On Scaling Time Dependent Shortest Path Computations for Dynamic Traffic Assignment

Amit Gupta, Weijia Xu
Texas Advanced Computing Center,
University of Texas at Austin, Austin, TX ,USA
Email: { agupta, xwj }@tacc.utexas.edu

Kenneth Perrine, Dennis Bell, Natalia Ruiz-Juri
Network Modeling Center,
University of Texas at Austin, Austin, TX ,USA
Email: { kperrine, dennis.bell }@utexas.edu, nruizjuri@mail.utexas.edu

***Abstract*—Dynamic Traffic Assignment (DTA) models provide a powerful tool to realistically represent the complex interactions between travelers and the transportation infrastructure in large regions, and they have been increasingly adopted by transportation network planners and operators in the last decade. Time dependent shortest path (TDSP) calculations at the core of most DTA methodologies usually require storing and comparing millions of discovered paths. This makes the problem I/O intensive in addition to it inherently being computationally demanding. In this paper we present a use case on scaling up the TDSP calculations within an established existing DTA software framework with distributed computing. Our approach alleviates I/O bottlenecks by using RAM disks and improves a label correcting shortest path algorithm by using priority queues which also leads to better workload balancing among parallel processes. Tests with real-world transportation networks show drastic run time performance improvements, in some cases by a factor of 12x. This suggests that our methodology enables the analysis of much larger networks. Furthermore, the improvements were achieved with relatively minor modifications to the base code, which makes this approach appealing for the enhancement of other existing DTA implementations.**

I. Introduction

Dynamic Traffic Assignment (DTA) models are powerful tools for realistically representing complex transportation networks of urban regions involving millions of daily interactions amongst travelers and between travelers and the infrastructure. Simulation based DTA models *e.g.,* [1], appropriately account for the impact of traffic control strategies, the propagation of congestion, and the presence of tolls and turning movement delays, and have seen increasing adoption by transportation network planners and operators in the last decade. A crucial component in DTA models is the computation of time-dependent shortest-paths (TDSP) [2]. With increasing volume and complexity of input data from large real-world networks, TDSP may often become the bottleneck in DTA implementations.

The structure of most DTA models used in practice are described by Chiu et al. [3] (depicted in Figure 1). The optimal path between an origin-destination (O/D) pair depends on prevalent traffic conditions and the dynamic nature of travel time and costs associated with corresponding road infrastructure. TDSPs are therefore used instead of regular shortest-path algorithms and their complexity depends on the granularity at which edge-cost variations are considered.

In DTA, the shortest paths computations are required for all O/D pairs, for each vehicle type (due to their differing constraints on infrastructure use *e.g.,* tolls, road prohibitions) and at each timing step during the simulation. The iterative nature of the DTA process can make TDSP a significant component of its computational cost. The shortest paths found are used in DTA to make subsequent vehicle routing decisions (called the "Assignment"). This requires storage of these paths for future access and makes an already computationally demanding process also I/O intensive.

To improve the scalability of DTA applications, our approach parallelizes the TDSP algorithm by a hybrid application model that uses both MPI processes and Linux Pthreads, to utilize all available cores across multiple compute nodes. We utilize a RAM disk (a local memory based filesystem) in place of a disk-based parallel file system to alleviate I/O bottlenecks. Additionally our proposed improvement to a popular label-correcting TDSP algorithm evaluates network nodes in dynamic order prioritized by label costs as the road network is explored and updated. We applied our approach on various road networks in the 6-county Austin, TX region and show (in Section IV) significant speedups, 6x to 12x, in the run time of TDSP and consequently of the DTA application. Our methodological improvements can be used to boost the scalability of other simulation-based DTA applications that utilize label-correcting TDSP implementations.

