. The beginning and ending nodes of a segment are called the tail and head, respectively. A segment is called a directed segment if links of the segment are all directed from the tail to the head.

Definition 7 (Cycle). A segment and one link that connects the tail and the head of this segment constitute a cycle. A cycle is called a directed cycle if all links of the cycle are directed clockwise or anticlockwise.

Definition 8 (Paired alternative segments). A set of paired alternative segments (PAS) is defined as two segments that have the following features: (1) they have the same tail and head nodes, other than which there are no more common nodes; (2) they are both directed segments (19).
OVERVIEW AND COMPLEXITY ANALYSIS

In this section, we provide an overview of the origin-based algorithms and their algorithmic structures. Nine algorithms are categorized into four algorithmic structures according to their algorithmic design resemblance. Based on these structures, we first review the key differences and similarities of the algorithms within each algorithmic structure; then compare the algorithms across different implementation structures by approximating the structural complexity of their subproblems, which can be used to assess computing costs in individual iterations.

Algorithm Overview

Algorithmic Structure 1

OBA, QBA and LUCE share the same algorithmic structure described as follows. In step 1, we utilize the bush management strategies suggested by Nie (14) for all three algorithms. The key difference among them are in step 2.2, where the RMP is decomposed into a sequence of quadratic approximation problems defined on each node, which are solved subsequently to provide a descent direction. OBA employs a gradient projection (GP) algorithm for these quadratic approximation problems, while LUCE uses a greedy algorithm instead. QBA follows OBA by using the GP algorithm for quadratic approximation problems, but it improves the veracity of these approximation problems by updating the cost and derivatives of links in the upper stream after each quadratic approximation problem is solved. OBA and LUCE need to adjust the step size of the line search for the RMP. OBA invokes a descending pass for each line search adjustment to make sure that the descent direction obtained can always diminish the objective function. LUCE does not invoke a descending pass and QBA further waives the line search.

Step 0 (Initialization): For each origin, initialize the bush as a shortest path tree rooted at the origin. Assign all flows to links on the tree.

Step 1 (Bush management): For each bush, do the following:

1.1 Update the shortest path tree and node labels by an ascending pass, scan each link in the bush once to trim the bush links.

1.2 Update the topology order, then update the shortest and longest path tree and node labels; scan every link out of the bush to expand the bush. Update the topology order again.

Step 2 (Bush equilibration): Repeat the following steps 20 times at most. For each bush, do the following:

2.1 Calculate the shortest path tree, update node and link labels by an ascending pass.
2.2 Solve the approximated quadratic problem defined on each node by a descending pass.

Step 3 (Convergence check): If the convergence criterion is achieved, stop; otherwise, go to step 1.

Algorithmic Structure 2

Algorithm B and εBA share the Algorithmic Structure 2. In our implementation, the bush management strategies adopted in Structure 2 is same with the one in Structure 1, and their main difference lies in the bush equilibration step. In step 2, Algorithm B calculates the longest and shortest path cost from the origin to each node in the bush; if the cost difference is larger than an exogenously given precision ε, then a PAS is searched and equilibrated for this node. Here the choice of value of ε appears to be a concern for Algorithm B in the sense that, if a relative large value of ε is chosen, then the algorithm may fail to converge to an expected precision; if a smaller value is chosen, then the number of PAS identified in each iteration may be too large to be needed, i.e. in the early stage of algorithms when the relative gap is still much larger than ε, then identifying PASs with a relatively small cost difference usually make negligible effects on convergence but would produce considerable computational effort. εBA proposes to successively refine an endogenously determined ε value to control the generation of PAS in each iteration more elaborately according the convergence needs. In εBA, ε is initialized as a relatively large value and refined in each iteration, and it reduces to zero gradually as the algorithm converges to the UE solution. One possible shortcoming of εBA is that one has to choose an initial value of ε and set a proper refinement process for it in the algorithmic procedure, mostly according to experience.

Step 0 (Initialization): For each origin, initialize the bush as a shortest path tree rooted at the origin. Assign all flows to links on the tree.

Step 1 (Bush management): For each bush, do the following:

1.1 Update the shortest path tree and node labels by an ascending pass, scan each link in the bush once to trim the bush links.

