QUASI-DYNAMIC TRAFFIC ASSIGNMENT USING HIGH PERFORMANCE COMPUTING

A PREPRINT

Cy Chan[*]
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
cychan@lbl.gov

Anu Kuncheria[†]
Institute of Transportation Studies
University of California
Berkeley, CA 94720
anu_kuncheria@berkeley.edu

Bingyu Zhao[‡]
Institute of Transportation Studies
University of California
Berkeley, CA 94720
bz247@berkeley.edu

Theophile Cabannes[§]
Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720
theophile@berkeley.edu

Alexander Keimer[¶]
Institute of Transportation Studies
University of California
Berkeley, CA 94720
keimer@berkeley.edu

Bin Wang[‖]
Energy Technology Area
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
wangbin@lbl.gov

Alexandre Bayen[**]
Lawrence Berkeley National Laboratory
Electrical Engineering and Computer Science
Institute of Transportation Studies
University of California
Berkeley, CA 94720
bayen@berkeley.edu

Jane Macfarlane[††]
Institute of Transportation Studies
Lawrence Berkeley National Lab
University of California
Berkeley, CA 94720
jfmacfarlane@lbl.gov

May 20, 2021

ABSTRACT

Traffic assignment methods are some of the key approaches used to model the routing and flow patterns that arise in transportation networks. Since static traffic assignment does not have a notion of time, it is not designed to represent temporal dynamics that arise as vehicles flow through the network and demand varies throughout the day. Dynamic traffic assignment methods attempt to resolve these issues, but require significant computational resources if modeling urban-scale regions (on the order of millions of links and vehicle trips) and often take days of compute time to complete the

[*]Research Scientist at Lawrence Berkeley National Laboratory
[†]Graduate Student Researcher at University of California, Berkeley
[‡]Researcher at University of California, Berkeley
[§]Graduate Student Researcher at University of California, Berkeley
[¶]Researcher at University of California, Berkeley
[‖]Researcher at Lawrence Berkeley National Laboratory
[**]Professor at University of California, Berkeley
[††]Executive Director Smart Cities at University of California, Berkeley

optimization. The focus of this work is two-fold: 1) to introduce a new traffic assignment approach - a quasi-dynamic traffic assignment (QDTA) model and 2) to describe how we parallelized the QDTA algorithms to leverage High-Performance Computing (HPC) capabilities and scale to large metropolitan areas while dramatically reducing compute time. We examine and compare different scenarios, including a baseline static traffic assignment (STA) and a quasi-dynamic scenario inspired by the user-equilibrium (UET). Results are presented for the San Francisco Bay Area which accounts for about 19M trips/day and an urban road network of about 1M links and is validated against multiple data sources. We utilize an iterative gradient descent method, where the step size is selected using a Quasi-Newton method with parallelized cost function evaluations and compare it to using pre-defined step sizes (MSA). Using the parallelized line search provides a 16 percent reduction in total execution time due to a reduction in the number of gradient descent iterations required for convergence. The full day QDTA comprising 96 optimization steps over 15 minute intervals runs in about 4 minutes on 1,024 compute cores of the NERSC Cori computer, which represents a speedup of over 36x versus serial execution. To our knowledge, this compute time is significantly lower than other traffic assignment solutions for a problem of this scale.

Keywords Dynamic Traffic Assignment · High Performance Computing · User Equilibrium

1 Introduction

Transportation planners often use static traffic assignment solutions to estimate traffic states over the course of one day in their cities [1]. Static traffic assignment does not deal with the dynamic behavior (realistically) that results from network dynamics - it simply assigns an origin/destination (O/D) routing solution that minimizes auto travel time for all mobile entities so that no drivers can unilaterally reduce his/her auto travel costs by shifting to another route. This is known as static user equilibrium (or Nash equilibrium, Wardrop's first principle) [2]. To accommodate the temporal changes in the demand profile over an entire day, the problem is typically partitioned into time slots of interest and static traffic assignment solutions are used to estimate average speeds and average flows for each time slot for the network [3]. Example time slots are early morning, morning rush hour, mid-day, evening rush hour, late evening - accounting for two to four hours of time in each time segment. For very large-scale networks, these models often take many days to run depending upon the hardware and software solutions available to city planners. Consequently, planners will often make adjustments to the model to make the computation tractable within their compute capabilities and time to solution requirements. For example, they may remove lower functional class roads from the network and aggregate travel demand to higher functional class roads. The results are then compromised by the loss of some roads and potential overestimation of flows on the remaining roads [4]. Several limitations of STA has also been identified and discussed in many studies [4, 1, 5]. The underlying assumptions of STA, related to the stationary demand, long aggregation intervals, and static network loading, can lead to unrealistic results when variations in traffic conditions are high [1].

The focus of this work is to use static traffic assignment algorithms implemented on high-performance computing (HPC) to generate results that more effectively estimates a dynamic traffic assignment. We introduce a Quasi Dynamic Traffic Assignment (QDTA), which assigns traffic demand over a sequence of time intervals, where the traffic demand varies and the traffic flow propagates across intervals. At each time step, a modified STA optimization problem is solved, which can be addressed by projected gradient methods where the gradient can be interpreted as a shortest path assignment. We capture the dynamic impact of trips that span multiple time segments by introducing two mechanisms: path truncation and residual demand. This approach addresses the complexity of a dynamic traffic assignment and removes the concern that static traffic assignment cannot provide value to practical traffic flow due to its single time assignment characteristic. In general, while the continuous-time DTA models satisfy the requirements of traffic flow theory, its uniqueness of network equilibrium flows is not necessarily guaranteed [6]. On the other hand, the quasi-dynamic model is based on static traffic assignment, somewhat sharing the formulation and properties of network equilibrium. It is also referred to as discrete-time DTA or semi-dynamic traffic assignment in some literature [7]. QDTA provides a better understanding of the traffic dynamics over STA as it can propagate traffic flows between the time periods using the residual demand.

We use high-performance computing to accelerate the quasi-dynamic traffic assignment algorithm to generate high-fidelity traffic assignments with significantly improved computational efficiency. Optimization on distributed-memory platforms is achieved by utilizing efficient routing algorithms and parallelizing multiple components of the computational workload across threads, including: 1) trip demand routing, 2) cost function evaluation, 3) network flow and weight updates, and 4) the residual demand calculations. The improvements gained using HPC allow us to shorten the time segment duration to 15 minute intervals and address more complex road networks and increased travel demand profiles. Our computational approach scales to one thousand compute cores, allowing us to run assignments with 19 million vehicle trips on a network with 1 million links in under 4 minutes. This capability enables urban-scale traffic assignments in which a variety of scenarios can be addressed with less concern for computational complexity. Examples include:

comparisons among counties or cities, integrating parallel discrete event simulation scenarios that implement and explore infrastructure implications of QDTA, and other objective functions for the optimization.

The remainder of this paper will be organized as follows. First, we will discuss the general form of the proposed quasi dynamic traffic assignment in its pure mathematical form and introduce a proper instantiation of these class of problems, based on the well-known static traffic assignment. Second, we provide the QDTA models computational flow and algorithmic representations that estimate the dynamics using a sequence of time intervals. Third, we present the parallelization of the QDTA model and demonstrate the significant computational gains associated with this approach. Finally, we discuss the scenario comparison for the STA and QDTA models for the entire urban region of the San Francisco Bay Area.

2 Quasi-dynamic traffic assignment and high-performance computing in the literature

In the literature, the dynamics of QDTA comes in two components: demand/routing and dynamic/quasi-dynamic network loading. QDTA differs from STA by the use of much smaller assignment intervals. The implication of having this temporal dimension is the presence of residual demand. The residual demand is handled differently in various previous work. [8, 9] uses a point-queue model to include "capacity constraints", preventing flow from exiting the link if larger than the capacity. This gives more realistic travel times; however, the authors did not demonstrate its application in multi time step situations.

2.1 Dynamic traffic assignment and routing

In this section, we give a short survey of existing modeling and routing approaches. Due to the vast number of publications in this area of research, the presented list of publications cannot be complete. However, we aim to describe the different approaches specifically and at a technical level for each class. We begin this overview with the famous books in this fields: [13, 14, 15].

Continuous time modeling Generally, in a time-continuous setting, the link dynamics can be described via ODEs (ordinary differential equations) or PDEs (partial differential equations). The articles [16, 17, 18, 19, 20] consider dynamic traffic assignment as optimal control problem over a given time horizon. The link dynamics are represented by a (system) of ODEs with in- and outflow functions. An objective function is specified and optimality conditions are deduced. However, there is no natural delay and the optimization is considered on the full time horizon, so that one has to know information about inflow over the full time horizon to determine the routing, i.e. traffic assignment, not to mention that ODEs are generally quite far away from modeling traffic physically accurately (congestion patterns, the named delay, and more).