II. Background

DTA is a family of methodologies branching from the work of Merchant and Nemhauser [4] for modeling the performance of a regional traffic network for a given travel demand pattern. DTA differs from traditional regional modeling approaches in that it considers the temporal variation of traffic conditions, as well as the limited capacity of roadways, the propagation of traffic congestion, and the explicit impact of traffic control strategies [3]. DTA methodologies typically run stages of Simulation, TDSP and Assignment

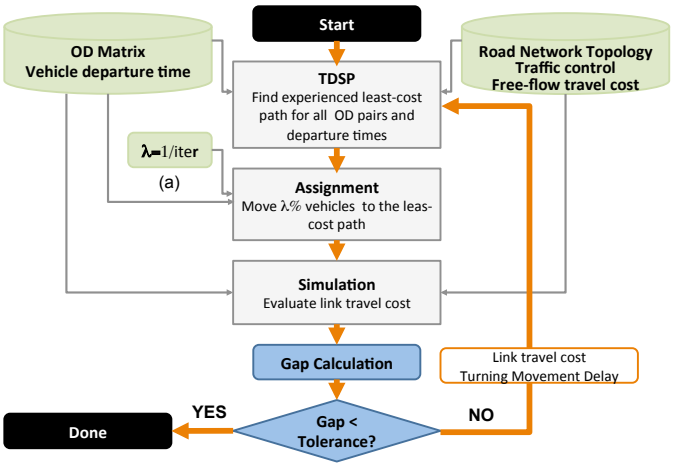

Figure 1. Basic execution flow of a DTA algorithm.

iteratively as shown in Figure 1 until an equilibrium is reached. (See Peeta and Ziliaskopoulos [5] for a survey of DTA research.)

Shortest path problems have been vastly studied in the transportation literature [6]. Efficient solution approaches include label setting and label correcting algorithms, which find optimal paths from one origin to all destinations (or vice versa) in $O(n \cdot log(n))$ and $O(n \cdot m)$ respectively for selected implementations. The dynamic version of the shortest path problem, TDSP, considers edges with time-varying cost (e.g. [2]). These algorithms account for observed changes in traffic conditions as a driver moves through the network. When dealing with a large geographic region, the substantial data storage and computation requirements for finding experienced travel times in path-finding have only been feasible with recent technology (Nezamuddin, [7]).

A large body of literature deals with different TDSP variations and efficient solution algorithms. Dreyfus [8] showed that label correcting algorithms can be extended to solve TDSP. Ziliaskopoulos [9] proposed three options to parallelize a dequeue implementation of the modified label correcting algorithm [10]. Among those, a simple framework distributing origin nodes across processors significantly improves the performance of one-to-all (one origin to all destinations) problem instances. Chabini [11] introduced a decreasing-order-of-time (DOT) algorithm to solve the one-to-all TDSP problem which has optimal worst-case complexity. His numerical experiments suggest that the DOT approach outperforms dequeue implementations in practice. Chabini et al [12] implement a distributed version of the DOT algorithm which leads to considerable efficiencies that scale well for very large networks. Chiu et al. [13] propose a different parallelization approach on the spatial and temporal dimensions of the problem, and combine it with a hierarchical TDSP algorithm (see [14]) in order to enable very large scale DTA applications. More recently, [15] focused on distributed computing versions of A*-type TDSP algorithms, which use heuristic methodologies to reduce the computational effort for single origin-destination pairs of interest. Other authors [16] have focused on approaches with massive parallelism to compute all-to-all shortest paths that efficiently handles sparse origin-destination matrices that characterize DTA applications.

Our work is based on the TDSP implementation in VISTA, a transportation modeling framework providing DTA, CTM-based traffic simulation [17] and internet-based reporting [18]. It simulates the movements of multiple modes of transportation across large networks, and incorporate the effects capacity additions, intelligent transportation system (ITS) traveler information systems and the deployment of traffic operation strategies into driver behavior. It is an established software package that has been used in transportation research and practice (e.g. [19] [20]). VISTA is at the core of a practical DTA implementation used by the Network Modeling Center at the University of Texas at Austin to support the modeling efforts of the Capital Area Metropolitan Planning Organization (CAMPO) [21]. Our goal is to enable efficient analysis of larger networks, thus facilitating the integration of DTA in the planning process.

