A Temporal Domain Decomposition Algorithmic Scheme for Large-Scale Dynamic Traffic Assignment

Eric J. Nava,
Department of Civil Engineering and Engineering Mechanics, University of Arizona, 1209 E 2nd St, Tucson, AZ, USA
ejnava@email.arizona.edu
+1 520-626-3367

Yi-Chang Chiu1
Department of Civil Engineering and Engineering Mechanics, University of Arizona, 1209 E 2nd St, Tucson, AZ, USA
chiu@email.arizona.edu
+1 520-626-8462

ABSTRACT
With emergent interest of Simulation-Based Dynamic Traffic Assignment (SBDTA) in the field of transportation network modeling, deployment of SBDTA models for traffic operations and transportation planning have increased significantly in recent years. In parallel, research and development of innovative approaches of the SBDTA model have enhanced the quality of both the assignment component, i.e, improvement of convergence quality of the Dynamic User Equilibrium (DUE) problem, and the traffic simulation element. However, computational requirement remains to be one of the great challenges for DTA implementations on large-scale networks with a long analysis period.

This paper presents a temporal decomposition scheme for large spatial- and temporal-scale dynamic traffic assignment, in which the entire analysis period is divided into Epochs. Vehicle assignment is performed sequentially in each Epoch, thus improving the model scalability and confining the peak run-time memory requirement regardless of the total analysis period. A proposed self-turning scheme adaptively searches for the run-time-optimal Epoch setting during iterations regardless of the characteristics of the modeled network. Extensive numerical experiments confirm the promising performance of the proposed algorithmic schemes.

Keywords: dynamic traffic assignment, method of isochronal vehicle assignment, simulation, computational efficiency, large scale

1Corresponding Author: Telephone: 520-626-8462; Email: chiu@email.arizona.edu

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
1. INTRODUCTION
Dynamic Traffic Assignment (DTA) methodology is ever-evolving, growing both theoretically and practically since the influential research of Merchant and Nemhauser (1978a; 1978b). Extended from Wardrop’s user equilibrium (UE) principle, the dynamic UE (DUE) principle states, generally, that travelers on all used routes of the same origin-destination (OD) pair departing within the same departure time window should have the same experienced travel time (cost), and travelers departing at different departure time windows may have different experienced travel time due to the onset and duration of time-varying congestion. The experienced travel time cannot be anticipated at departure, but must be evaluated through experience and learning of time-varying link travel times and node delays. This DUE condition allows the model user to compare scenarios with a plausible behavioral foundation of route choice, as well as stable and consistent solutions in comparative analysis of scenarios. DTA has the potential to address rising issues related to the deployment of innovative technologies and strategies for demand management, congestion management, incident management and others. Such emerging issues, in need of higher-resolution of complex performance measures for informed decision making, cannot be sufficiently answered by static assignment models.

Historically, DTA research has branched into two modeling approaches that can be generally categorized as analytical or simulation-based methods. A simulation-based dynamic traffic assignment (SBDTA) model typically employs a traffic simulator as the network loading method to capture complex demand/supply interactions, whereas analytical models apply mathematical formulations such as exit/volume-travel time functions for similar purposes.

As shown in Figure 1, the common algorithmic framework of a SBDTA model includes the iterative execution of network loading (simulation), the path set update procedure (time-dependent travel time cost of routes for all origin, destination, and departure time triplets), and the path set adjustment (DTA) procedure to update vehicle paths. To determine how close the current solution is to the DUE condition, the evaluation of path assignments, by means of simulation, requires checking a defined convergence criterion. The algorithm terminates if the stopping criterion is met.

With rising interest in SBDTA from practitioners and researchers who recognize the potential capability of the DTA methodology, deployment of SBDTA models for traffic operations and transportation planning have increased. In parallel, research and development of innovative approaches of the SBDTA model have improved, particularly in the realms of DUE solution methodology and traffic simulation. However, computational requirement for large-scale implementation of a DTA model remains to be one of the great challenges. Network size, e.g., links, nodes, and zones, the amount of vehicles being simulated/assigned and the length of the analysis period are the key factors determining the computational requirements of the SBDTA model, thus potentially making assignment procedures intractable. The assignment procedure becomes intractable as the combination of network size and the analysis period expand. Typically, most existing SBDTA models can only handle short analysis periods (peak periods) when operating in a personal computing environment (e.g., desktop computers...
and server computers). The concern then arises in whether SBDTA models can withstand large, regional networks with extensive analysis periods given limited computational resources.

In light of enhancing the computational performance of DTA models, critical literature can be found in the respective areas of network loading/simulation and path processing. A typical methodology to improving the computational efficiency of simulation is parallel processing. Both spatial and temporal domain partitioning strategies were found in literature. Spatial domain partitioning can be also be further classified into set partitioning or graph partitioning. Set partitioning is to partition the set of simulation objects such as sets of links or vehicles into groups. The computation of each group is handled by different computing processors, regardless of whether these processors access the shared memory or not. The set partitioning method is a widely used technique for parallelizing simulation computations in which at each simulation interval, the loop for simulating vehicle movements on each link or updating signal timing at each node is partitioned into smaller “chunks” so that they can be computed in parallel (Junchaya and Chang 1993; Chronopoulos and Johnston 1998; Peeta and Chen 1999; Nokel and Schmidt 2002; Klefstad, Zhang et al. 2005; O’Cearbhaill and O’Mahony 2005).