In [21, 22, 23], more general classes of ODE models (sometimes also with delay) are considered and time-dependent variational inequalities are determined to describe the routing at the junctions. Depending on whether the variational inequality is considered over the full time horizon or at every time (with the ordinary scalar product in $\mathbb{R}^n$ and the variational inequality to hold at every time), the computation of a solution again requires full information of the input datum over the full time horizon considered. In [15] a more detailed mathematical analysis concerning existence of solutions is provided. [24] considers the well-posedness for an ODE model with delay and routing operators at every intersection which can dependent on the entire network state up to real-time. In [25] the routing is realized by a specific routing function taking into account the status of the network. It also proves some stability estimates with regard to the routing considered. Another modeling approach which is also considered in [26, 27] consists of using the Vickrey or point-queue model (or a modified version) [28, 29, 30], again aiming for a routing (and departure choice time) based on variational inequalities. These modeling approaches can be generalized to dynamics which are prescribed by *partial differential equations* (PDEs). We refer to [31, 32, 33, 34, 35, 36] for an overview. Usually, the underlying link dynamics are modeled by the LWR (Lighthill, Whitham, Richards) PDE [37, 38], a hyperbolic conservation law, allowing spillback and congestion. Another approach uses non-local PDE models to simulate traffic flow at junctions [39, 40, 41] and there are also higher order models available, see for instance [42, 43]. However, as mentioned before, a reasonable routing at the intersection needs to be prescribed and the underlying models need to be solved for a given space-time discretization on the entire network, resulting in a quite expensive computational situation.

As stated before, the (time dependent) routing itself – which might depend on the entire network state at a given time (or also the future) – will make the problem even more computationally challenging, not to mention the problem of calibrating the entire system reasonably. These are reasons why we consider in this work a class of simpler models which still have a notion of time and delay, but not to the detail level which the previously mentioned models provide. Roughly, the model parameters required for a static traffic assignment are enough for the proposed QDTA.

Table 1: QDTA in the literature

	Routing	Network loading	Residual demand	Advantages	Limitations	Case studies
[8]	Route-based SUE formulated as VI solved using MSA	first-order node model (i.e., reduction factor based on link demand and supply) formulated as a fixed-point problem solved iteratively	...	Models node capacity at diverges and merges; correct representation of congestion upstream of bottleneck	Does not have temporal dynamics ([8] distinguishes quasi-dynamic assignment from semi-static assignment)	Gold coast, Australia: 9,565 links, 2,987 nodes, 0.3 hour on a personal computer. Sydney, Australia: 75,379 links, 30,573 nodes, 1.1 hour on a personal computer
[9]	System Optimal based on a set of reasonable and independent paths (path-based), assigned using MSA	same as above	...	...	...	Sinox Falls
[10]	SUE, from a shortest path set, no rerouting	Link cost function (e.g., BPR function)	...	...	No rerouting, limited path set	Rome, Italy: 15,000 links, 6,000 nodes, simulates several hours of traffic in a few minutes on a personal computer
[11]						
[12]	Dijkstra shortest path	Link cost constructed from average of GPS speed measurements, weighted by traffic volume reconstructed from the Licence Plate Recognition (LPR) data	No residual demand, likely the OD data is pre-partitioned	Data-driven link cost and demand	Not applicable where the high resolution data (e.g., GPS or toll demand) is limited (e.g., outside of the highways)	Expressway network in Hunan, China. 530 edges and 490 vertexes. Total length is 6725.5km. Computational performance unclear.

Discrete time modeling There is a significant number of articles which deal with discrete time dynamical models. For the sake of an exhaustive overview, see [44, 45, 46, 47, 48, 49]. The articles [50, 51, 52] consider a discretized traffic flow model based on ODEs and time optimal control problem to obtain the routing. Most of the existing literature considers the assignment as an optimization problem across the entire period of interest, with the objective being either minimizing the total travel time (i.e., the System-Optimum, SO, condition, [52]), or some equivalence of the User Equilibrium (UE) conditions ([11]). Depending on the decision variable used in the optimization formulation, the models can be categorized into link-based, path-based, or occasionally intersection-movement-based ([49]). Link-based models can avoid enumerating and storing large path sets and are particularly advantageous for large networks.

In the discretized formulation, the choice of the time step interval is important. Generally, it should be much longer than the link travel time ([51]), but not too long that makes each time slice close to static assignment. In addition, the size of the time step sometimes also depend on the availability of the temporal demand data, which may be available only at coarse grain ([11]). The time-dependent demand is usually considered fixed and known, though [47] proposed algorithms for the cases with uncertainty in the demands (random with certain probability distributions). An important piece of consideration in time discretized DTA formulation is the treatment of residual demand, namely the trips that did not reach the destination in the previous time step. In the existing literature, the residual demand has been treated as model constraints (solved endogenously, e.g., in [52]), a fixed ratio over the inflow ([11]), or obtained from a simulator (e.g., the DYNASMART simulator in [45]).

2.2 High-performance computing for transportation modeling

Parallel computation can be divided broadly into two categories: *shared memory* (e.g. a workstation or a single server node) and *distributed memory* (e.g. clusters, cloud platforms, and HPC systems) [53]. The key distinction is that all cores in a shared memory system have direct access to data in the same address space. That is, data that is written by each compute core is immediately visible to all other cores in the same memory. In contrast, distributed memory systems have separate address spaces that require explicit message passing to share data between cores connected to distinct memory spaces. Programming distributed memory systems is more difficult due to the separate memory, as it requires mechanisms to handle data and load distribution, synchronization, and data movement between processes. Furthermore, there are increased performance overheads due to the time needed to communicate data and synchronize computation across processes. However, the benefits of distributed memory parallelism are two-fold: access to more compute cores and a greater total memory capacity to enable solving larger problems more efficiently. In this contribution, we consider the more difficult problem of distributed memory parallelization of the QDTA algorithm.

The use of high performance computing (HPC) in transportation modeling and simulation tools has not yet achieved widespread adoption. Most simulation tools in this domain support either only sequential execution or (shared memory parallelism, which limits parallelism to the number of cores on a single compute node. Examples that use this type of parallelism include the Aimsun [54] and SUMO [55] simulators. Some traffic simulation software projects, such as FastTrans [56], BEAM [57] and POLARIS [58], have also enabled the use of distributed memory systems. Within the domain of traffic assignment, there have been some previous efforts that demonstrate the use of parallel computation on small to medium sized networks [59, 60]. In [59], they analyze the Nagoya network consisting of 152K links and 38K nodes, using embarrassingly parallel path finding and processing only the active sub-networks to achieve up to 10x speedups using 25 processors. In [60], they utilize multi-threading and MPI to parallelize a link transmission model (LTM) based dynamic traffic assignment for a network with 11k nodes and 23K links, achieving up to 3x speedup on up to four cluster servers. As we demonstrate in our contribution, the combination of an algorithm that lends itself to efficient parallelization along with the use of distributed computing presents a large opportunity to further increase the performance of traffic assignment algorithms for larger scale systems with **millions** of links and vehicles.

3 A tractable dynamic assignment problem: QDTA formulation

The QDTA framework is to propose an efficient and parallelizable method to generate an optimized solution of traffic flows over the network according to the dynamic extensions of the widely used traffic assignment principles, such as the Wardrop's equilibrium, as shown in Sections 3.2 and 3.3. The solution can be used to offer centralized and real-time routing guidance to enhance the traffic flow for a large network. Specifically, the Wardrop's equilibrium has two variations, namely the user equilibrium (UE) solution and the system optimal (SO) solution (see literature review). In this paper, the dynamic extension of the UE solution is adopted as an example in the mathematical formulation of the QDTA framework, with the detailed formulation given in Equation 3 in Section 3.3.

The QDTA model proposed in this article divides the analysis period into small time steps and uses a sequence of STA steps to obtain an optimized route assignment for each time step. The main distinction between the proposed model and the conventional STA framework is the inclusion of *route truncation* and *residual demand*, to address the fact that some trip legs cannot be finished in a single analysis time step and need to be split across multiple steps. As the analysis time step gets shorter to capture short-term traffic dynamics (i.e., 15 minutes), the fraction of trips spanning over multiple time steps become particularly prominent. Section 3.1 introduces notation. Section 3.2 describes route truncation and residual demand in detail. The integration of the residual demand into the well-understood framework of the STA is given in Section 3.3. Lastly, some discussions are presented in Section 3.4 regarding key questions such as the choice of time step length and the ability to resolve adaptive route selection (dynamic rerouting) behavior.

3.1 Notation

The road network is represented by a directed connected graph $G = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of graph nodes/vertices and $\mathcal{A}$ is the set of graph links/edges. A road intersection is represented by a node in the graph $v \in \mathcal{V}$, and the stretch of road between two intersections is represented by a graph link $a \in \mathcal{A}$. Each link has several static properties, such as the traffic flow capacity C_a, and traversal time at the free-flow conditions (free-flow travel time) c_0. Other static node and link properties, such as the coordinates of the nodes, the geometries of the links, are often stored (e.g., for visualization and post-processing analysis). But they are not described here since they are not integral in the formulation of the QDTA framework.

On the travel demand side, the time-dependent travel demand on this road network is represented by time-stamped origin-destination (OD) flows, $d_{p,q}(i)$. p and q denote the origin and destination, with $p \in \mathcal{V}$ and $q \in \mathcal{V}$. Integer N is the total numbers of time steps considered. The time values $t_i \in \boldsymbol{t} = \{t_0, t_1, ..., t_N\}$ refer to the bounds of the analysis time steps, and when we refer to time interval i, we are referring specifically to interval $[t_i, t_{i+1})$. Thus, the duration of time interval i is $\Delta t_i = t_{i+1} - t_i,\ \forall\, 0 \leq i < N$. For example, if considering a traffic assignment period of 24 hours with a uniform time interval of 15 minutes, N then equals to 96 and Δt_i is 15 minutes for all i. Let $\mathcal{R}_{p,q}$ represents the set of acyclic paths in the graph that connect p and q. The traffic flow on each path $r \in \mathcal{R}_{p,q}$ is denoted by $h_r(i)$, or $\boldsymbol{h}(i)$ for all path flows at a particular time interval i. Similarly, the traffic flow on each link $a \in \mathcal{A}$ is denoted by $f_a(i)$, or $\boldsymbol{f}(i)$ for all links.