III. Computational Algorithm Overview

VISTA implements a dequeue based dynamic label-correcting algorithm by Zilaskopolos et al [9] of run time complexity $O(T^2 \cdot V^3)$, where T is the number of discrete time intervals for the period simulated and V is the number of nodes in the network. The algorithm computes the time varying least-cost paths for all O/D pairs for which there is demand (usually a subset of the entire set of vertices in the network). The cost on an edge is a user-specified combination of travel time, distance, and monetary costs accommodating variations across vehicle types. TDSPs are computed for several departure times, *e.g.,* every 5 minutes, within the analyzed period that may range from a peak period case (2-3 hours) to an entire day. For each destination the algorithm keeps a vector of labels (one per time interval) at every node in the network. Each label represents the travel time to destination from that node when departing at the start of that time interval. Each label also contains information of nodes adjacent to it that are next on path to the destination. The key tradeoff is between simulation accuracy using more time intervals *i.e.,* larger label vectors, and label maintenance overhead in TDSP.

The inputs to the TDSP algorithm are namely the travel times that vehicles experienced in the simulation. The TDSP algorithm has two major parts:

1) **Update Labels:** This is the label-correcting component which begins at the given destination and works backwards (using the incoming links) until all reachable nodes are labeled. At the end of this process,

```
1: procedure UPDATELABELS(destination)
2:     Start at destination
3:     Initialize labels at destination as ZERO
4:     Initialize labels of all other nodes as INFINITY
5:     Add the destination to Scan-Eligible data structure
6:     repeat
7:         Pop node from Scan-Eligible data structure
8:         for all incoming links this node do
9:             if Current Cost < Existing Label Value then
10:                Update the cost labels of the node
11:                Add node to Scan-Eligible data structure
12:            end if
13:        end for
14:    until Scan-Eligible data structure is empty
15: end procedure
```

Figure 2. The algorithm for updating labels

these labels represent the travel times to the destination from each node at every considered departure time.

2) **SEARCH FOR NEW ROUTES:** Once the labels are made current as described above, new routes are built by tracing back along the lowest cost route from every origin to the corresponding destination using the updated label vectors.

Our profiling indicates that the UPDATE LABELS procedure is the more computationally intensive part of the entire DTA application. As seen in Figure 3, it can approx. 80% of the total application run time. SEARCH FOR NEW ROUTES steps were comparatively less expensive.

The UPDATE LABELS step implements the label correcting algorithm presented by Ahuja et al. [6] adapted to using a vector of labels to model time varying costs. As show in Figure 2, it begins at a destination node, updates its label values, traverses its incoming links to reach the nodes at the preceding end, and again recursively processes those nodes in a similar fashion. As the algorithm proceeds, the interim label values serve as an upper bound on the cost at that point in the algorithm, being "corrected" every time they are updated. The algorithm halts when there are no more nodes left to examine; the final label values indicate the lowest costs at their corresponding time intervals. The "Scan-Eligible data structure" (Figure 2) represents any internal data structure used to hold the interim nodes and to determine the order they will be examined in. The TDSP algorithm in VISTA uses a dequeue implementation, giving priority to nodes that have already been examined.

TDSP work is divided up on a per-destination basis. Since a vector of labels is destination-specific and is independent across destinations, this forms a natural point for parallelization of this algorithm. For scalability, we've adopted a hybrid approach that uses MPI processes in conjunction with Linux Pthreads. The workload is distributed in a two-level round robin fashion. First, destinations are distributed across all the MPI processes spawned (usually one per compute node). Next, each MPI process distributes its pool of destinations across all its threads (usually one per core). Label updating and route finding is distributed, while simulation and route aggregation takes place in the master process.

Network	Nodes	Links	O/D Pairs	Demand (Vehicles)
Downtown Austin	546	1251	3640	63322
South Austin	1784	973	27225	253097
Central Austin	1871	4003	51984	131425
Austin Region AM	11446	23777	4407520	697885
Austin Region PM	11446	23777	4492280	1095899

Table I
TOPOLOGY AND DEMAND STATISTICS FOR TEST NETWORKS USED

IV. EXPERIMENTS, RESULTS & DISCUSSION

A. Hardware, Software Environment & Datasets

Experiments were run on the Stampede supercomputer at the Texas Advanced Computing Center, University of Texas at Austin [22]. Each compute node contains a dual configuration of 8-core Intel Xeon E5-2680 processors and 32 GB of RAM and runs the CentOS 6.5 Linux distribution. The compute nodes are connected via the Mellanox FDR 56 Gb/s InfiniBand network. The MPI implementation uses Intel MPI library (ver 4.1.0.030).