1.2 Update the topology order, then update the shortest and longest path tree and node labels; scan every link out of the bush to expand the bush. Update the topology order again.

Step 2 (Bush equilibration): Repeat the following steps 20 times at most. For each bush, do the following:

2.1 Calculate the shortest path and longest path trees, update the node labels by an ascending pass.
2.2 Perform a descending pass to search and equilibrate a PAS for each node that its longest path and shortest path cost difference is larger than a specific coefficient $\epsilon$. One segment of the PAS is searched following the shortest path, and the other follows the longest path.

**Step 3** (Convergence check): If the convergence precision is achieved, stop; otherwise, go to step 1.

Algorithmic Structure 3

NS and RNS share Algorithm Structure 3. Other than a subnetwork, NS maintains a directed spanning tree for each origin in order to identify a PAS by adding an out-of-tree link $(i, j)$ into the tree. The identified PAS is then equilibrated by shifting flows from the higher cost segment to the lower cost segment to force the reduced cost of link $(i, j)$ to zero. Note that if one link on the higher cost segment has zero origin flow, then the PAS will be unable to shift any amount of flow for link $(i, j)$ for this origin. A search of this kind is called a degenerated search. Degenerated searches may incur useless computation efforts or even make the algorithm unable to converge. For this reason, special techniques are usually needed for NS to handle degenerated searches. After the PAS is equilibrated, the tree is updated by removing the zero-flow link from the tree and adding a non-zero link to keep the tree connected. Zheng (20) fixed an error in the pivot rule of NS that may generate directed cycles and subsequently fail to update the tree properly. He further proposed a variant of this algorithm (named RNS) that allows the spanning tree to contain links that could be either directed away from or toward the origin, such that his revised version of NS searches cycles instead of PASs. The superiority of such a revision is that a strongly feasible spanning tree can be maintained for each origin, which will not generate degenerated searches; the shortcoming is making it more complex to implement the algorithm.

**Step 0** (Initialization): For each origin, initialize the subnetwork and the spanning tree as the shortest path tree. Assign all flows to links on the tree.

**Step 1**: For each spanning tree, calculate the path cost from the origin to each node by a depth first algorithm (DFS).

**Step 2**: Repeat the following steps 20 times at most. For each origin, do the following:

1. Scan every link out of the spanning tree, if the reduced cost of link $(i, j)$ is negative or positive but with nonzero origin flow, search a PAS or cycle in step 2.2.

2. Search a PAS or a cycle by adding link $(i, j)$ into the spanning tree, and augment flows around the PAS or cycle to equilibrate the cost. Update the spanning tree with a pivot rule.

**Step 3** (Convergence check): If the convergence precision is achieved, stop; otherwise, go to step 1.
TAPAS and iTAPAS share Algorithmic Structure 4. TAPAS systematically explores the use of PAS for solving TAP and its three notable innovations are summarized as follows: (1) it employs a breadth first search method to identify PASs directly from the shortest path tree rooted at each origin at step 1.2; (2) it stores PASs in a list in order to save the time spent on repeatedly identifying same segments at step 1.2; (3) it connects multiple origins to one PAS by matching an origin to an existing PAS in the list, which helps secure a unique path solution by forcing proportionality. Xie and Xie examined the TAPAS algorithm in great detail and found its several issues related to the identification and operations of PASs that may affect the overall algorithm convergence efficiency, including: (1) the breadth first search method chooses the higher cost segment arbitrarily, which could lead to unpredictable and inconsistent behavior; (2) the breadth first search method may fail to identify a PAS successfully if a flow effective factor is imposed, such that the algorithm convergence cannot be guaranteed unless an additional branch shift operation is conducted; (3) the association of multiple origins with one PAS may incur a large number of link flow operations that lag the algorithm convergence in solving large, congested networks. The resulting improved algorithm is named iTAPAS, which differs TAPAS mainly in the following aspects: (1) a new PAS identification method which is easier, faster and more effective is proposed to instead the breadth first search method and the branch shift operation. The new method identifies the higher cost segment continuously with the maximum flow link in the backward set of the node that starts from tail node of the target link to the origin; (2) only one origin is related to each PAS in step 1.2 in the algorithm convergence process and a post-processing can be added to achieve the proportionality. Implementation of iTAPAS becomes significantly easier than that of TAPAS.