In our discussion of the parallelized versions of the QDTA algorithms (Section 5.1), we require notation to represent each compute thread's local view (i.e. partition or partial sum) of the above variables. In the rest of this paper, we use the term *partition* to mean the division of a parent vector into two or more child vectors in a way such that each element of the parent vector is assigned to exactly one child vector. In this way, partitioning a parent vector into child vectors is analogous to splitting a parent set into disjoint subsets whose union is the parent. We utilize the subscript $k \in \{1, 2, \ldots, M\}$, where M is the total number of threads, to denote the k-th thread's partition of the original variable. For example, parallel decomposition of the demand is accomplished by partitioning the total travel demand $\boldsymbol{d}(i)$ into $\{\boldsymbol{d}_k(i) \mid \forall k\}$, and similar for $\boldsymbol{d}^o(i)$ and $\boldsymbol{d}^r(i)$. Furthermore, $\boldsymbol{h}_k(i)$ represents the flows on the paths that correspond to $\boldsymbol{d}_k(i)$. Another form of parallel decomposition is of the links in the network, where $\mathcal{A}_k$ represents thread k's partition of $\mathcal{A}$. $\boldsymbol{f}_k(i)$ is different in that it represents the flows on **all** links that result from the partitioned demand $\boldsymbol{h}_k(i)$, such that $\boldsymbol{f}(i) = \sum_k \boldsymbol{f}_k(i)$. These variables and their use will be further detailed in Section 5.1.

3.2 Residual demand and route truncation

In QDTA, travel demands are supplied in stages (time step intervals) according to the analysis period that the departure times are in. The assignment framework presented in this paper does not constrain the duration of the time steps. As a result, different time step lengths can be used, e.g., 5 minutes or 1 hour. However, a time step that is too long would approximate the STA solution and lack the desired temporal variations in the modeling results, while too short a time step increases computational cost and requires a higher resolution temporal distribution of the demand inputs.

A major complication of selecting a small time interval (relative to the duration of the average trip, e.g., 15 minutes) is that a large fraction of trips do not finish within one assignment period. This is problematic as established optimization algorithms to obtain traffic assignment solutions (e.g., the Frank-Wolfe's algorithm) assume all vehicles reach their destinations during the assignment [61], which is not realistic for assignment over short time intervals. To address this issue, we introduce a modification to the STA approach by adding a *route truncation* operation that is used repeatedly in the algorithm. In this section, the mathematical formulation of the route truncation operation will be introduced. Its integration into the QDTA framework will be given in Section 3.3.

The route truncation operation estimates the intermediate stop location s that a trip in $d_{p,q}(i)$ can reach before time t_{i+1}. For long trips, s may be different from the trip destination q, and the remaining leg of the trip, i.e., from s to q, will enter the next time step as the *residual demand*. The determination of s relies on knowing the travel time (or other general cost) of each link, $\boldsymbol{c}(i) = \{c_a(i)\}, a \in \mathcal{A}$, which is a function of the link flow $\boldsymbol{f}(i) = \{f_a(i)\}, a \in \mathcal{A}$ assigned to the network during interval i.

Table 2: Meanings of mathematical expressions

Symbol	Meaning
	Parallel decomposition
$M, k \in \{1, 2, \ldots, M\}$	Number of compute threads, thread index.
	Time-step related variables
$N, i \in \{0, 1, \ldots, N-1\}$	Total number of time intervals, time interval index.
$[t_i, t_{i+1})$	i^{th} time interval bounds.
$\Delta t_i = t_{i+1} - t_i$	Duration of time interval i.
	Network properties
$G = (\mathcal{V}, \mathcal{A})$	Road network graph.
$\mathcal{V}$, v	Set of road network vertices, one vertex in the road network.
$\mathcal{A}$, a	Set of road network edges, one edge in the road network.
$\mathcal{A}_k$	Thread k's partition of network edges.
C_a	Flow capacity of edge a.
$\boldsymbol{c}_0$, $c_{0,a}$	Free-flow travel time/cost of all edges, or a specific edge a.
$\boldsymbol{c}(i)$, $c_a(i)$	Time/Cost of traversing all edges, or a specific edge a, in time interval i.
	Travel demand
$p \in \mathcal{V}$	Trip origin.
$q \in \mathcal{V}$	Trip destination.
$s \in \mathcal{V}$	Intermediate stop location.
$d_{p,q}(i)$	Travel demand (trips) starting at node p, ending at node q, departing in time interval i .
$\boldsymbol{d}^o(i)$, $\boldsymbol{d}^r(i)$, $\boldsymbol{d}(i)$	All original, residual, and combined trips traveling in time interval i.
$\boldsymbol{d}^o_k(i)$, $\boldsymbol{d}^r_k(i)$, $\boldsymbol{d}_k(i)$	Thread k's partition of corresponding trips in time interval i.
	Paths and network flows
$\mathcal{R}_{p,q}$, r	All acyclic paths in the road network that connect p and q, a single path in the set.
$len(r)$, r_k	Total number of vertices along path r, the k^{th} vertex along path r.
$\boldsymbol{h}(i)$, $h_r(i)$	Traffic flow assigned to all paths, or path r, in time interval i.
$\boldsymbol{f}(i)$, $f_a(i)$	Traffic flow assigned to all links, or link a, in time interval i.
$\boldsymbol{h}_k(i)$, $\boldsymbol{f}_k(i)$	Traffic flow assigned to paths corresponding to $\boldsymbol{d}_k(i)$, and their resulting partial link flows.

In general, the travel time costs are a function of the link flows: $\boldsymbol{c}(i) = \boldsymbol{c}(\boldsymbol{f}(i)), \forall i$. This function is usually selected to satisfy both element-wise monotonicity and separability assumptions. The element-wise monotonicity underlines the fact that as more flow is assigned to a road, the longer it takes on average to traverse it. The separability assumption states that travel time on a link only depends on the current flow assignment on the considered link. A typical choice of the travel time function that satisfies both assumptions is the well-known Bureau of Public Roads (BPR) curves [62], as shown in Equation (1). In Equation (1), $c_a(i)$ is the travel time on link a in time interval i; $c_{0,a}$ and C_a are the free-flow travel time and capacity associated with the link; α and β are calibration parameters selected such that $c_a(i)$ is monotonic with respect to $f_a(i)$.

$$c_a(i) = c_{0,a} \cdot \left(1 + \alpha\left(\frac{f_a(i)}{C_a}\right)^{\beta}\right) \tag{1}$$

Based on the link-level travel time $\boldsymbol{c}(i)$, the stop location s can be determined using Equation (2). r is the path of the trip to its destination q. If the time step duration is long enough, longer than the total time required to traverse r, i.e., first row in Equation (2), the intermediate stop location s is the trip destination q. However, if this condition cannot be met, the stop location is a node along the path r. Let $len(r)$ denote the total number of nodes along path r, and r_j to be the j^{th} node on the path, the second row of Equation (2) leads to the furthest distance that can be covered during a time duration of Δt_i.

$$s = \begin{cases} q & \text{if } \Delta t_i \geq \sum\limits_{a \in r} c_a(i) \\ r_{\arg\min\limits_{j \in len(r)} \sum\limits_{a \in r_j} max\{\Delta t_i - c_a(i), 0\}} & \text{otherwise} \end{cases} \tag{2}$$

Thus, for trip $d_{p,q}(i)$ with route r, only the first part of the trip up to vertex s will contribute towards the path/link flow in the time interval i, while the remainder of the trip will be added to the next time step's in the form of carry-over demand $d_{s,q,t_{i+1}}$. Note that particularly long trips may carry over several times, and thus span several time segments.