All the roadway networks we have analyzed in this paper (Table I) with VISTA [18] represent some practical reduction of real networks of the Austin, TX region. These reductions serve to simplify the network to focus on either specific sub-regions or on higher-capacity arterial roads. The original networks and their associated data come from CAMPO [21].

B. Effect of I/O Environment

At the end of the TDSP algorithm, a list of shortest paths between all origins to each destination is produced for all time intervals. Depending on the demand, departure times and the dynamics of the simulation, there are certain regions whose traffic conditions (and thereby, the shortest paths it contains) are relatively stable. The highly dynamic parts of the network that see a considerable traffic load, are likely to see a fervent change in the shortest paths of their associated O/D pairs. VISTA records the newly discovered path topologies on disk. The two primary I/O activities are therefore to first determine if a newly found path is unique from those found in previous iterations by scanning through the file and comparing topologies, and second to actually store a new route and read them prior to the next simulation. Our initial attempt was to use a shared Lustre Parallel File System as a means of sharing this data across all nodes concurrently. However the random, small-sized read patterns and their distributed nature significantly slowed down the DTA application. As networks grow in size, the complexity

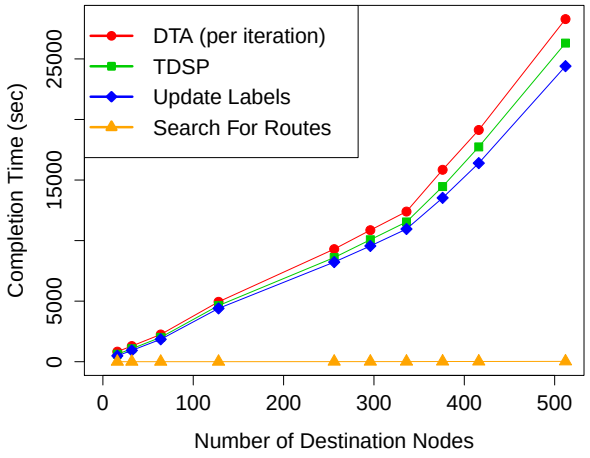

Figure 3. **Austin Region AM**: DTA run time breakdown by application component when scaling up destinations

in simulation dynamics increase and so does the number of shortest paths to be compared. These factors compound the I/O load and contribute to the performance deterioration. For the **South Austin** network, our I/O profiling test indicated as much as 77% of the application run time was being spent in blocking I/O and for the larger networks some of the runs failed to complete in reasonable time (48 hours) even when using upto 8 nodes. We've overcome this problem by moving this I/O activity to a RAM disk, largely eliminating the I/O bottleneck. We make this scale to distributed computations by syncing across the RAM disks of all nodes between iterations. This simple approach has been successful for our real world large networks that we have encountered. We intend to explore a more generic, distributed architecture for efficiently sharing this data across several nodes at low latency in future research.

C. Scalability with Network Sizes

To analyze the effect of increasing the input size on the underlying algorithms in the DTA process, we ran an experiment where we gradually increased a partial set of possible destinations in **Austin Region AM** network. We see in Figure 3 a breakdown of the completion times of TDSP components relative to the DTA application completion time. As we scale up the number of destinations, we first observe a linear increase in application run-time which later becomes exponential. We see that the TDSP component constitutes a large share of application run-time, particularly its UPDATE LABELS portion (*e.g.,* at 512 destinations, it is about 86% of the run-time of the DTA application per iteration). This is therefore the computational bottleneck when scaling DTA applications to larger, more complex networks.