Graph partitioning is to partition the entire network into smaller sub-networks and have each sub-network’s computations handled by different computing processors, and in parallel. In general, the paralleled processed objects need to be logically independent, namely the update of the state of one object does not affect the boundary condition of another object. The simulation of each sub-network needs to be synchronized periodically and vehicles crossing the sub-network boundaries need to be reconciled at each synchronization instance (Nagel and Rickert 2001; Lee and Chandrasekar 2002; Liu, Ma et al. 2005). The graph partitioning concept is also applied in the shortest path problem, in which a network is partitioned into a disjoint set with a certain number of
common “boundary” links or nodes connecting neighboring sub-networks. In this strategy, each compute node updates the node/link labels within its domain and passes the updated label to the neighboring domains (Habbal, Koutsopoulos et al. 1994; Romeijn and Smith 1997; Hribar, Taylor et al. 2001; Chabini and Ganugapati 2002).

Temporal partitioning of simulation is conceptually more challenging due to the temporal state dependence in traffic simulation. A method termed time-parallel traffic simulation was proposed and discussed pertaining to the advantages of temporal partitioning and the concept of approximative state-matching in which the final and initial states of adjacent time intervals are approximated with some level of introduced error (Kiesling and Pohl 2004; Kiesling and Luthi 2005). However, this method is applicable to very specific cases and is not feasible for general traffic simulation applications.

The time-dependent shortest path (TDSP) algorithm is a critical and computationally demanding step of the overall SBDTA solution process. In the aforementioned DTA algorithmic scheme, the TDSP is computed to update the path set for each origin, destination (centroid), and departure time triplet. In order to improve the computational efficiency of such an all-to-many TDSP, various strategies have been attempted in literature. One of the concepts is to re-optimize shortest paths for a new destination by identifying the node potential (Gallo 1986) from the previously solved shortest path. While applicable to a Dijkstra-type static shortest path, the comparable concept has also been applied to time-dependent cases (Pallottino and Scutellí 2003; Miller-Hooks and Yang 2005). Other widely applied strategies include destination-based or topology-based parallelization schemes. The destination-based parallel processing scheme recognizes that the TDSP tree for each destination (assuming the algorithm is destination-based) can be computed independently; therefore, in a multiple-processor environment, each processor computes only a sub-set of the destinations. (Hribar 1997; Ziliaskopoulos, Kotzinos et al. 1997; Hribar, Taylor et al. 2001; Chabini and Ganugapati 2002).

Solving for the DUE solution in a distributed computing environment is relatively limited in literature. Wisten and Smith (1997) calculated the DUE solution in terms of intersection flow split rates and proposed an algorithm to compute this in a distributed computing manner. The computational performance of a DTA model following the algorithmic structure shown in Figure 1 has not been extensively studied in literature, although, both existing parallel computing techniques can be applied to both network loading and the TDSP.

Comparing the shared-memory and message-passing computing paradigms, the message-passing is memory scalable but not shared-memory as parallel processing requires duplicating array/objects in memory so that computations can take place in parallel. A personal computing environment contains only a fixed amount of physical (and virtual) memory. Computation cannot continue (or may be severely slowed down) if the algorithm requests memory usage exceeding the available physical (and virtual) memory.

Message-passing is scalable because multiple standalone computers work as a whole and more computers can be added if more memory is required. Generally, these
computer cluster systems are still not widely adopted in transportation profession other than some advanced research institution. Multi-core shared-memory continues to be a prevalent computing environment. Differing from most existing literature, our primary interest of this study is, assuming shared-memory architecture, on the temporal decomposition of the DUE problem, particularly on the TDSP and assignment procedure, such that the memory usage of the entire DUE solution procedure becomes scalable and the run-time speed is also optimized. Improved memory scalability is crucial in solving large-scale (e.g. region-scale), long time period (e.g., 24-hour period or longer) DTA models regardless of the computer memory limitation. Most known DTA model applications on shared-memory computers are limited to a corridor-based model and/or modeling a short time period. A computational scheme that enables a successful application of DTA model in a large regional scale and 24-hour simulation and assignment on a shared-memory computer is non-existent.

In this study, the Method of Isochronal Vehicle Assignment (MIVA) is proposed in which the temporal domain of the TDSP and path assignment is decomposed, thereby improving the scalability of the SBDTA model and allowing for simulation and assignment of long-period, large-scale models. The self-tuning adaptive (STA) algorithm is also developed to search for an optimal MIVA specification during run-time. To the best of our knowledge, this paper is the first known effort to enable SBDTA for region-sized networks with diurnal demand modeling period.

In the following sections, first provided is the explanations of the MIVA concept and algorithmic structure in Section 2. The computational performance of the proposed MIVA algorithm is shown through extensive numerical experiments presented in Section 3. The run-time improvement achieved by the self-tuning adaptive (STA) algorithm is presented and discussed in Section 4. Section 5 concludes this paper.

2. THE METHOD OF ISOCHRONAL VEHICLE ASSIGNMENT FOR DYNAMIC TRAFFIC ASSIGNMENT

The proposed Method of Isochronal Vehicle Assignment (MIVA) decouples the time domain between simulation and both the path set update and path adjustment procedures (comprised of both the TDSP and assignment solution algorithm) into forward-sliding time periods which allows the memory requirement for both the path set update and path adjustment to be bounded solely on the length of the determined temporal segment instead of the entire analysis period.

The memory usage for storing time-varying link travel times is of memory size $|I| \times |T|$, where $I$ is the set of links and $T$ is the set of assignment intervals. For storing the time-varying node (intersection) delay is of size $|I| \times |M| \times |T|$, where $M$ is the set of movements. These arrays are of modest size, even for a large network and long analysis period. The TDSP memory requirement, depending on the algorithmic implementation, generally requires the memory for storing many arrays with dimension $|I| \times |M| \times |J| \times |T|$, where $J$ is the set of destinations (or zones if the destination is the centroid of the zone). For example, a network with 20,000 links, 1,000 zones, 5 movements per intersection, and 100 departure intervals would need $1 \times 10^6$ elements to store information for one array, let alone the multiple arrays typically needed during full network TDSP
computation. The applied TDSP algorithm for this research is label-correcting with complexity $O(nmT^2)$, where $n$ is the number of nodes, $m$ is number of links, and $T$ is number of assignment intervals, it is apparent that the TDSP computational time will grow polynomially with network size and the analysis period. The memory usage for the assignment procedure, although varied by implementation, typically would require a significant amount of memory to store time-dependent path set for each origin-destination-departure time triplet. The memory usage is linear in the temporal domain, but could be large if each path is stored in terms of individual nodes/links comprising the path.