3.3 Full formulation

In this section, the temporal update steps to obtain the path/link traffic flow assignment will be presented. At each time step, the travel demand $\boldsymbol{d}(i)$ consists of two parts, the original trips $\boldsymbol{d}^o(i)$ that start in time interval i, and the residual demand trips $\boldsymbol{d}^d(i)$ that started before t_i but have not yet reached their destinations. This applies to all time steps except the first time interval 0, where there is no residual demand. Let $\boldsymbol{F}$ be the function that maps the travel demand to path flows:

$$\boldsymbol{h}(i) = \begin{cases} \boldsymbol{F}(\boldsymbol{d}^o(0)) & i = 0 \\ \boldsymbol{F}(\boldsymbol{d}^o(i) + \boldsymbol{d}^r(i)) & i \in 1, ..., N-1 \end{cases} \tag{3}$$

Then, Equation (4) is an instantiation of $\boldsymbol{F}$ that finds an optimum static assignment of path flow $\boldsymbol{h} = \{h_{p,q,r}\}, p, q \in \mathcal{V}, r \in \mathcal{R}_{p,q}$ that satisfies the UE condition (provided the cost function $c_a(s)$ is strictly monotone) with proof in [62].

$$\boldsymbol{F}(\boldsymbol{d}) = \arg\min_{\boldsymbol{h}} \sum_{a \in \mathcal{A}} \int_0^{f_a} c_a(s) ds \tag{4a}$$

$$\text{subject to} \sum_{r \in \mathcal{R}_{p,q}} h_r = d_{p,q} \qquad \forall (p,q) \in \mathcal{V} \times \mathcal{V} \tag{4b}$$

$$h_r \geq 0 \qquad \forall r \in \mathcal{R}_{p,q} \tag{4c}$$

$$f_a = \sum_{p,q \in \mathcal{V}} \sum_{r \in \mathcal{R}_{p,q}} \Delta_{a,r} h_r \qquad a \in \mathcal{A} \tag{4d}$$

$\boldsymbol{\Delta} = [\Delta_{a,r}]$ is the incidence matrix, whose value is 1 if the link is on the path before the intermediate stop s, and 0 otherwise:

$$\Delta_{a,r} = \begin{cases} 1 & \text{if } a \in \{(r_0, r_1), ..., (r_{j-1}, r_j)\}, r_j = s \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Equation (4) is essentially an extension of the static UE formulation in [62], with the static travel demand, link and path flow replaced by the time-dependent $\boldsymbol{d}(i)$, $\boldsymbol{f}(i)$ and $\boldsymbol{h}(i)$. Unlike the original static formulation, where the path-link incident matrix only depends on knowing whether a link is on a path, in the quasi-dynamic formulation, a link on a path may not be traversed in the current time step. As a result, the path flow in QDTA is truncated at the stop location, s, first using Equation (2), before mapped to the link flow $\boldsymbol{f}(i)$ using Equations (4d)-(5).

3.4 Discussions of the QDTA framework

Some further discussions are provided regarding the proposed QDTA framework. First, the selection of the time interval durations $\{\Delta t_i\}$ reflects a few trade-offs. If the time step length is too long, the solution will approximate the STA solution and fail to capture the changing temporal traffic dynamics. However, if the time step length is too short, it will not only increase the computational time significantly, but also lead to less accurate results as microscopic, sub-link dynamics start to show. For example, consider an extreme scenario, where Δt_i is shorter than the free flow time of the road link, $c_{0,a}$, Equation (2) indicates that the intermediate stop node s is always the origin node. In other words, the trips can never propagate through the link. This limitation can potentially be overcome by using alternative formulations of Equation (2) that tracks the distance of the traffic flow on a link, and cumulatively adds up the distance across time steps if the traffic flow cannot be propagated to the next link.

A key feature of the proposed QDTA framework is the inclusion of dynamic rerouting in response to changes in the network at each time interval. For every time interval i, the routing is updated to reflect the current traffic conditions and potential changes in the network (e.g., road closures due to earthquake damages). The route assignment is thus obtained through an iterative process between the path flow, $\boldsymbol{h}(i)$, and the path truncation, and reflects the optimum (e.g., UE) traffic distribution within each time step. The dynamic rerouting is not considered in some previous DTA algorithms. But it is particularly relevant, especially in the era of navigation applications that dynamically update the navigation routes for users, to include such phenomena in traffic modeling and predictions.

4 Algorithmic Representations

The mathematical formulation of the QDTA framework in Section 3 is solved using a number of scalable algorithms. The pseudo-code of these algorithms are presented in this section. At the highest level, the QDTA implementation (Algorithm 1) loops through each time step $\{t_0,\ t_1,\ ...,\ t_{N-1}\}$ (Algorithm 1). Inside each time step, an STA-based flow solution is first obtained using the Frank-Wolfe's algorithm (Algorithm 2, Algorithm 3). Note that path truncation is performed in each iterative step of the Frank-Wolfe's algorithm (Algorithm 3). This ensures that, for each path, only the approximate portion traversed *within the current time segment* is considered so as to prevent erroneous congestion effects of traffic that occurs outside the current time segment. Based on the converged solution from Algorithm 2, a last round of route truncation is performed to calculate the intermediate positions s of the residual demand to be carried over to the next time step (Algorithm 4).

```
Algorithm 1: Quasi-dynamic traffic assignment
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t_i$
      Total time step counts $N$
      Original travel demand of all time steps $\boldsymbol{d}^o = \{\boldsymbol{d}^o(i)\}$, with $i \in \{0, ..., N-1\}$

Initialize $\boldsymbol{d}^r(0)$ = empty nested associative array ;     // No residual demand in the first time step

for $i \in \{0, \dots, N-1\}$ ;                                          // Sequential discrete time step
do
    $\boldsymbol{d}(i) = \boldsymbol{d}^o(i) + \boldsymbol{d}^r(i)$ ;                          // Get original and residual demand
    $\boldsymbol{h}^{STA}(i) =$ Traffic_assignment$(G,\ \Delta t_i,\ \boldsymbol{d}(i))$ ;          // Path flow assignment using STA
    $\boldsymbol{h}(i), \boldsymbol{d}^r(i+1) =$ Residual_demand$(G,\ \Delta t_i,\ \boldsymbol{h}^{STA}(i))$ ; // Truncate path flows and get residual
     demand based on time step length
end
Result: $\{\boldsymbol{h}(i) \mid i \in \{0, \dots, N-1\}\}$ ;                        // Path flow for all time steps
```

```
Algorithm 2: Traffic_assignment: using Frank-Wolfe's algorithm
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t$
      Travel demand of current step $\boldsymbol{d} \in \mathbb{R}_{\geq 0}^{\mathcal{V} \times \mathcal{V}}$
      Free-flow travel time of each link $\boldsymbol{c}_0 = \{c_{0,a}\}$, with $a \in \mathcal{A}$

Take $\boldsymbol{h} =$ All_or_nothing$(G, \Delta t, \boldsymbol{d}, \boldsymbol{c}_0)$ ;     // Set initial path flow in the free-flow condition

while True ;                                                              // Gradient descent step
do
    $\boldsymbol{c} =$ BPR$(\boldsymbol{\Delta h})$ ;                                    // Calculate the edge travel time
    $\boldsymbol{h}^{AON} =$ All_or_nothing$(G, \Delta t, \boldsymbol{d}, \boldsymbol{c})$ ;       // All-or-nothing path flow with new edge weights
    $\alpha^\star = \arg\min_\alpha$ Cost_function$\Big(\boldsymbol{\Delta}\Big[\boldsymbol{h} + \alpha \cdot (\boldsymbol{h}^{AON} - \boldsymbol{h})\Big]\Big)$ ;             // Exact line search
    $\boldsymbol{h}_{new} = \boldsymbol{h} + \alpha^\star \cdot (\boldsymbol{h}^{AON} - \boldsymbol{h})$ ;                              // Update path flows
    if Converged $(\boldsymbol{\Delta h}, \boldsymbol{\Delta h}_{new})$ ;                               // Check convergence
    then
        break
    end
    $\boldsymbol{h} = \boldsymbol{h}_{new}$ ;
end
Result: $\boldsymbol{h}$ ;                                                       // Path flow using STA
```

Algorithm 3: All_or_nothing: iterative step of the Frank-Wolfe Algorithm

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t$
      Travel demand of current step $\boldsymbol{d} \in \mathbb{R}_{\geq 0}^{\mathcal{V} \times \mathcal{V}}$
      Edge travel time/cost $\boldsymbol{c} = \{c_a\}, a \in \mathcal{A}$

Initialize $\boldsymbol{h}$ = empty associative array ;          // Initialize the path flow vector

for $d_{p,q} \in \boldsymbol{d} \mid d_{p,q} > 0$ ;                // Iterate over non-zero elements
do
    $r_{sp}$ = Get_shortest_path$(p, q, \boldsymbol{c}, G)$ ;      // Get shortest path to destination
    $r_{tp}$ = Truncate_path$(r_{sp}, \boldsymbol{c}, G, \Delta t)$ ; // Truncate path according to time step length
    $h_{r_{tp}}$ += $d_{p,q}$ ;                                    // Add trips to the path flow vector
end
Result: $\boldsymbol{h}$ ;                                         // Path flow from all-or-nothing assignment
```

Algorithm 4: Residual_demand: trips that cannot finish in one assignment interval

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t_i$
      Intermediate path flow results from the STA $\boldsymbol{h}^{STA}(i)$
      All paths used in the time step $\mathcal{R}(i) = \{\mathcal{R}_{p,q}\}$, with $(p,\ q) \in \mathbb{R}_{\geq 0}^{\mathcal{V} \times \mathcal{V}}$

Initialize $\boldsymbol{d}^r(i+1)$ = empty nested associative array ;   // Initialize the residual demand vector,
 will be added to the demand of the next step
Initialize $\boldsymbol{h}(i)$ = empty associative array ;          // Initialize the truncated path flow vector
Take $\boldsymbol{c} = BPR(\boldsymbol{\Delta h}^{STA}(i))$ ;        // Edge travel time based on path/link flow from STA

for $r \in \mathcal{R}(i)$ ;                                        // Parallelizable for loop
do
    $r_{tp}$ = Truncate_path$(r, \boldsymbol{c}, G, \Delta t_i)$ ;   // Truncate path to what can be traversed in the time
     step
    $h_{r_{tp}}(i)$ += $h_r^{STA}(i)$ ;                              // Populate the path flow vector
    if $r_{tp} \neq r$ ;              // Flow has not reached its destination within the time slice
    then
        $s$ = Get_last_vertex$(r_{tp})$;
        $q$ = Get_last_vertex$(r)$;
        $d_{s,q}^r(i+1)$ += $h_r^{STA}(i)$ ;                          // Add to residual demand
    end
end
Result: $\boldsymbol{h}(i), \boldsymbol{d}^r(i+1)$ ;   // Path flow for the current time step using QDTA, and residual
        demand to be added to the next time step
```

The repeated usage of route truncation resolves the issue of traffic that is resident in the network for a longer time horizon than the given QDTA time segment duration by re-assigning the residual traffic to the next time interval. With this capability, the demand entering the network at the next time interval along with the residual demand is then introduced to the following time segment's traffic assignment. This process is repeated until the final time of the QDTA is reached. The model can be interpreted as an time-expanded network approach where the expansion of the network represents the evolution with respect to time.