Cores	16	32	64	128	256	512
Factor Improvement in Runtime	6.22	6.84	7.22	7.93	8.64	12.50

Table II
AUSTIN REGION AM: SPEED UP FACTOR WITH CORES USED

D. Improved Algorithm Performance

While the existing dequeue implementation does correctly favor nodes that have been visited in the past, nodes are otherwise evaluated on a FIFO basis. A single label value changing in a vector of labels for a node causes that node to be queued for reexamination by the algorithm. This is because any shortest paths that route through it at that time step will change upstream. Examining this property closer reveals several situations where evaluating nodes in a FIFO order could cause a whole tree of other nodes to be repeatedly updated and sent back into the dequeue for re-examination, thereby drastically slowing down the algorithm (*e.g.,* if a path later circles around a region and changes even a single label of one of the nodes already processed by the dequeue, this in-turn causes the whole tree of descendant nodes to be marked for re-examination). Such update cycles can have devastating effects on the practical run time of this algorithm, moving it towards the theoretical upper bound.

To prevent this we posit that the order nodes are evaluated in *does* matter and that FIFO ordering of the dequeue scheme is not optimal. We propose that if the vector of labels could be quantitatively compared to each other in some sensible manner (perhaps by reduction to a singular weight), this would give us a scheme for favoring exploration of the nodes that lie on lower-cost paths first. This effectively chases down and updates labels on the lower cost paths first, causing path labels to more readily converge to their "correct" values in fewer numbers of update steps. Furthermore, when paths to the origins via the higher-cost nodes are later evaluated, this causes fewer update operations to take place if the label value that already exists there is smaller (by virtue of being reached by a lower cost path earlier). This prevents the cyclic nature of updates that we described earlier and in practice improves run times for this algorithm. This goal can be accomplished by using a priority queue (min-heap) which always favors the lowest-cost node to be the next one explored of all the nodes enqueued thus far.

We've devised a heuristic of using the label value of the first time index—*i.e.,* $T[0]$—as the weight for this node. This label value represents the cost at that node at the start of the time window of the most recent simulation. Our intuition behind this heuristic is that a high cost at the start of a simulation time window can be indicative of a high cost during the next simulation window (*i.e.,* an already congested link is likely to stay congested in the "near" future). This assumption may break down with larger analysis time windows. It is however, a reasonable

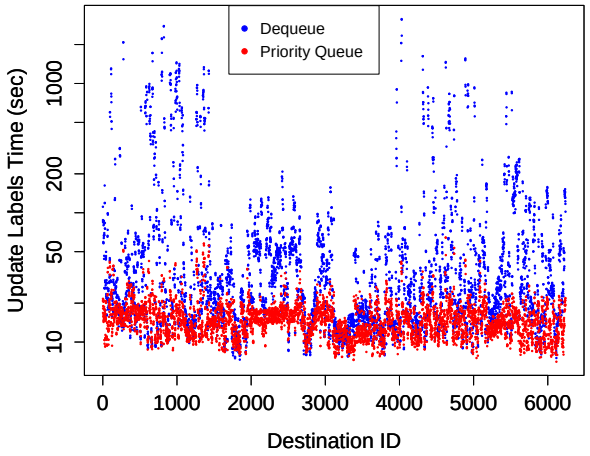

Figure 4. **Austin Region AM**: Distribution of time spent in UPDATE LABELS for each destination by the two queueing methods

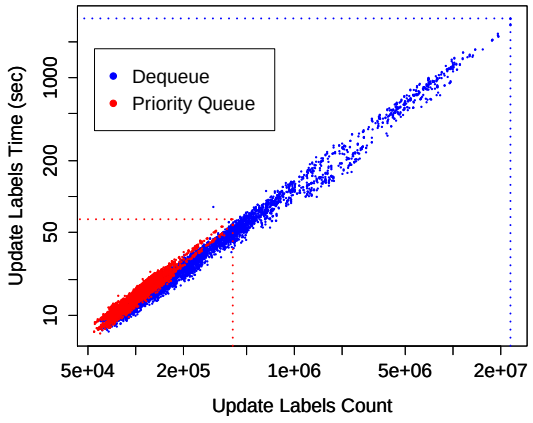

Figure 5. **Austin Region AM**: Linear relationship between time spent in UPDATE LABELS at each destination and the number of recursive calls