### 2.1 The MIVA Temporal Decomposition Framework

Shown in Figure 2, the MIVA scheme is denoted by two inter-associated time periods: *Epoch* and *Projection Period*. The MIVA scheme decouples the temporal domain of the analysis period (also termed simulation period) into sequential segments of equal length called *Epochs*. For vehicles departing within a single Epoch, the arrival times to their destination are used to estimate the time period, known as the *Projection Period*, in which the domain for the TDSP algorithm is defined for path set update for vehicles

![Figure 2: The MIVA computational scheme implemented within the SBDTA algorithmic framework.](image-url)
departing within the current Epoch. At the end of one Epoch, all TDSP and assignment-related memory is de-allocated, and then re-allocated for the next Epoch. The MIVA scheme then slides the path set update and adjustment operations from one Epoch to the next until completing all Epochs. As a result, the memory usage during the entire path set update and adjustment operation is only a function of the Epoch length.

2.1.1 Epoch

The Epoch is the partitioned period that acts as the temporal segment for the path adjustment procedure, meaning the TDSP and assignment procedure is bounded solely by the length of the Epoch. An Epoch consists of multiple assignment intervals (interchangeably termed as departure intervals as assignment is performed for vehicle departing at the same departure interval). An aggregation interval pertains to the time interval in which traffic data, i.e., time-dependent link travel times and intersection delays, are averaged to be the input for the TDSP. The assignment interval is bounded by the number of simulation intervals. A simulation interval is defined as the time resolution that traffic simulation states are updated. An assignment interval is a multiple of simulation intervals, and, in the same manner, an Epoch is a multiple of assignment intervals, as shown in Figure 3.

Let \( H \) be the simulation period and \( h \) be the length of the assignment/departure time interval in which \( H \) is discretized resulting in a time discrete model. Let \( T = \{ \tau^1, \tau^2, \ldots, \tau^{H/h} \} \) be the set of departure time intervals. Let the length of each Epoch (number of assignment intervals) be in terms of the integer number of assignment intervals \( b = H/h/n \) where \( n \) is the pre-specified total number of Epochs within \( H \). Let \( E = \{ e^1, e^2, \ldots, e^n \} \) be the set of Epochs. Let \( e^s = \{ \tau^{(s-1)b+1}, \tau^{(s-1)b+2}, \ldots, \tau^{sb} \} \), \( \forall \, e^s \in E \) be the set of assignment intervals for Epoch \( e^s \) containing \( b \) number of assignment intervals. In each Epoch domain there is a set of departing vehicles \( V^e(i,j,\tau) \subseteq V \), where \( V = \{ v^1, v^2, \ldots, v^{|V|} \} \). Those vehicles \( v \in V^e(i,j,\tau) \) are assigned based on the TDSP solved over the Projection Period \( P(e^s) \): Time-Dependent Shortest Path.

![Figure 3: The Epoch and Projection Period of the MIVA structure.](image-url)
Period associated with $e^i$, which will be further explained in the next section.

2.1.2 Projection Period

The Projection Period, $P(e^i)$ is defined as the set of assignment intervals for each Epoch $e^i$. Let $P(e^i) = \{ \tau^{(s-1)b+1}, \tau^{(s-1)b+2}, \ldots, \tau^b, \tau^{b+1}, \ldots, \tau^{b+y} \}$, $\forall e^i \in E$ be the set of assignment intervals contained in the Project Period for Epoch $e^i$. By definition, the start of the Projection Period is synchronized with each associated Epoch. However, the Projection Period is extended beyond the end of the Epoch by $\{ \tau^{b+1}, \ldots, \tau^{b+y} \}$ as shown in Figure 3. This temporal extension is to allow the TDSP to solve for the later arrival times of those vehicles departing toward the end of the Epoch.

It is intuitive to set the limit of the Projection Period based on the latest arrival time of all vehicles departing within the Epoch, as beyond this limit, link travel times and intersection delays would not be needed for the current Epoch’s TDSP calculation. However, binding the Project Period limit based on the latest arrival time may be too conservative and thus include too many additional assignment intervals if the latest arriving vehicle’s travel time is likely to improve in the next iteration because (1) it is assigned with a new path, and/or (2) other vehicles assigned with a better path would also improve the overall traffic condition and improve all vehicles’ travel times. With this being recognized, the length of the Projection Period can be defined as a ratio of total vehicles based on ranked experienced arrival times. This means vehicles belonging to $V^e(i,j,\tau)$ are sorted by increasing experienced arrival times. The Projection Period length is then defined based on a predefined ratio. For example, a 0.95 means that the end of the Projection Period is set at the 95th percentile of all arrival times. In other words, $P(e^i) = \{ \tau^{(s-1)b+1}, \tau^{(s-1)b+2}, \ldots, \tau^b, \tau^{b+1}, \ldots, \tau^{b+y} \}$, where $h \cdot \tau^{b+y} \geq G(\phi)$, where the increasing arrival time profile and is the predefined ratio, $0.0 \leq \phi \leq 1.0$. One should also expect that $P(e^i)$ may vary due to different levels of congestion experienced by vehicles in different Epochs. Vehicles departing in an Epoch corresponding to off-peak hours would have a shorter Projection Period than those corresponding to peak hours due to more severe congestion during peak hours even though the Epoch lengths are identical.