**Step 0 (Initialization):** For each origin, initialize the subnetwork as the shortest path tree. Assign all flows to links on the tree.

**Step 1:** For each origin, do the following:

1.1 Calculate the shortest path tree rooted at the origin.

1.2 Scan each link out of the tree: if the origin link flow is larger than 0 and the reduced cost is smaller than 0, match an existing PAS from the PAS list for this link; if an existing PAS is not matched, identify a new PAS based on the shortest path tree. Equilibrate the segments cost once for the new identified PAS and save it into the PAS list if both segments still have positive flows.

1.3 Select a number of PASs (say, 400) from the PAS list and equilibrate the segments cost once.
Step 2: Repeat the following step 20 times at most: scan each PAS in the list, if the flow (contributed by all associated origins) on one segment reduces to 0, delete it from the PAS list; otherwise, equilibrate the segments cost once for this PAS.

Complexity Analysis

Given the above structures, the algorithms accommodated by the same structure tend to share more common features than those in different structures. Here we propose to evaluate the structural complexity of different algorithmic structures in each iteration by estimating the frequency that they scan the set of nodes and links, which are usually the most expensive step in TAP algorithms. Such a complexity analysis provides us with helpful insights of how many the computation efforts are needed to conduct one algorithmic iteration by different algorithmic structures. Table 1 reports the structural complexity of different algorithmic structures. Note that scanning a general network is commonly more expensive than scanning a bush or a tree. Here we do not distinguish them because the scanning process involves link flow and cost operations that usually spread over the whole node set and link set, so the structural complexity difference caused by different network topology is negligible. Let $R$ denote the number of origins, and $P$ denotes the number of PASs. Structure 1 and Structure 2 share the same bush management strategies, so the structural complexity of step 1 for both two structures are $6R$. We roughly estimate the structural complexity of the bush equilibration (step 2) in Structure 1 as $40R$ and that in Structure 2 as $20R$, respectively. We employ a fast implementation strategy suggested by (18) for B and $eBA$, such that over 50% of the network scanning usually can be omitted in the Structure 2. Structure 3 only scans the network once for each origin in step 2 such that its structural complexity is estimated as $20R$. The structure of TAPAS and iTAPAS has one major difference from the other three structures that it stores PASs into a list in step 1 and subsequently obviates scanning over the network in step 2. When the number of PASs in the list is smaller than the node set size, Structure 4 spends less computational efforts than that of Structure 3; as the PAS list size increases to a size that is comparable to that of nodes or links, then the computational effort of Structure 4 is roughly equal or even larger than that of Structure 3.

Note that the structural complexity estimation cannot reflect the algorithm convergent efficiency unless we further know the average convergence magnitude achieved by each iteration. Actually, these two parts are subject to a tradeoff in designing a TAP algorithm. Take Algorithm B described in Structure 2 as an example, a larger number of repetitions of bush equilibration will invoke more ascending and descending passes of the bush and thus increase the computation time of each iteration, but it also will let the RMP more converged. Even though, the complexity analysis is still very useful in explaining our numerical experiments results and making our comparison results more convincible in next section, where the average computation times and the average convergence magnitude per iteration can be calculated based on the assignment results.
Table 1 Structural complexity of different algorithmic structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBA, QBA, LUCE</td>
<td>$46R$</td>
</tr>
<tr>
<td>2</td>
<td>B, $\varepsilon$BA</td>
<td>$26R$</td>
</tr>
<tr>
<td>3</td>
<td>NS, RNS</td>
<td>$21R$</td>
</tr>
<tr>
<td>4</td>
<td>TAPAS, iTAPAS</td>
<td>$2R + 20P$</td>
</tr>
</tbody>
</table>

Note: $R$ denotes the number of origins; $P$ denotes the PAS list size.

NUMERICAL ANALYSIS

We will further compare the nine origin-based algorithms by conducting a set of numerical experiments. This section is organized into three parts. The first part describes the algorithm implementation environment and the test networks used in the numerical experiments. The performance of the nine algorithms in solving different networks are compared in the second part. The last part examines the average computation time and the average convergence magnitude by each iteration of the algorithms for large networks.