In the algorithm, the following operations are used but not explicitly defined:

- `Shortest_path` returns the shortest path between the origin p and the destination q given the road network graph $G(\mathcal{V}, \mathcal{A})$ with cost $\boldsymbol{c} = \{c_a\}$, with $a \in \mathcal{A}$.
- `Truncate_path` returns the sub-path of a path r that can be traversed in time Δt_i given the travel time on the edges.
- `Get_last_vertex` returns the last vertex of a path r.
- `Cost_function` is the function to be minimized, i.e. the Rosenthal potential function with $c_a(s)$ chosen to be a monotonic function as in Eq. (1):

$$\text{Cost\_function}(\boldsymbol{f}) = \sum_{a \in \mathcal{A}} \int_0^{f_a} \boldsymbol{c}_a(s)\, ds \tag{6}$$

- `Converged` checks the convergence of the Frank-Wolfe algorithm. The algorithm is considered to be converged if the relative absolute change in the cost function is less than $T_0 = 0.0001$. This value was chosen as a compromise between quality and speed of convergence.

$$\text{Converged}(\boldsymbol{f}_1, \boldsymbol{f}_2) = \begin{cases} \text{True}, & \text{if } \left| \frac{\text{Cost\_function}(\boldsymbol{f}_1) - \text{Cost\_function}(\boldsymbol{f}_2)}{\text{Cost\_function}(\boldsymbol{f}_1)} \right| < T_0 \\ \text{False}, & \text{otherwise.} \end{cases} \tag{7}$$

5 High-performance computational solutions for QDTA

Our overall goal is to reduce the compute time for modeling urban mobility so that city planners can investigate a large number of scenarios in a reasonably small period of time, while still preserving the fidelity of the inherent traffic dynamics. The scope of the models considered in our work are full urban networks that include all functional road classes and a full urban scale travel demand. Our software framework takes the network graph (nodes and links) and traffic demand (origin and destination nodes and start times) as inputs. The demand is converted into $\{\boldsymbol{d}^o(i) \mid \forall i\}$ based on their origin and destination nodes $p, q \in \mathcal{V}$ and starting times t.

Figure 1 shows the computational flow of our implementation of QDTA. All algorithms in the blue boxes have been parallelized to run on HPC platforms. The main (outer) loop in Figure 1 corresponds to one iteration of Algorithm 1. The Frank-Wolfe optimization is fully parallelized, including the routing, line search, and network weight updates. The method of parallelization of the core algorithms described in Section 4 will be described in detail in the remainder of this section.

5.1 Routing and network flow parallelization

Algorithm 5 describes the distributed-memory parallel QDTA algorithm. When parallelizing any algorithm, one of the design choices is how to partition both the data and computational work across available compute resources. This choice may differ depending on the algorithmic step being parallelized (see Figure 1). Due to the high computational cost of computing shortest path routes over a large network, the most critical step to optimize for performance is the routing of all active vehicles in the current time segment (Algorithm 3). We can parallelize the routing with respect to each origin and destination pair $p, q \in \mathcal{V}$ as they do not influence each other within a single step of the Frank-Wolfe algorithm. In order to achieve effective parallelization of the routing step, and to distribute storage and management of the resulting routes, each thread k is assigned a partition of the demand $\boldsymbol{d}_k^o(i)$ to compute their routes and flows $\boldsymbol{h}_k(i)$, manage their corresponding residual demand $\boldsymbol{d}_k^r(i+1)$, and finally store their results to disk at program completion.

Algorithm 6 describes the parallelized Frank-Wolfe algorithm that assigns traffic to each time segment. The first step in this algorithm is the parallel all-or-nothing routing step described in Algorithm 7. Note that each thread requires full knowledge of the network's current link weights to be able to route its partition of trip legs d_k. An important optimization we made for the routing step involves utilizing a multi-phase routing algorithm, where the network is first pre-processed with connectivity and weight information to enable subsequent routing queries to be computed

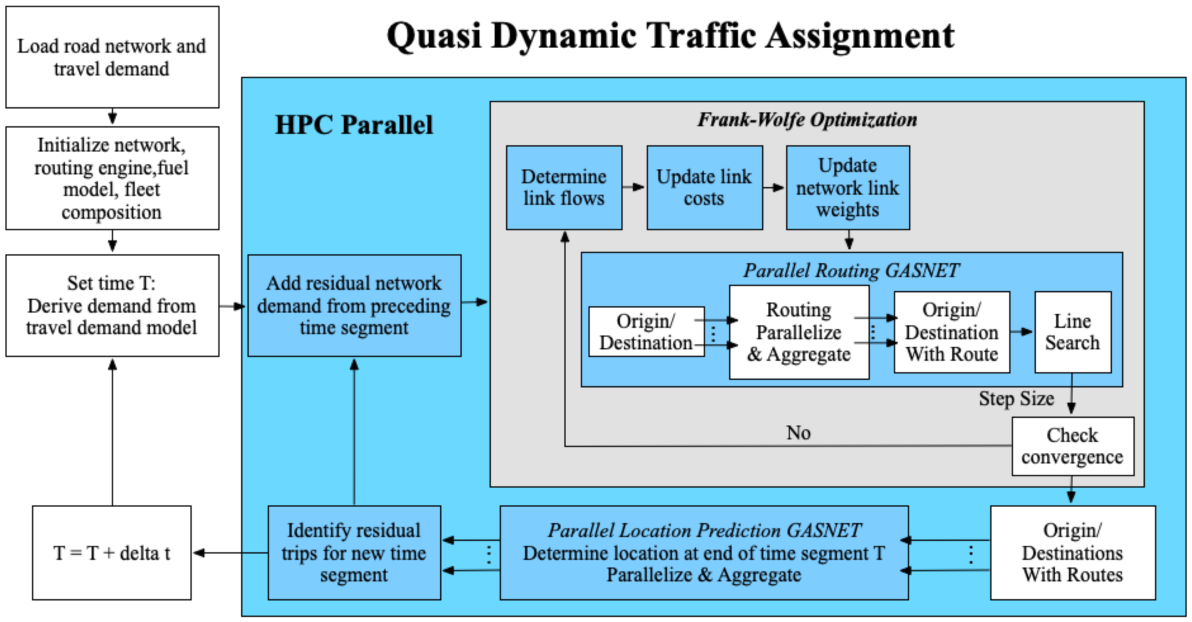

Quasi Dynamic Traffic Assignment
Load road network and travel demand
Initialize network, routing engine, fuel model, fleet composition
Set time T: Derive demand from travel demand model
T = T + delta t
HPC Parallel
Add residual network demand from preceding time segment
Identify residual trips for new time segment
Frank-Wolfe Optimization
Determine link flows
Update link costs
Update network link weights
Parallel Routing GASNET
Origin/ Destination
Routing Parallelize & Aggregate
Origin/ Destination With Route
Line Search
Step Size
No
Check convergence
Parallel Location Prediction GASNET
Determine location at end of time segment T
Parallelize & Aggregate
Origin/ Destinations With Routes

Figure 1: Computational Flow for QDTA : blue box indicates the parts which have been parallelized. This flow diagram corresponds to the pseudo-algorithm described in Algorithm 3.