After solving the TDSP, the path becomes available over the domain $P(e^i) = \{ \tau^{(s-1)b+1}, \tau^{(s-1)b+2}, \ldots, \tau^b, \tau^{b+1}, \ldots, \tau^{b+y} \}$. The computational requirement to maintain $V^e(i,j,\tau)$ in memory rather than for $V$ is the major advantage of the MIVA computational scheme. Memory is only allocated to the TDSP of size $P(e)$ rather than the entire simulation period. For the given $V^e(i,j,\tau)$ set, the path set adjustment, i.e., the DTA solution algorithm procedure, is performed which is bounded by a given $e$. Once path adjustment is completed $\forall v \in V^e(i,j,\tau)$, the traffic data and TDSP calculations for the current Epoch $e$ are de-allocated from memory, and the MIVA scheme continues on to the next Epoch.

This staging process may be found seemingly similar to the rolling horizon (RH) methodology such as what was proposed by Baker (1977). While many applications such as economic decision making (Sethi and Sorger 1991) and production scheduling (Ovacik and Uzsoy 1994) have utilized such rolling horizon methods, this type of methodology has been applied to transportation-related topics such as traffic signal control (Newell 1998) and real-time DTA applications (Peeta and Mahmassani 1995;
Ran, Lee et al. 2002), in which “forecasting future condition” is needed in the modeling of the real-time contexts. At each roll period (with current reliable network condition known) the assignment solution found from the roll period is implemented in the real world while future network condition continues to be forecasted for the next stage.

Although semantically similar to the rolling horizon application in prior DTA literature, the MIVA computational scheme has fundamental distinction. MIVA is not about predicting future condition to solve for the current solution, but temporal decomposition of the entire DTA problem into isochronal periods. Forecasting is irrelevant in the DTA solution algorithmic design standpoint where link travel times and node penalties are known for the entire analysis period after simulation.

The following describes the MIVA algorithmic structure within the SBDSA framework:

---

**Step 0** Initialization:

- Iteration count, \( l = 0 \)
- Given \( n: b \) is integer, determine \( E \)
- Initiate SBDSA algorithm
  - Time-varying OD demand tables
  - Instantaneous shortest paths
  - All-or-nothing path assignments

**Step 1** Network Simulation:

- Execute network simulation
  - Acquire resulting time-dependent link and path travel times

**Step 2** Check Convergence:

- If Convergence criteria < \( \epsilon \), then:
  - Stop, Exit SBDSA algorithm
- Else:
  - \( l = l + 1 \)
  - Continue to Step 3

**Step 3** MIVA Preparation

- De-allocate all simulation-related data structures except those used by MIVA and assignment

**Step 4** MIVA Enter:

- For \( e \in E \),
  - Determine \( V^*(i,j,\tau) \) and allocate memory based on
  - Sort \( V^*(i,j,\tau) \) by arrival time and obtain \( G(\cdot) \)
  - Establish \( P(\epsilon) \) based on \( G \), set percentile, and \( \phi \), from \( V^*(i,j,\tau) \)

---

Figure 4: The Pseudo code of the MIVA Scheme. *(continued overleaf)*
3. NUMERICAL ANALYSIS OF MIVA COMPUTATIONAL PERFORMANCE

The MIVA scheme was implemented in a SBDTA model DynusT (Chiu, Nava et al. 2010). Three networks, as shown in Figure 5, were used for the evaluation of the computational performance of the MIVA scheme. Note that all networks for this batch of performance runs were done on a shared-memory computer with a 2.80 GHz Intel Pentium D Dual-Core Processor with 4GB of RAM. Comparisons for each network were benchmarked by simulating each network without the MIVA scheme implemented (termed full-scale). The goal of the comparison was to investigate the computational memory efficiency that can be achieved by the MIVA scheme. There are numerous choices of an Epoch set size since \( e^2 \) can be of various equal number of assignment intervals; therefore, \( n \) (number of Epochs in Epoch set \( E \)) can be of different sizes. For each network, different Epoch sets were used to test the MIVA scheme, and described below for each network.

The Fort Worth, TX in Figure 5(a) contains 13 zones, 180 nodes, 445 links, and an assignment interval length of 2 minutes. 165,276 vehicles were generated for the simulation period of 300 minutes. The Epoch set sizes used for this network were \( n = \{2,3,4,6\} \). The Guam network in Figure 5(b) contains 157 zones, 540 nodes, 1183 links, and an assignment interval length of 5 minutes. 70,088 vehicles were generated for the simulation period of 120 minutes. The network is relatively sparse with only local highway and arterial links giving very limited routes choice for network assignment. The Epoch set sizes used for this network were \( n = \{2,3,4,5\} \). Lastly, the Minneapolis, MN network in Figure 5(c) contains 558 zones, 2837 nodes, 6872 links, and an assignment interval length of 10 minutes. 1,259,594 vehicles were generated for the simulation period of 300 minutes. The network contains 2 major lateral interstates of 394 and 694, while majority of interstates 494, 94 and 35W travel vertically. This network provided adequate connectivity to search for many possible alternative routing. The Epoch set sizes used for this network were \( n = \{3,5,6\} \).
3.1 Convergence Quality

Figure 6 displays the Relative Gap convergence performance for different numbers of Epoch sets in comparison to full-scale assignment. The relative gap value for iteration can be expressed as Eq. (1) below in which $u_{i,j}^{e}$ is the minimal travel time for the origin-destination-departure time triplet $(i,j,\tau)$, $r_{i,j}^{e}$ is the number of vehicles traveling for triplet $(i,j,\tau)$, and $q_v$ is the experienced travel time for vehicle $v$.

$$RG^i = \frac{\sum_{i,j,\tau,k} \sum_{v \in V(i,j,\tau,k)} q_v - \sum_{i,j,\tau}(r_{i,j}^{e} \cdot u_{i,j}^{e})}{\sum_{i,j,\tau}(r_{i,j}^{e} \cdot u_{i,j}^{e})}$$ (1)

The patterns of convergence appear similar across all networks. For the Fort Worth network, the initial RG values range from 100% - 160% for different Epochs, but scenarios with different Epoch setting maintain similar convergence patterns and all reach less than 2% convergence at the final iteration. The RG values for the Guam network starts at about 12% and most Epoch scenarios reach 5% level by the 10th
iteration. Most of them reach 4% at the final iteration except that the full-scale scenario maintains a slight better RG value at about 3%. For the Minneapolis dataset, all Epoch scenarios follow very similar convergence pattern from initial to the final convergence at about 5%.