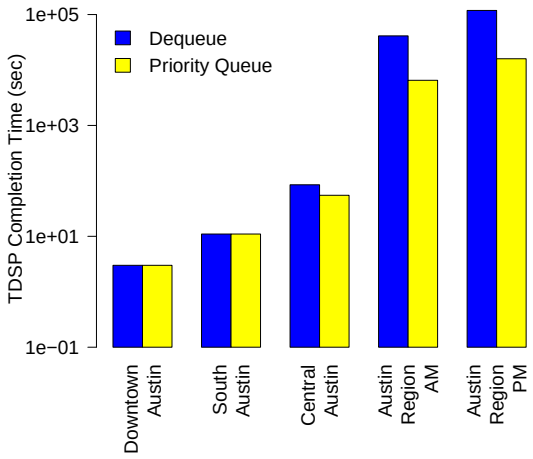

Figure 6. TDSP performance improvements using the Priority Queue across our test networks

assumption during close to peak traffic conditions for shorter time windows, which quite frequently holds true for the scope of many research investigations in this field.

We can see from Figure 4 that when using the Dequeue scheme there are several "straggler" nodes that hold back the entire DTA process and slow down its overall run time drastically. For the Dequeue scheme the mean, median and maximum UPDATE LABELS completion times observed were $78.85s$, $23.39s$ and $3148s$. When using the Priority queue scheme however, we find that the times are $15s$, $14.39s$ and $64.37s$. Thus, in this example, we see the average case improves by roughly 5x and the worst case improves by roughly 48x. The significant overall reduction in the presence of "stragglers" is a manifestation of the result. We kept a log of the number of times the UPDATE LABELS procedure was called for these destinations. A look at Figure 5 confirms our reasoning of "stragglers" being caused by a high number of UPDATE LABELS operations (in the mean case by a factor of 6x) caused by the susceptibility to the kind of cyclic updates we described earlier.

In terms of units of work, not all destinations are equal. Depending on demand, network topology, simulation parameters, etc. the "units of work" we've created by destination can still be unbalanced, causing one thread to be more burdened than the others. This was further exaggerated by the exorbitant number of UPDATE LABELS steps required for certain nodes in the dequeue scheme. However, the priority queue scheme manages to smooth out some of that sharp disparity by drastically cutting down the number of UPDATE LABELS steps required even with the worst-case nodes. Therefore some "stragglers" still exist—*i.e.,* some nodes still finish slower than others—but their worst case is not as bad as that of dequeues. This gives us a less variant, more balanced workload and improving overall application performance when scaling.

Performance of the priority queue scheme relative to the dequeue scheme can been seen in Table II showing a 6x to 12x improvement when scaling up across cores of multiple nodes for one of our larger networks. Figure 6 additionally shows us their relative performance across various network sizes.

V. CONCLUSION & ONGOING WORK

We have presented a distributed TDSP algorithm for DTA with an enhancement and the use of state-of-art computing infrastructure. The evaluation on real world traffic networks shows drastic performance improvements over the exist-

ing implementation. Our approaches can scale well with available computing resources, enabling analysis on much larger traffic networks with complex analytic needs. Our implementation is based on an existing software framework and can be used for DTA-based analysis in traffic research. The approach could be also applied to similar network-based analysis problems in other fields. The use of the RAM disk effectively relieves the I/O bottleneck but also increases the memory requirement. We are working on the development of a generic distributed architecture to efficiently shard route file data across the memory of different nodes to reduce the single-node memory required for large networks.

VI. Acknowledgment

This work has been partly funded through an inter-agency cooperation program with CAMPO and the Texas Department of Transportation. All computations are run using Stampede cluster at Texas Advanced Computing Center funded by the National Science Foundation.

References

[1] H. S. Mahmassani and K. Abdelghany, "Dynasmart-ip: Dynamic traffic assignment meso-simulator for intermodal networks," *Advanced Modeling for Transit Operations and Service Planning*, pp. 201–229, 2003.