Algorithm 5: Parallel quasi-dynamic traffic assignment

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t_i$
      Total time step counts $N$
      Original travel demand of all time steps $\boldsymbol{d}^o = \{\boldsymbol{d}^o(i)\}$, with $i \in \{0, ..., N-1\}$

Let $k$ be this thread's index ;
Initialize $\boldsymbol{d}_k^r(0)$ = empty nested associative array ;    // No residual demand in the first time step

for $i \in \{0, \dots, N-1\}$ do
    Let $\boldsymbol{d}_k^o(i)$ be thread $k$'s partition of $\boldsymbol{d}^o(i)$ ;
    $\boldsymbol{d}_k(i) = \boldsymbol{d}_k^o(i) + \boldsymbol{d}_k^r(i)$ ;    // Combine original and residual demand
    $(\boldsymbol{h}_k^{STA}(i), \boldsymbol{f}^{STA}(i))$ = Parallel_traffic_assignment$(G,\ \Delta t_i,\ \boldsymbol{d}_k(i))$ ;
    $\boldsymbol{h}_k(i), \boldsymbol{d}_k^r(i+1)$ = Parallel_residual_demand$(G,\ \Delta t_i,\ \boldsymbol{h}_k^{STA}(i))$ ;
end
Result: $\boldsymbol{h}_k = \boldsymbol{h}_k(i)$, with $i \in 0, ..., N-1$ ;    // Distributed path flows for all time steps
```

Algorithm 6: Parallel_traffic_assignment: using Frank-Wolfe's algorithm

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t$
      Partition of travel demand of current step $\boldsymbol{d}_k$
      Free-flow travel time of each link $\boldsymbol{c}_0 = \{c_{0,a}\}$, with $a \in \mathcal{A}$

Let $(\boldsymbol{h}_k, \boldsymbol{f}) =$ Parallel_all_or_nothing$(G, \Delta t, \boldsymbol{d}_k, \boldsymbol{c}_0)$ ;      // Set initial path flow

while True ;                                                  // Gradient descent step
do
    $\boldsymbol{c} = BPR(\boldsymbol{f})$ ;                                // Calculate the edge travel time
    $(\boldsymbol{h}_k^{AON}, \boldsymbol{f}^{AON}) =$ Parallel_all_or_nothing$(G, \Delta t, \boldsymbol{d}_k, \boldsymbol{c})$ ;
    $\alpha^\star =$ Parallel_line_search$(G, \boldsymbol{f}, \boldsymbol{f}^{AON})$ ;
    $\boldsymbol{f}_{new} = \boldsymbol{f} + \alpha^\star \cdot (\boldsymbol{f}^{AON} - \boldsymbol{f})$ ;                       // Update link flows
    if Converged $(\boldsymbol{f}, \boldsymbol{f}_{new})$ ;                                // Check convergence
     then
        break
    end
    $\boldsymbol{h}_k = \boldsymbol{h}_k + \alpha^\star \cdot (\boldsymbol{h}_k^{AON} - \boldsymbol{h}_k)$ ;                   // Update path flows
    $\boldsymbol{f} = \boldsymbol{f}_{new}$ ;
end
Result: $\boldsymbol{h}_k, \boldsymbol{f}$ ;                          // Path flows (local) and link flows (global)
```

Algorithm 7: Parallel_all_or_nothing: iterative step of the Frank-Wolfe Algorithm

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Time step length $\Delta t$
      Partition of travel demand of current step $\boldsymbol{d}_k$
      Edge travel time/cost $\boldsymbol{c} = \{c_a\}, a \in \mathcal{A}$

Pre-process (customize) network with updated link weights $\boldsymbol{c}$ ;
Initialize $\boldsymbol{h}_k =$ empty associative array ;          // Initialize the path flow vector

for $d_{p,q} \in \boldsymbol{d}_k \mid d_{p,q} > 0$ ;                           // Iterate over non-zero elements
do
    $r_{sp} =$ Get_shortest_path$(p, q, \boldsymbol{c}, G)$ ;                              // Get shortest path
    $r_{tp} =$ Truncate_path$(r_{sp}, \boldsymbol{c}, G, \Delta t)$ ;        // Truncate path according to time step length
    $h_{r_{tp}}$ += $d_{p,q}$ ;                             // Add trips to the local path flow vector $\boldsymbol{h}_k$
end
$\boldsymbol{f}_k = \boldsymbol{\Delta} \boldsymbol{h}_k$ ;                           // Compute local link flows from local path flows
$\boldsymbol{f} =$ Global all-reduce$(+, \boldsymbol{f}_k)$ ;              // Compute global link flows from local link flows
Result: $\boldsymbol{h}_k, \boldsymbol{f}$ ;                          // Path flows (local) and link flows (global)
```

very efficiently and in parallel. To this end, we leverage Customizable Contraction Hierarchies [63], which further splits the preprocessing phase in two, allowing the weights of the existing links in the network to be updated with less computation time compared to doing a full topological update (where the connectivity of the links may also change). In Algorithm 7, the preprocessing is done first so that all threads can subsequently utilize the pre-processed network with updated weights to compute its assigned routes.

After the parallel routing and truncation step (the **for** loop in Algorithm 7), each process's memory contains only the routes computed by the threads local to that process (implicit in $\boldsymbol{h}_k$). However, in order to proceed to the next algorithmic step, the impact of all routes *globally* must be taken into consideration and reflected in the network flow data in every process. A key observation is that we do not need to globally broadcast the actual routes $\boldsymbol{h}$ computed for every vehicle trip leg, as this would be a very large amount of data to communicate. Instead, the thread local *route* flows $\boldsymbol{h}_k$ are reduced to thread local *link* flows $\boldsymbol{f}_k$ in parallel, and then the local link flows are globally all-reduced to calculate the total link flows $\boldsymbol{f}$ that result from all vehicle trip legs. In this way, we avoid having to communicate any route information between parallel processes, only the resultant flows on the links themselves.

5.2 Line search parallelization

The next stage in Algorithm 6 after the all-or-nothing calculation is the line search to select the optimal α step size. The parallelized algorithm selects the step size using $\boldsymbol{f}$ directly instead of $\boldsymbol{h}$ (as was done in Algorithm 2), but is equivalent since multiplication with the incidence matrix $\boldsymbol{\Delta}$ distributes in the cost function expression, i.e.:

$$\boldsymbol{\Delta}\left[\boldsymbol{h} + \alpha \cdot (\boldsymbol{h}^{AON} - \boldsymbol{h})\right] = \boldsymbol{f} + \alpha \cdot (\boldsymbol{f}^{AON} - \boldsymbol{f}), \text{ thus} \tag{8}$$

$$\alpha^\star = \arg\min_{\alpha} \text{Cost\_function}\left(\boldsymbol{f} + \alpha \cdot (\boldsymbol{f}^{AON} - \boldsymbol{f})\right). \tag{9}$$

If evaluated sequentially, the line search is a very computationally expensive step since it requires an iterative search where the cost function is evaluated many times for potential values of α until the minimum is found. Furthermore, each single evaluation of the cost function requires summing the cost contribution of every link in the network, which is substantial for large networks with millions of links. As a result, many computational traffic assignment implementations will simply use the method of successive averages (MSA) [64] or other pre-determined step size sequences as a heuristic to select the gradient descent step size to avoid this high computational cost. However, the MSA method has a drawback in that taking sub-optimal step sizes for each gradient descent iteration sometimes results in requiring more iterations overall for the traffic assignment to converge. We have found that by selecting the optimal step size for each iteration, the number of gradient descent iterations may be reduced significantly for some time intervals. Therefore, computing the line search efficiently through parallelization is an effective capability of our approach.

Algorithm 8 describes our parallel line search algorithm, which is the implementation of Equation 9. In our algorithm, the value of α is constrained to the range $[0, 1]$ to ensure the solution is a convex combination of legal route assignments to prevent unrealistic (e.g. negative) flow values. For brevity, we introduce the short-hand function $C(x)$, which is equivalent to the cost function evaluated with step size x and implicit arguments $\boldsymbol{f}$ and $\boldsymbol{f}^{AON}$. The algorithm uses a Quasi-Newton method to identify where C is minimized (i.e. where the derivative of the C equals zero). Thus, at each step in the Quasi-Newton method, we require approximations of the first two derivatives of $C(x)$ at the current guess. These approximations are calculated using the finite difference method by evaluating C at the values: $\{x - \Delta x, x, x + \Delta x\}$, where nominally $\Delta x = 0.0001$ is chosen because it is a small fraction of the search range and works well in practice. Then:

$$C'(x) \approx \frac{C(x + \Delta x) - C(x - \Delta x)}{2\Delta x}, \quad \text{and} \quad C''(x) \approx \frac{C(x - \Delta x) - 2C(x) + C(x + \Delta x)}{\Delta x^2}. \tag{10}$$

The thresholds T_1 and T_2 in the algorithm are tunable parameters to control the quality of convergence. We have used $T_1 = T_2 = 0.0001$ in our experiments to ensure good convergence of the line search. Figure 2a shows the average and maximum number of Quasi-Newton iterations required to conduct the line search for all gradient descent iterations taken within each time segment. The average number of line search iterations ranges between 1 and 5, while the maximum is as high as 9 for the most highly congested time segments.

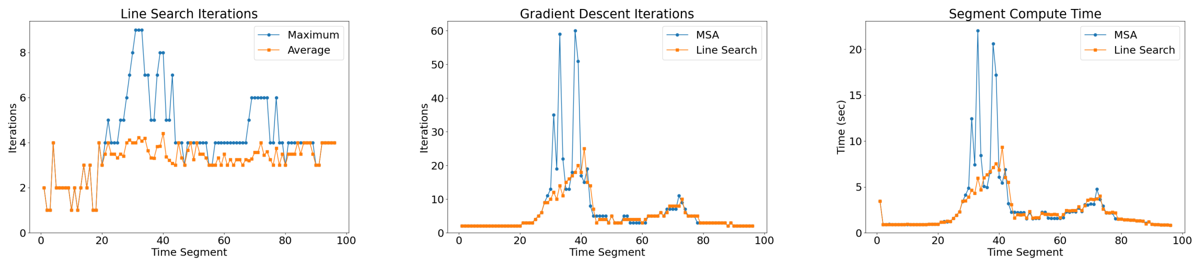
The key to computing the line search efficiently is evaluating $C(x - \Delta x)$, $C(x)$, and $C(x + \Delta x)$ in parallel and as a batch. Since $C(x)$ is the sum of partial cost contributions from each link in the network (see Equation 6), parallelization is achieved by partitioning the links in the network across available threads. Algorithm 9 describes our approach to evaluate the cost function and its derivatives in parallel. For each set of three cost function evaluations, each thread computes its local contributions to the three cost functions for the partition of links assigned to that thread $\mathcal{A}_k$. The resulting local cost contributions are then globally all-reduced across threads to obtain the total cost function values. In

Algorithm 8: Parallel_line_search: Quasi-Newton line search for optimal $\alpha^\star$

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Current link flows $\boldsymbol{f}$
      All-or-nothing link flows $\boldsymbol{f}^{AON}$
      Current Frank-Wolfe gradient descent iteration $j$
      Line search iteration maximum steps $L$
      Convergence threshold (slope) $T_1$
      Convergence threshold (step size) $T_2$
Initialize $\alpha = \frac{2}{2+i}$ ;                                  // Initial guess
for $l = 1;\ l < L;\ l \mathrel{+}= 1$ ;               // Quasi-Newton line search iteration
 do
    $[C(\alpha), C'(\alpha), C''(\alpha)] =$ Parallel_cost_function$(G, \alpha, \boldsymbol{f}, \boldsymbol{f}^{AON})$ ;
    if $C'(\alpha) < T_1$ ;                                             // Threshold 1
     then
        $\alpha^\star = \alpha$ ;
        break
    end
    $\alpha_{new} = \alpha - C'(\alpha)/C''(\alpha)$ ;
    Enforce $0 <= \alpha_{new} <= 1$ ;
    if $|\alpha_{new} - \alpha| < T_2$ ;                                // Threshold 2
     then
        $\alpha^\star = \alpha_{new}$ ;
        break
    end
    $\alpha = \alpha_{new}$
end
Result: $\alpha^\star$
```