An important conclusion can be understood from these experiments: different Epoch settings do not degrade the convergence performance as compared to the full-scale assignment. This is due to the MIVA projection period as the time period provides adequate “look-ahead” of time-varying link travel time and node penalties for TDSP computation, and is comparable to the same TDSP calculation of a full-scale assignment.

Figure 6: Convergence Performance ($\varphi = 0.9$) (continued over)
3.2 Peak Memory Usage

Table 1 gives the peak memory usage of MIVA of each network based on the Epoch set sizes $n$ comparing to the full-scale case. For each network, each column represents a different Projection Period percentile. Each row block (e.g. Full Scale, 2 Epochs, etc...) represents each Epoch set $E$ that was tested. The peak memory usage was reported for each Epoch within a single simulation. For example, for the Fort Worth network, the full scale case required 90.0 megabyte (MB) peak memory usage. For a 2-Epoch case, the peak memory usage dropped from 82.0 to 77.0 MB. As expected, lower percentile $\psi$ leads to a shorter Projection Period length and less memory usage. For the 3-Epoch case, the overall peak memory usage continued to reduce to the range of 78 to 73.6 MB. The lowest memory usage was observed for the 1st Epoch in the $\psi = 0.7$ and 5-Epoch case. Comparing the highest and the lowest memory usage, the maximum memory savings was 24.2%.

The Guam network, the peak memory for the full-scale case was 70.2 MB. The memory usage continued to drop as low as 50.9 MB with the increasing number of Epochs and lower percentile value. The maximum memory savings was about 27.0%. Consistent memory savings was also observed for the Minneapolis dataset in which the maximum peak memory usage occurred at the full-scale scenario with 598.6 MB and the lowest memory usage was 430.6 MB in the 6-Epoch case. The maximum memory savings was about 28.2%.

From the results, it becomes apparent that memory saving always increases with increasing number of Epochs. This is a desirable computational property as a model user can always adjust the number of Epoch according to the memory limitation of the intended computing environment. This accomplished the primarily goal of designing a memory-scalable SBDTA solution algorithm.

![Figure 6](continued from previous page)
Table 1: Peak Memory Usage (MB) for MIVA Benchmarking

<table>
<thead>
<tr>
<th>Epoch Set</th>
<th>Fort Worth</th>
<th>Guam</th>
<th>Minneapolis</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>77.0</td>
<td>78.1</td>
<td>467.3</td>
</tr>
<tr>
<td>0.8</td>
<td>77.1</td>
<td>78.2</td>
<td>485.3</td>
</tr>
<tr>
<td>0.9</td>
<td>77.7</td>
<td>78.8</td>
<td>476.3</td>
</tr>
<tr>
<td>1.0</td>
<td>90.0</td>
<td>77.0</td>
<td>598.6</td>
</tr>
</tbody>
</table>

Full-Scale

<table>
<thead>
<tr>
<th>Epoch Set</th>
<th>Fort Worth</th>
<th>Guam</th>
<th>Minneapolis</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Epochs</td>
<td>77.0</td>
<td>78.1</td>
<td>467.3</td>
</tr>
<tr>
<td></td>
<td>77.1</td>
<td>78.2</td>
<td>485.3</td>
</tr>
<tr>
<td></td>
<td>77.7</td>
<td>78.8</td>
<td>476.3</td>
</tr>
<tr>
<td></td>
<td>90.0</td>
<td>77.0</td>
<td>598.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch Set</th>
<th>Fort Worth</th>
<th>Guam</th>
<th>Minneapolis</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Epochs</td>
<td>73.6</td>
<td>73.8</td>
<td>434.7</td>
</tr>
<tr>
<td></td>
<td>74.1</td>
<td>74.9</td>
<td>462.3</td>
</tr>
<tr>
<td></td>
<td>73.1</td>
<td>74.9</td>
<td>455.3</td>
</tr>
<tr>
<td></td>
<td>74.0</td>
<td>73.1</td>
<td>450.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch Set</th>
<th>Fort Worth</th>
<th>Guam</th>
<th>Minneapolis</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Epochs</td>
<td>70.8</td>
<td>72.5</td>
<td>430.6</td>
</tr>
<tr>
<td></td>
<td>71.2</td>
<td>73.2</td>
<td>445.3</td>
</tr>
<tr>
<td></td>
<td>71.5</td>
<td>73.8</td>
<td>458.6</td>
</tr>
<tr>
<td></td>
<td>71.5</td>
<td>73.3</td>
<td>454.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch Set</th>
<th>Fort Worth</th>
<th>Guam</th>
<th>Minneapolis</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Epochs</td>
<td>68.2</td>
<td>69.4</td>
<td>443.3</td>
</tr>
<tr>
<td></td>
<td>70.5</td>
<td>71.0</td>
<td>453.2</td>
</tr>
<tr>
<td></td>
<td>71.2</td>
<td>71.2</td>
<td>445.4</td>
</tr>
<tr>
<td></td>
<td>71.5</td>
<td>71.4</td>
<td>51.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch Set</th>
<th>Fort Worth</th>
<th>Guam</th>
<th>Minneapolis</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Epochs</td>
<td>50.9</td>
<td>51.0</td>
<td>51.2</td>
</tr>
</tbody>
</table>