The Computing Environment and Test Networks

All nine algorithms are coded using the toolkit of network modeling (TNM) (23), a C++ class library specialized in modeling transportation networks. In our implementation, different algorithms share as many subroutines as they can, and their main differences are all described based on the four algorithmic structures identified in the last section. All numerical results reported in this section were produced on a Windows 8 64-bit workstation with Core i5 3.2GHz CPU and 16G RAM.

Table 2 Profile information of the test networks used in the numerical experiments

<table>
<thead>
<tr>
<th>Network</th>
<th>Number of nodes</th>
<th>Number of links</th>
<th>Number of zones</th>
<th>Number of O-D pairs</th>
<th>Average link V/C ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Winnipeg</td>
<td>1,050</td>
<td>2,836</td>
<td>135</td>
<td>4,345</td>
<td>0.33</td>
</tr>
<tr>
<td>Chicago Sketch</td>
<td>933</td>
<td>2,950</td>
<td>387</td>
<td>142,890</td>
<td>0.54</td>
</tr>
<tr>
<td>Large</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRISM</td>
<td>14,639</td>
<td>33,937</td>
<td>898</td>
<td>453,089</td>
<td>0.26</td>
</tr>
<tr>
<td>Chicago</td>
<td>12,982</td>
<td>39,018</td>
<td>1,771</td>
<td>3,136,441</td>
<td>0.60</td>
</tr>
</tbody>
</table>
The algorithms are applied to six networks, among which two networks are medium-size and four networks are large-scale. The profile information of these networks is reported in Table 2. The last column shows the average volume/capacity (V/C) ratio of networks at user equilibrium, we can see that Austin is the most congested network and PRISM is the least congested one in the four large regional networks. The main convergence indicator used in this study is the relative gap, which measures how close the current solution is to the true user equilibrium. In our experiments, the relative gap (RG) is calculated as,

\[
RG = 1 - \sum_{rs} u_{rs} q_{rs} / \sum_{(i,j) \in A} x_{ij} t_{ij}
\]

where \(u_{rs}\) is the minimum travel cost between O-D pair \(r-s\) based on the current link travel cost \(t_{ij}\), and \(q_{ij}\) is the travel demand between the O-D pair \(r-s\). Finally, the Bureau of Public Roads (BPR) function is used to calculate link travel costs.

**Overall Convergence Performance**

This section compares the convergence performance of the nine algorithms on different networks. The convergence criterion is set to \(10^{-12}\) and the maximum running time is set to six hours for all experiments. Figure 1 illustrates the evolution of different algorithms in solving the six networks. The horizontal axis of these graphs denotes the computation times in seconds/minutes, the vertical axis denotes the RG in the logarithmic scale. All nine algorithms can solve the two medium-size networks to a relative gap of \(10^{-12}\) within two minutes and hence testify their capacity of achieving highly precise solutions. The convergence capacity of all algorithms tends to degrade with the size and congestion level of test networks. For the four larger regional networks, most algorithms except for LUCE can converge to RG of \(10^{-12}\) in solving the less congested PRISM network, and nearly all algorithms other than iTAPAS fail to reach RG of \(10^{-12}\) in solving the most congested Austin network. Apparently, how much the performance of an algorithm degrades with the size and congestion level increasing is a critical consideration for researchers and practitioners to choose an algorithm, and TAPAS or iTAPAS is usually preferable to OBA or LUCE.
FIGURE 1 Convergence performance of nine algorithms on different test networks
Particularly, iTAPAS is found to be the algorithm with the best performance in all cases, and TAPAS follows as the second best algorithm. Note that iTAPAS can be two times faster than TAPAS in solving the Chicago Regional, Philadelphia and Austin networks, and this may attributes to the new PAS identification method and the simplified PAS flow operations of iTAPAS. NS, RNS, Algorithm B and $\epsilon$BA perform similarly in our test. RNS may outperform the other three slightly if examined closely. NS stops to converge further at RG of $10^{-8}$ in solving the Philadelphia and Austin networks for unclear reasons. OBA, QBA and LUCE perform less efficiently than the other six algorithms to a large extent, mainly due to their unfaithful approximation of RMP into a sequence of quadratic node problems. The reader is referred to (24) for detailed discussions about these three algorithms.