Algorithm 9: Parallel_cost_function: evaluate $C(x)$ and first two derivatives

```
Data: Network graph $G(\mathcal{V}, \mathcal{A})$
      Step size $\alpha$
      Current link flows $\boldsymbol{f}$
      All-or-nothing link flows $\boldsymbol{f}^{AON}$
      Quasi-Newton step size $\Delta x$
Initialize $\boldsymbol{y}_k = [0, 0, 0]$;
Let $\mathcal{A}_k$ = this thread's partition of network links (for thread $k$);
Let $C(x) =$ Cost_function$\Big(\boldsymbol{f} + x \cdot (\boldsymbol{f}^{AON} - \boldsymbol{f})\Big)$ ;
for $a \in \mathcal{A}_k$ do
    $\boldsymbol{y}_k \mathrel{+}= [C_a(x - \Delta x), C_a(x), C_a(x + \Delta x)]$ ;
end
$\boldsymbol{y} = [C(x - \Delta x), C(x), C(x + \Delta x)] =$ Global all-reduce$(+, \boldsymbol{y}_k)$ ;   // Reduce values simultaneously
$C'(x) = \frac{C(x+\Delta x) - C(x-\Delta x)}{2\Delta x}$ ;
$C''(x) = \frac{C(x-\Delta x) - 2C(x) + C(x+\Delta x)}{\Delta x^2}$ ;
Result: $[C(x), C'(x), C''(x)]$
```



(a) (b) (c)

Figure 2: a) Maximum and average number of Quasi-Newton iterations to conduct the line search over all gradient descent iterations within each time segment for the San Francisco Bay Area network model. Comparison of b) the number of gradient descent iterations and c) the total segment compute time required using the method of successive averages (MSA, blue circles) versus using a optimal step size via Quasi-Newton method line search (orange squares). Data is for the San Francisco Bay Area network model, and execution times are measured when running on the Cori computer with 512 cores.

order to avoid the communication overhead of three separate global reductions for each of the three evaluations, our implementation simultaneously reduces all three values with a single vectorized all-reduce operation. The derivatives are then estimated using the approximations in Equation 10. Due to the efficient parallel evaluation of $C(x)$ and its derivatives, we observed that even for the cases with the highest number of Quasi-Newton iterations (see Figure 2a), the line search completes in less than 500 milliseconds using 512 cores of the Cori supercomputer (see Section 5.4 for details on Cori).

We observed the parallelized line search improves the performance of the QDTA compared to using the method of successive averages (MSA) for time segments with high levels of congestion. Figures 2b and 2c show the impact of using a line search versus MSA on number of gradient descent iterations and total time to solution for each time segment in our experiment on the San Francisco Bay Area network (see Section 5.4 for details). The line search and MSA are roughly equivalent for most time segments with low congestion, while there is a significant improvement for the most congested time segments during the morning peak. Furthermore, the segment compute times are highly correlated with the number of gradient descent iterations primarily because of the expensive all-or-nothing routing step required for each iteration. By finding the optimal step size for each iteration, the number of iterations required to converge is reduced, resulting in an overall savings of 16 percent in total execution time compared to MSA.

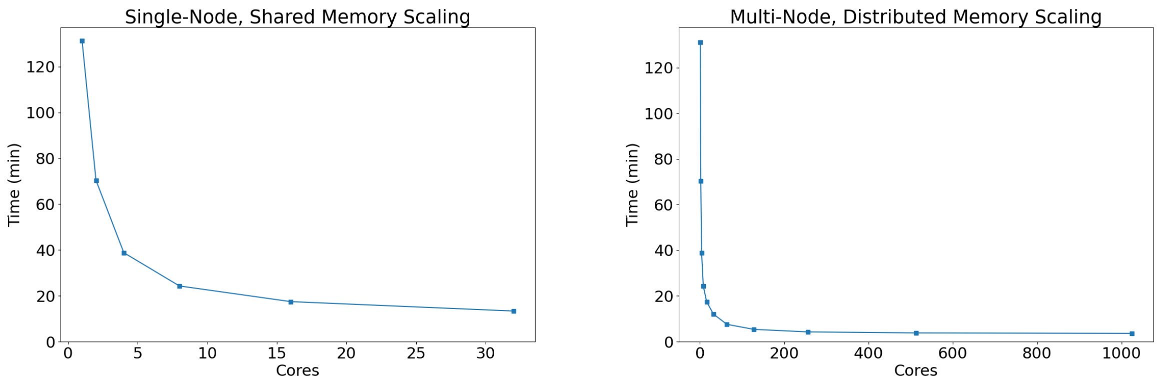
5.3 Network update and residual demand parallelization

Once the line search has converged and the optimal step size $\alpha^\star$ has been identified, the link weights for the network must be updated in every process. This step is parallelized across threads by assigning each thread a partition of the links to update the flow values and compute the new weights associated with its partition of links. Because each process must update all of the links in the network for the subsequent routing step to perform correctly, the degree of parallelism in this step is reduced to the number of threads within each process (as opposed to across all processes). This step corresponds to the upper blue boxes in Figure 5.

Finally, after the Frank-Wolfe algorithm has converged, the residual demand allocation must be performed to forward residual vehicles into the next QDTA time segment (Algorithm 4). The parallelization strategy for this algorithm partitions the traffic demand across threads in the exact manner as Algorithm 7. Each thread iterates over its partition ($\boldsymbol{d}_k$) and forwards the demand that is still in transit at the end of the current time segment into the next time segment. Once the residual demand is computed, the non-residual demand in the next time segment is identified and combined with the residual demand in parallel, and the Frank-Wolfe optimization for the next time segment begins. As we have described, the majority of the algorithms required for the QDTA methodology (as shown in Figure 1) have been parallelized to achieve high performance on distributed memory computer platforms.

5.4 Evaluation of computational performance and parallel scalability

We use high performance computing to address the computational challenges of a traffic assignment at urban scale. The solutions are implemented on the Cori supercomputer, a Cray X40 at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. In order to evaluate the computational *strong scaling* performance (improvement in program execution time for a fixed input as the number of cores is increased), we



(a) (b)

Figure 3: Computational scaling performance for our QDTA approach as parallelism is increased.

ran the QDTA algorithm for the San Francisco Bay Area network, utilizing up to 32 nodes of Cori with 1,024 cores total (2 processes per node, 16 cores per process, 2 threads per core). As we described in Section 5.1, the computational advantage is realized through parallelization of the various algorithmic steps in Fig. 1 across many compute cores. We segmented the 24-hour day into 96 time segments, each 15 minutes (versus the multiple hour time segments that are traditionally used). For all runs, we solved for optimized route plans for all 19 million vehicle trips that constitute the demand $\{\boldsymbol{d}^o(i) \mid \forall i\}$ over the San Francisco Bay Area network (0.5 million nodes and 1 million links). Refer to Section 7 for more details about the input network and demand.