Projection Period Pct (φ)
3.3 Run-Time Performance

The run-time performance of the proposed MIVA procedure can be understood from both analytical and numerical standpoints. Analytically, the run-time can be expressed as:

\[ TT = \Omega\left\{ nTDSP|N||M|\left(\phi(b+k)^2\right) + n\lambda + |V|\mu \right\} \]

where \( b = H/h/n \) is number of assignment intervals in an Epoch as expressed in Section 2.1.1. \( k \) is the average travel time in terms of assignment intervals. The term \( b+k \) approximates the project period length. \( \phi \) is the ratio of the Project Period for TDSP calculation. \( H \) is the simulation period, \( h \) is the length of assignment interval and \( n \) is the number of Epochs, \( \lambda \) is the fixed amount of computational overhead for each Epoch such as memory allocation and de-allocation. \( |V| \) is the total number of vehicles and the time needed for vehicle assignment is independent of the Epoch specification. Taking the first-order derivative \( TT \) equal to zero, with respect to \( n \), the following is obtained:

\[ n^* \approx \frac{H}{h \left( k^2 + \frac{\lambda}{\phi^2|N||M|} \right)^{1/2}} \]  

Eq. (2) is an analytical approximation of the time-optimal number of Epoch, which indicates that the optimal value is dependent upon multiple factors such as size of the network, simulation period, congestion level, assignment interval, and the Epoch specific overhead. The formulation generally indicates that a less number of Epochs would run faster for higher congestion, overhead, and longer assignment intervals. Given the complex relationship between these factors in determining the optimal Epoch number, extensive numerical experiments were also performed to provide further insights on the computational performance.

Table 2 displays the recorded CPU Time for each network, as the columns represent different Projection Period ratios, and each row represents the size of the Epoch set.

For the Fort Worth network, it is noted that the run-time for the full-scale case is 590 seconds for 100 iterations. For all Epoch settings, the run-time remains lower than 590 seconds for all ratios setting except \( \phi=1.0 \). For the Guam network at 50 iterations, the 2-Epoch case with \( \phi=0.7 \) achieved 1390 second, which was lower than the full-scale case; however, at worst, the run-time for the 6-Epoch case with \( \phi=1.0 \) was found at 2,679 seconds, a 91% increase from the full-scale case. In the Minneapolis, MN network at 50 iterations, the memory usage for the full-scale case was 50,813 seconds. Most of the cases were worse than the full-scale case, except the 5-Epoch case with \( \phi=0.8 \) with 49,413 seconds, a 2.7% savings. The worse case, being the 5-Epoch case with \( \phi=0.9 \), was 10.9% higher than the full-scale case with 56,354 seconds.

Cross-referencing the results shown in Table 1 and Table 2, one can observe that for a given network, simulation period, assignment interval, and total number of vehicles, a scenario yielding faster time compared with the full-scale benchmark case does exist, although the majority of other cases underperform the full-scale case. This phenomenon indicates that the memory saving generally comes with a price of worsening computational speed, which is consistent with the Time-Memory Tradeoff (TMTO) situation discussed in the computer algorithm design literature (Hellman 1980; Borst, Preneel et al. 1998).
Given Eq. (2) in which several factors give rise to complex relationship for the optimal Epoch setting, developing a run-time self-tuning algorithm to move toward the run-time optimal Epoch setting over assignment iterations is intuitively more adaptive and robust than analytically computing the optimal Epoch \textit{a priori}. The next section presents such an algorithm.

4. SELF-TUNING ADAPTIVE ALGORITHM FOR OPTIMAL EPOCH SETTING

During the DTA iteration set, the proposed online, self-tuning adaptive (STA) mechanism determines an optimal size of Epoch set \( n \) from a set of permissible number of \( n \) sizes. This Epoch set of size \( n \) is then applied for the current iteration. The search for a new \( n \) is commenced again at the beginning of the next iteration until no better \( n \) can be found. Determining the optimal Epoch set is done by performing a line search using a “bisection” search method.

4.1 Bisection Search Method

In the bisection search method, the leftmost function value and rightmost function value of the feasible set of function values are examined. The feasible set’s midpoint value is determined while the minimum function value between the leftmost and rightmost value is determined. The feasible set is cut in half at the midpoint value as the feasible set half with the larger evaluated function value is dropped and the value whose function was the minimal and the midpoint value become the new range of the feasible set. This procedure is iterated until the minimum value of the feasible set is determined.

Let \( H \) be the analysis period and the assignment interval length be \( h \); let \( N^l=\{n^1, \ldots, n^l\} \) be the increasing-order of permissible set of feasible \( n \) Epoch set sizes, where \( n^1\leq n^2\ldots\leq n^l \), and let \( \hat{n}=H/h \) be the maximum permissible \( n \) Epoch set size. The first 2 iterations of the simulation are used to determine the initial direction of the search. At iteration \( l=1 \), the first \( n \) value \( n^l \) (being the leftmost set value of the \( N \) set: \( n^j=\hat{n}(l) \)) and records the run-time \( f[\hat{n}(l)] \). Next, at iteration \( l=2 \), the rightmost set value,
\( \hat{n}(l) \), of the \( N \) set is \( n^h \), and the run-time \( f[\hat{n}(l)] \) is recorded. The bisection method takes the 2 values \( \hat{n}(l) \) and \( n^\ell(l) \), evaluates the respective run-times. The algorithm splits set \( N^l \) into two sets - eliminated set \( \bar{N} \) and retained set \( N^\ell \), where \( \bar{N} \) contains the one of the two set range values with higher run-time and \( N^\ell \) contains the value with the lower run-time and \( \bar{N} \cup N^\ell = N \). From the retained set \( N^\ell \), the value \( n^\ell \), being the midpoint of the current \( N \) set, is now the new range value including the determined minimum value. That is, the value \( n^\ell \) is assigned as one of the two extreme values for the next iteration \( l+1 \) in \( N^\ell \) as \( N^\ell = N^{l+1} \), depending on which of the two following conditions are met: \( \hat{n}(l+1) = n^\ell \) if \( f[\hat{n}(l)] > f[\hat{n}(l)] \) or \( \hat{n}(l+1) = n^h \) if \( f[\hat{n}(l)] < f[\hat{n}(l)] \).