Convergence Performance per Iteration

In this section, we will further explore the assignment results, particularly in several per-iteration statistics, in order to provide more comparisons in the convergence behavior of different algorithms, and to give possible explanations for the degradation effects of algorithms with network size and congestion level. Recall that the overall convergence of algorithms can be determined by two aspects: the average computation time per iteration and the average convergence magnitude per iteration. Given the assignments results, we can easily calculate the average computation time of one iteration for each algorithm on a specific network. Here we further define an indicator $M$ to measure the average convergence magnitude of each iteration as follows:

$$M = \sum_{k \in [1..K-1]} (\log R_G_k - \log R_G_{k+1}) / (K - 1)$$

(0)

where $R_G_k$ denotes the relative gap defined in equation (0) on iteration $k$, and $K$ is the maximum iteration index when the algorithms converge to $RG$ of $10^{-12}$ or reach the maximum running time.

The two computation statistics averaged by all iterations for each algorithm and each network are reported in Table 3. $K$ denotes the total number of iterations considered, $T$ denotes the average computation time of one iteration and $M$ is the convergence magnitude indicator defined above. We can obtain some useful conclusions from this table. By taking the statistics of LUCE in solving the Chicago Regional network as an example, we found that it takes about 18.2 minutes for LUCE to achieve $M$ of 0.23. If we optimistically assume that $T$ and $M$ in this case will keep unchanged through the whole convergence process, then it is easy to calculate that LUCE will take at least 15.8 hours$^1$ to converge to $RG$ of $10^{-12}$ in this case.

---

$^1$ This is calculated as $[(\log 1 - \log 10^{-12})/0.23] \times 0.303 \approx 15.8$ hours.
1 TABLE 3 Average computation time and convergence magnitude per iteration

<table>
<thead>
<tr>
<th>Network</th>
<th>Statistics per iteration</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OBA</td>
</tr>
<tr>
<td>PRISM</td>
<td>$K$</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>$T$</td>
<td>7.38m</td>
</tr>
<tr>
<td></td>
<td>$M$</td>
<td>0.816</td>
</tr>
<tr>
<td>Chicago Regional</td>
<td>$K$</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>$T$</td>
<td>13.3m</td>
</tr>
<tr>
<td></td>
<td>$M$</td>
<td>0.240</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>$K$</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>$T$</td>
<td>16.5m</td>
</tr>
<tr>
<td></td>
<td>$M$</td>
<td>0.342</td>
</tr>
<tr>
<td>Austin</td>
<td>$K$</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>$T$</td>
<td>7.25m</td>
</tr>
<tr>
<td></td>
<td>$M$</td>
<td>0.089</td>
</tr>
</tbody>
</table>

Note: $m$ denotes minutes.

2 In Section 3, we propose to estimate the structural complexity of each structure to obtain a glance at the computation effort/time needed for one iteration. The real computation times shown in Table 3 show a consistent pattern with the prediction of the structural complexity. We plot these computational times in Figure 2. The per-iteration computation times of OBA, QBA and LUCE are clearly larger than those of other six algorithms in all cases, as consistent with the complexity analysis results. iTAPAS is always the algorithm with the lowest computation time in all the test cases. The remaining five algorithms are comparable and a close look reveals that the computation times of Algorithm B and $\varepsilon$BA are slightly larger than those of NS and RNS. The computation time of TAPAS seems to heavily depend on the number of PASs maintained. Our assignment results report that the average number of PASs maintained by TAPAS per iteration for the PRISM, Chicago Regional, Philadelphia and Austin networks are 333, 14,792, 48,913 and 81,768, respectively. As the number of PASs increases, the computation time required per iteration increases accordingly. The variations of computation times of iTAPAS is less obvious for the following two reasons: (1) iTAPAS maintains less PASs than TAPAS does per iteration due to the more effective PAS identification, those numbers are 253, 9,692, 20,017, 56,429 for
the PRISM, Chicago Regional, Philadelphia and Austin networks, respectively; (2) iTAPAS associates only one origin with each PAS, so its operations on a PAS take less time than that of TAPAS, which relates multiple origins to one PAS.