[2] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *Journal of the ACM (JACM)*, vol. 37, no. 3, pp. 607–625, 1990.

[3] Y.-C. Chiu, J. Bottom, M. Mahut, A. Paz, R. Balakrishna, T. Waller, and J. Hicks, "Dynamic traffic assignment: A primer," *Transportation Research E-Circular*, no. E-C153, 2011.

[4] D. K. Merchant and G. L. Nemhauser, "Optimality conditions for a dynamic traffic assignment model," *Transportation Science*, vol. 12, no. 3, pp. 200–207, 1978.

[5] S. Peeta and A. K. Ziliaskopoulos, "Foundations of dynamic traffic assignment: The past, the present and the future," *Networks and Spatial Economics*, vol. 1, no. 3-4, pp. 233–265, 2001.

[6] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[7] N. Nezamuddin and S. T. Waller, "Improving the efficiency of dynamic traffic assignment through computational methods based on combinatorial algorithm," in *Transportation Research Board 91st Annual Meeting*, no. 12-4040, 2012.

[8] S. E. Dreyfus, "An appraisal of some shortest-path algorithms," *Operations Research*, vol. 17, no. 3, pp. 395–412, 1969.

[9] A. Ziliaskopoulos, D. Kotzinos, and H. S. Mahmassani, "Design and implementation of parallel time-dependent least time path algorithms for intelligent transportation systems applications," *Transportation Research Part C: Emerging Technologies*, vol. 5, no. 2, pp. 95–107, 1997.

[10] S. Pallottino, "Shortest-path methods: Complexity, interrelations and new propositions," *Networks*, vol. 14, no. 2, pp. 257–267, 1984.

[11] I. Chabini, "Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1645, no. 1, pp. 170–175, 1998.

[12] I. Chabini and S. Ganugapati, "Design and implementation of parallel dynamic shortest path algorithms for intelligent transportation systems applications," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1771, no. 1, pp. 219–228, 2001.

[13] J. A. Villalobos, Y.-C. Chiu, and P. B. Mirchandani, "Numerical performance of spatially and temporally scalable dynamic traffic simulation and assignment system malta," in *Transportation Research Board 88th Annual Meeting*, no. 09-0901, 2009.

[14] Y. Gao and Y.-C. Chiu, "Hierarchical time-dependent shortest path algorithm for dynamic traffic assignment systems," in *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE, 2011, pp. 728–733.

[15] D. Kirchler, L. Liberti, and R. Wolfler Calvo, "A label correcting algorithm for the shortest path problem on a multi-modal route network," in *Experimental Algorithms*, ser. Lecture Notes in Computer Science, R. Klasing, Ed. Springer Berlin Heidelberg, 2012, vol. 7276, pp. 236–247. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30850-5_21

[16] G. Lawson, S. Allen, G. Rose, D. Nguyen, and M. Ng, "Parallel label-correcting algorithms for large-scale static and dynamic transportation networks on laptop personal computers," in *TRB 92th Annual Meetig Compendium of Papers*, no. 13-2103, 2013.

[17] C. F. Daganzo, "The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory," *Transportation Research Part B: Methodological*, vol. 28, no. 4, pp. 269–287, 1994.

[18] NMC. (2014, Jun.) Vista documentation. [Online]. Available: https://nmc-compute1.ctr.utexas.edu/vista/doc/

[19] M. Levin, M. Pool, T. Owens, N. Juri, and S. Travis Waller, "Improving the convergence of simulation-based dynamic traffic assignment methodologies," *Networks and Spatial Economics*, pp. 1–22, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11067-014-9242-x

[20] O. Cavusoglu, V. Sisiopiku, and N. Juri, "Role of transit in carless evacuation planning," *Natural Hazards Review*, vol. 14, no. 3, pp. 191–199, 2013. [Online]. Available: http://dx.doi.org/10.1061/(ASCE)NH.1527-6996.0000106

[21] CAMPO. (2014, Jun.) Campo website. [Online]. Available: http://www.campotexas.org/

[22] TACC. (2014, Jun.) Stampede website. [Online]. Available: https://www.tacc.utexas.edu/stampede/