If \( |N| \) is an odd number when cut, then either the ceiling value of \( n^\ell \) is taken if \( f[\hat{n}(l)] > f[\hat{n}(l)] \) or the floor value of \( n^\ell \) is taken if \( f[\hat{n}(l)] < f[\hat{n}(l)] \) is taken. The algorithm will iteratively evaluate the new \( N^{l+1} \) until only a single value is left, thereby determining the optimal Epoch set with value setting \( \hat{n} \).

The following describes the STA-MIVA algorithm for the SBDTA procedure:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>Initialization:</strong></td>
</tr>
<tr>
<td></td>
<td>Iteration count, ( l = 0 ), Determine ( N^l, n^h, n^\ell )</td>
</tr>
<tr>
<td></td>
<td>Initiate SBDTA algorithm</td>
</tr>
<tr>
<td></td>
<td>Time-varying OD demand tables</td>
</tr>
<tr>
<td></td>
<td>Instantaneous shortest paths</td>
</tr>
<tr>
<td></td>
<td>All-or-nothing path assignments</td>
</tr>
<tr>
<td>1</td>
<td><strong>Network Simulation:</strong></td>
</tr>
<tr>
<td></td>
<td>Execute network simulation</td>
</tr>
<tr>
<td></td>
<td>Acquire resulting time-dependent link and path travel times</td>
</tr>
<tr>
<td>2</td>
<td><strong>Check Convergence:</strong></td>
</tr>
<tr>
<td></td>
<td>If Convergence criteria ( &lt; \varepsilon ), then:</td>
</tr>
<tr>
<td></td>
<td>Stop; Exit SBDTA algorithm</td>
</tr>
<tr>
<td></td>
<td>Else:</td>
</tr>
<tr>
<td></td>
<td>( l = l + 1 )</td>
</tr>
<tr>
<td></td>
<td>Continue to Step 3</td>
</tr>
<tr>
<td>3</td>
<td><strong>STA Mechanism</strong></td>
</tr>
<tr>
<td></td>
<td>If (</td>
</tr>
<tr>
<td></td>
<td>( n^\ell = n(l) \in \bar{N} )</td>
</tr>
<tr>
<td></td>
<td>Else:</td>
</tr>
<tr>
<td></td>
<td>If ( l = 1 ), then:</td>
</tr>
<tr>
<td></td>
<td>( n(l) = n^h )</td>
</tr>
<tr>
<td></td>
<td>( n(l) \in \bar{N}, n(l) \notin N^\ell )</td>
</tr>
<tr>
<td></td>
<td>Else if ( l = 2 ), then:</td>
</tr>
<tr>
<td></td>
<td>( n(l) = n^h )</td>
</tr>
<tr>
<td></td>
<td>( n(l) \in \bar{N}, n(l) \notin N^\ell )</td>
</tr>
<tr>
<td></td>
<td>Else:</td>
</tr>
<tr>
<td></td>
<td>Bisect ( \bar{N} )</td>
</tr>
<tr>
<td></td>
<td>Evaluate ( f[n(l)] ), Determine ( \bar{N}, N^\ell, n^\ell )</td>
</tr>
<tr>
<td></td>
<td>If ( f[\hat{n}(l)] &gt; f[\hat{n}(l)] ), then:</td>
</tr>
<tr>
<td></td>
<td>( n(l+1) = n(l + 1) = ceil(n^\ell) )</td>
</tr>
<tr>
<td></td>
<td>( \bar{N} = { n(l+1), \ldots, \hat{n}(l) } )</td>
</tr>
<tr>
<td></td>
<td>( n(l+1) \in \bar{N}, n(l+1) \notin \bar{N} )</td>
</tr>
<tr>
<td></td>
<td>Else if ( f[\hat{n}(l)] &lt; f[\hat{n}(l)] ), then:</td>
</tr>
<tr>
<td></td>
<td>( n(l+1) = \hat{n}(l + 1) = floor(n^\ell) )</td>
</tr>
<tr>
<td></td>
<td>( \bar{N} = { n(l), \ldots, \hat{n}(l + 1) } )</td>
</tr>
<tr>
<td></td>
<td>( n(l+1) \in \bar{N}, n(l+1) \notin \bar{N} )</td>
</tr>
</tbody>
</table>

**Figure 7:** The Pseudo code of the STA-MIVA Algorithm (continued overleaf).
For a long simulation period, $|N|$ may be quite large and may not be sensible to evaluate both extreme values of the $N$ set, knowing that each value will not be used (e.g. $n=1$; $n=72$). Therefore, the STA-MIVA algorithm will select a subset $N=\{n, \ldots, n\}$ where $n$ is the leftmost value of the subset ($1<n<n$) and $n$ is the rightmost value of the subset ($n<n<n$). By doing so, this will reduce the number of iterations spent evaluating various sizes of $n$ Epoch sets and find a reasonable value setting relatively quickly.