(a) PRISM

(b) Chicago Regional
Based on the iteration statistics in Table 3, one can further calculate the algorithm convergence rate (termed CR) as follows,

$$CR = \frac{M}{T}$$
The above convergence rate can reflect the average convergence magnitude in terms of time. Figure 3 shows the convergence rate of different algorithms for the Chicago Regional network. It reveals that iTAPAS is the most time-effective algorithm in this example, and it is more than two times faster than TAPAS.

Figure 3 Convergence rate of different algorithms for the Chicago Regional network

CONCLUDING REMARKS

In this paper, the class of origin-based algorithms for TAP under the UE principle is investigated and compared in both the analytical and numerical aspects. All nine known origin-based algorithms are accommodated into four algorithmic structures, each of which distinguishes from others in terms of some structural differences. We implemented these algorithms on the same programming platform, which allows us to conduct a fair comparison on their convergence performance. The results of numerical experiments reveal that the relative performance of different algorithms is consistent with our complexity analysis.

All the origin-based algorithms own the capacity of achieving a highly precise solution for TAP under the UE principle. However, our numerical results show that the level of precision that each algorithm is capable of achieving tends to degrade with the network size and congestion level increasing. It seems that the worse an algorithm performs, the more it degrades. This phenomenon was further explored by decomposing the convergence efficiency into two parts, the average computational time and the average convergence magnitude per iteration. The algorithms with worse performance usually have a larger per-iteration computational time because they scan the network more frequently, and have a smaller per-iteration convergence magnitude because they usually execute link flow/cost operations more ineffectively.
Hence, one can, of course, improve the efficiency of the origin-based algorithms by either reducing the computation time per iteration or increasing the convergence magnitude per iteration. However, these two aspects usually are subject to a tradeoff. Our complexity analysis and numerical computation results show that the average computation time per iteration is closely relevant to the frequency of network scanning. Many existing algorithms were designed to scan the origin list (or the subnetwork list) multiple times in each main iteration in order to increase the convergence magnitude per iteration, in the expense of increasing the computational time per iteration. This technique is quite effectively in fastening the overall convergence efficiency when the network size is small or medium where the computation time incurred by repeating network scans is not considerable. In the case of large regional networks, Nie (14) suggested to check a convergence criterion before an origin is scanned, such that the scanning operation on this subnetwork can be skipped if it is well converged. Zheng (21) further proposed to skip $K$ times of scanning operations if one subnetwork is converged under a certain precision, such that the computational time per iteration diminishes significantly while keeping the convergence magnitude unchanged. Following this line, TAPAS (19) spends its most computational effort on the flow/cost operations of PASs while successfully avoiding repeated scans of subnetworks, which partly accounts for its high efficiency. Our analytical and numerical analysis in this paper well verifies the above experience in a full-scale perspective.

The past decade observes the development and improvement of various origin-based algorithms for TAPs of the UE type. Our limited experience suggests that the following aspects may deserve further research efforts in the future. First, in this paper, we placed our main focus on an implementation and numerical comparison. However, the theoretical and technical reasons behind the algorithm performance are rarely discussed. Hence, a comprehensive review on existing mathematical and algorithmic principles on traffic assignment algorithms is valuable and desired. Second, given that most efficient algorithms for large-scale problems presented in this paper all employ the PAS structure, a further exploration and application of PAS for traffic assignment algorithms or for algorithms for other types of network equilibrium problems is recommended. Third, use of PAS and many other algorithmic devices in origin-based algorithms naturally allows the implementation of parallel computing. Introducing parallel computing techniques into origin-based algorithms is another exciting research direction.

ACKNOWLEDGEMENTS

We are grateful to Professor Yu Nie at Northwestern University for his kindness of sharing the TNM programming package, without which we cannot finish this work. The numerical results of OBA, QBA and LUCE were produced by the code written by Professor Nie, and the code of the rest six algorithms was written by the first author. We also want to thank the four anonymous reviewers for their valuable comments. Any possible errors are with the authors alone. This research is jointly sponsored by a research project awarded to the first author by the China Postdoctoral Science Foundation (Grant No. 2014M551414) and a research grant awarded to the
second author from the State Key Laboratory of Ocean Engineering at Shanghai Jiao Tong University (Grant No. GKZD010061).

REFERENCES