4.2 Computational Performance

Two large-scale, real-world networks, each with an analysis period of 24 hours, were used to the run-time performance of the STA algorithm. The two networks, illustrated in Figure 8, were compared to the full-scale SBDTA. The first network is El Paso, TX network which contains 681 zones, 2,437 nodes and 5,233 links. The aggregation

Figure 7: (continued from previous page)

Figure 8: Networks for STA-MIVA testing.
interval set for the network was 10 minutes. With a 24-hr simulation, the number of generated vehicles was about 2.1 million. A total of 15 iterations were specified. The other network was the DRCOG (Denver Regional Council of Governments) network, which consists of 2,832 zones, 10,095 links and 23,147 links. The aggregation interval was set to be 15 minutes. Approximately 6.8 million vehicles were generated within the 24-hour analysis period.

Both networks were run on a shared-memory computer with two 2.4 GHz Quad-Core AMD Opteron 2378 processors with 32 GB of RAM.

4.2.1 Optimal Epoch search

As shown in Figure 9(a), the set of permissible Epoch sets for the El Paso network contains 9 possible Epochs. In the 1st iteration, the STA algorithm selects 4 Epochs and evaluated the run-time to be 0.64 hours. In the 2nd iteration, the next Epoch set was 36 Epochs, and the run-time recorded to be 0.625 hours. Since the 36-Epoch setting resulted in a shorter run-time, the set $N$ was cut in half with the value determined to be cut half was $n=12$. The half containing the 4-Epoch setting, $\tilde{N}$ was eliminated, while the set $N$ half containing the 36-Epoch setting, $\tilde{N}$, was now the new $N$ set for the next iteration, $\tilde{N} = N^{i+1}$. The new extreme values for the new set $N^{i+1}$ were now $n = \bar{n} = 12$ and $\bar{n} = 36$. The iterative process continued until in the 5th iteration that the run time starts to degrade and the optimal Epoch setting was determined to be 18 Epochs found at the 4th iteration, with each Epoch being 80 minutes. The performance gain comparing the best and worse setting was about 10% improvement in run-time.

![Figure 9: El Paso Network](image-url)
There were a total of six permissible $n$ values for Epoch set for the DRCOG network. The search pattern differed from the El Paso network, but still the results clearly demonstrated the STA algorithm’s ability to converge to an optimal $n$ value. The marginal savings in run time from the first iteration to the final convergence iteration is significantly demonstrated by saving approximately 1 hour. Comparing to the worse setting, then the execution time savings is about 3.8 hours, representing a 27% improvement. The final Epoch setting value was 6 (240 minute per Epoch).

This testing provided the numerical evidence that the STA algorithm will lead to an optimal Epoch setting that would guarantee to outperform the full-scale scenario in memory usage, and optimal possible run-time.

4.2.2 Peak memory usage

The memory usage requirements for both El Paso and DRCOG networks are displayed in Figure 10. The El Paso network using the MIVA scheme established a memory savings of almost 77.7% as the STA-MIVA scheme took only 871MB and the full-scale assignment required 3.9 GB of memory. The peak memory requirement for the STA-MIVA algorithm was even less than the memory requirement of simulation (as the simulation module for this SBDTA application is multi-threaded using 8 CPUs during the simulation procedure for both El Paso and DRCOG networks). Similarly, the DRCOG network using the MIVA scheme demonstrated a memory usage reduction of 79.2% when compared with the 5.4 GB usage for the STA-MIVA and 25.9 GB for the full-scale case. The memory savings results re-affirm the superior performance of STA-MIVA from both run-time and memory standpoints.
4.2.3 Run-time performance

The iterative computational run-time for both STA-MIVA and the full-scale case for both El Paso and DRCOG networks are depicted in Figure 11(a) and (b). From the beginning, the El Paso network results in Figure 11(a) demonstrated quicker run-time even with the Epoch setting changing within the first 4 iterations as discussed in section 4.2.3. (a) El Paso Network

(b) DRCOG Network

Figure 10: Peak Memory Usage for El Paso and DRCOG comparing the STA-MIVA Algorithm and Full-Scale Case.
4.2.1. The STA-MIVA algorithm reported a 4.66 hour savings in the run-time at 11.43 total hours for the 15 iterations; this is a 30% time saving compared to the full-scale case. Also evident from the DRCOG network was the run-time savings with the STA-MIVA algorithm was implemented. The total run-time for the STA-MIVA algorithm was 118.91 hours. However, compared to the full-scale assignment, the run-time was 175.29 hours; a 32.1% computational run-time improvement was obtained.

Figure 11: The cumulative run-time savings with STA-MIVA Algorithm.
5. CONCLUSIONS
The MIV A computational scheme provides a robust treatment of the temporal domain issue in long-term analysis periods by introducing two time periods known as **Epoch** and **Projection Period**. Epoch splits the simulation period into smaller, more manageable stages. The Projection Period is the time interval which provides the most minimal sufficient travel time data to calculate the TDSP efficiently.

Experimental results demonstrate the potential memory savings from implementing decoupling technique. However, decoupling the SBDTA procedure too much will outweigh potential benefits of using MIVA due to overhead of the TDSP algorithm. To determine a feasible Epoch setting value without empirical analysis of multiple Epoch value settings separately, a self-tuning adaptive algorithm was introduced in which an optimal Epoch set value was determined in an efficient manner. Numerical analysis results concluded that the STA-MIVA provided an efficient way to determine an optimal Epoch set and demonstrated considerably acceptable and proficient results in downsizing the memory requirements for large-scale, long-term roadway networks.

Apparently, the TDSP and assignment procedure for each Epoch can be bundled and distributed in parallel to different compute nodes. This strategy will increase the memory usage if this is applied to share-memory architecture; however, if the work is distributed to the message-passing system then memory usage is distributed.

Overall, the proposed technique MIVA, with the self-tuning mechanism, demonstrates a promising algorithmic strategy to confront the bottlenecking issue brought by the use of SBDTA in long-period, large-network analysis by improving computational efficiency and still maintains the quality of the DUE solution.

6. REFERENCES