Figure 3 shows the strong scaling performance results: a) when increase the number of cores on a single node, and b) when scaling across multiple nodes. Running the 96 time segment QDTA on a single node, the execution time is reduced from more than two hours (131 minutes) when run on a single core to under 14 minutes utilizing all 32 cores. This represents a parallel speedup of about 10x compared to single-core execution. Running the QDTA on multiple nodes, we see that the execution time is reduced further to under 4 minutes when run on 32 nodes (1,024 cores), representing a total speedup of over 36x compared to the single core case. The reason the speedup is sub-linear (less than N times speed up when using N times the compute cores) is due to a combination of overhead of parallelization (communication and synchronization costs between cores and processes) and the parts of the code which are not fully parallelized (resulting in Amdahl bottlenecks [65]). The most notable parts that not fully parallelized are the computational routines that update the network flows and weights after each gradient descent iteration. Because the entire network state must be replicated in each process to parallelize the expensive routing step, a fully parallel network update would require a global reduction of state the size of the entire network (millions of link weights). To avoid this reduction, the update is partially parallelized across threads within each process. Furthermore, the customization preprocessing step of the multi-phase routing library RoutingKit [63] has not been parallelized.

6 Simple Network Example

To help illustrate the QDTA model, we describe its application to a simple network with four links and five nodes and compare it with STA. For simplicity, all links are connected serially and no route choice decisions are involved in the network. The link free flow travel times and capacities are given in Table 3. We consider the demand for one hour with four time periods with 15 minutes interval. The demand is given in Table 4. The volume delay function considered is of BPR form as shown in Equation (1).

Table 3: Network attributes for the example

Link	Free flow travel time(min)	Capacity(v/h)
l_{12}	10	200
l_{23}	5	150
l_{34}	10	200
l_{45}	10	200

Table 4: Demand used in the example

Time	OD	Demand(v/h)
t_1	d_{14}	175
t_2	d_{35}	50
t_3		0
t_4		0

As shown in Fig. 4,for time t_1, demand d_{14} traverses link l_{12} and reaches node 2, but it cannot reach its destination in the same time interval and hence stored in the downstream node 3 as residual. In the next time period t_2, the new demand d_{35} and the residual from previous time segment will traverse link l_{34}. All demand reaches its destination in time segment t_2. Time periods t_3 and t_4 have no demand and hence all trips reaches destination in the first two time interval. The total system travel time is: $(175 * 10/60) + (175 * 6.3/60) + (225 * 12.4/60) + (50 * 10/60) = 94$. The congested links in the network are l_{23}, l_{34} during time periods t_1 and t_2 respectively.

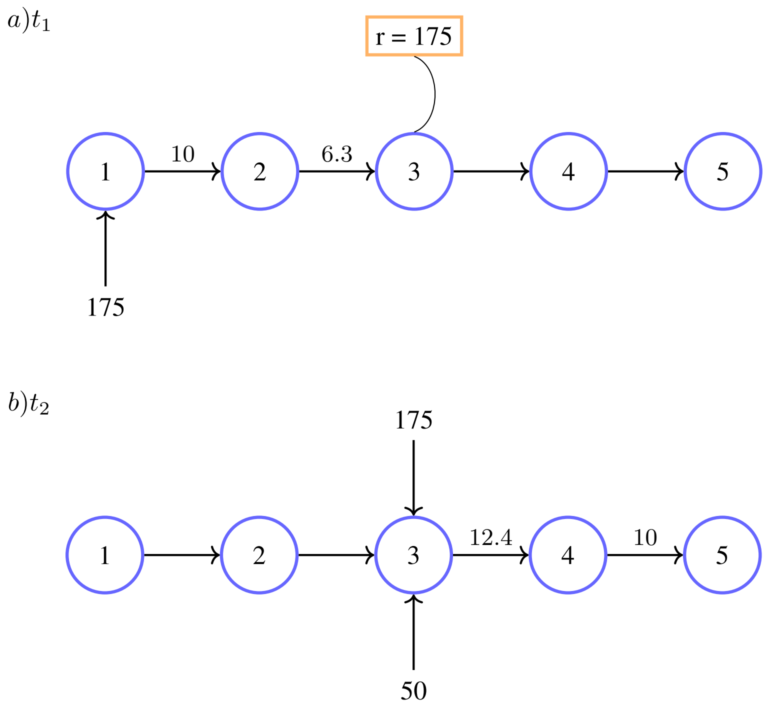


Figure 4: QDTA assignment for simple network. The network is loaded in 2 time segments. r represents the residual stored in the node at the end of each time period. The link travel time in minutes is noted above each link.

The same demand is assigned using STA as shown in Fig. 5. The demand is assigned as averaged over the entire 1 hour duration. The total system travel time is: $(44 * 10/60) + (44 * 5/60) + (57 * 10/60) + (13 * 10/60) = 22$. None of the links are congested during any time interval in this assignment because the STA spreads the demand over the entire one-hour analysis period and is unable to capture the peak congestion that occurs on link l_{34}. On the other hand, the QDTA algorithm correctly captures the peak loading of l_{23} in t_1 as well as the residual demand resulting from d_{14} that loads link l_{34} at the same time that the demand d_{35} enters the network in t_2. We will next describe how the QDTA is able to capture similar peak loading effects over a much larger network in the following section.

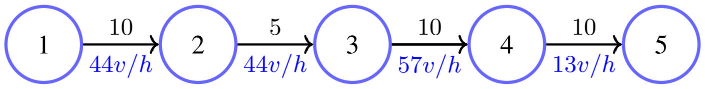


Figure 5: STA assignment for simple network. Full demand is assigned at the same time and gets averaged for the entire 1 hour time duration. The link travel time in minutes and link flows in vehicles per hour is noted above and below respectively.

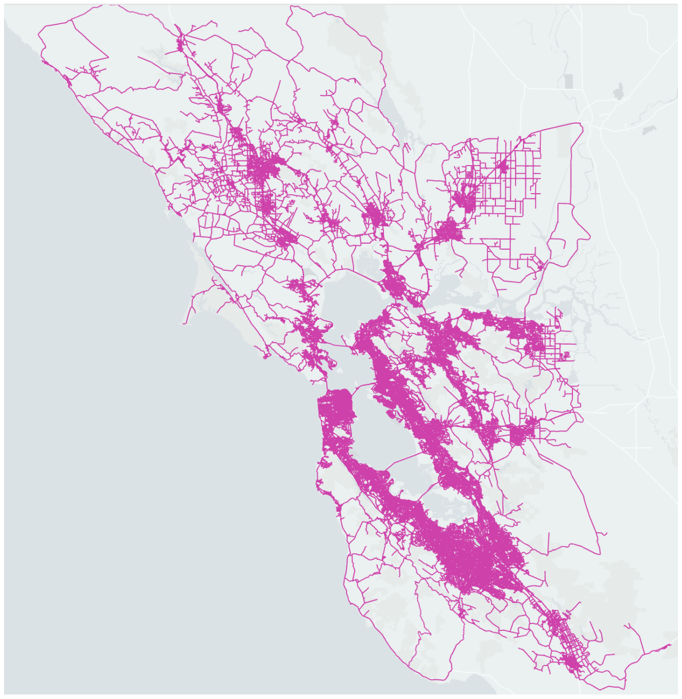
7 Application to San Francisco Bay Area Network

7.1 Analysis of Traffic Assignment Results

In this section, we present an analysis of the results from applying our QDTA methodology to a large-scale urban transportation system: the San Francisco Bay Area. For our network representation, we utilize a professional map from HERE Technologies [66]. The HERE map divides the network links into several functional class categories, depending on their characteristics. Specifically, functional classes classify roads according to the speed, importance and connectivity of the road. A road can be one of five functional classes – these are defined in Table 5. The analysis presented here will use these functional classes to help explore the results that compare QDTA to STA. The road network for San Francisco is shown in Fig. 6a and accounts for ∼0.5M nodes and ∼1M links ([66]). We have also successfully applied our QDTA approach to the Los Angeles road network with ∼1M nodes and ∼2M links with ∼40M trips in a 24 hour period. For brevity, we will present results for the Bay Area only.

Table 5: Functional Road Classes

Functional Class	Definition
1	Allowing for high volume, maximum speed traffic movement
2	Allowing for high volume, high speed traffic movement
3	Providing a high volume of traffic movement
4	Providing for a high volume of traffic movement at moderate speeds between neighbourhoods
5	Roads whose volume and traffic movement are below the level of any other functional class



(a) Bay Area Network ∼1M Links

Figure 6: Road Network for the SF Bay Area

We obtained demand data from the SFCTA CHAMP 6 [67] agent-based time-dynamical model for the Bay Area network, which uses data such as observed travel patterns, representations of the transportation system, and population and employment data to synthesize ∼19 million trips during a 24-hour period. Each trip in the CHAMP model was identified with an origin and destination micro-analysis zone, which was then assigned to specific network nodes by weighting with the population density obtained from Global Human Settlement [68].

For our analysis of QDTA, we evaluate two traffic assignment models for the San Francisco Bay Area. We compare a static traffic assignment (STA) that assigns all traffic in the 7-10 am morning peak in a single time interval, while the QDTA model divides the same period into twelve 15-minute intervals. Both cases solve for a user equilibrium traffic assignment within each time segment. Fig. 7 shows the demand profile for both cases under consideration. While STA assumes a constant demand for the entire morning peak period, QDTA enables modeling a variable demand for the same duration.

Table 6 provides a comparison of the system level metrics using vehicle miles travelled (VMT), average volume over capacity (VOC) ratios, and vehicle hours of delay (VHD) categorised by functional class. Only links with positive flow are included in the analysis. Additionally, Table 7 provides a comparison between STA and QDTA for the **congested**

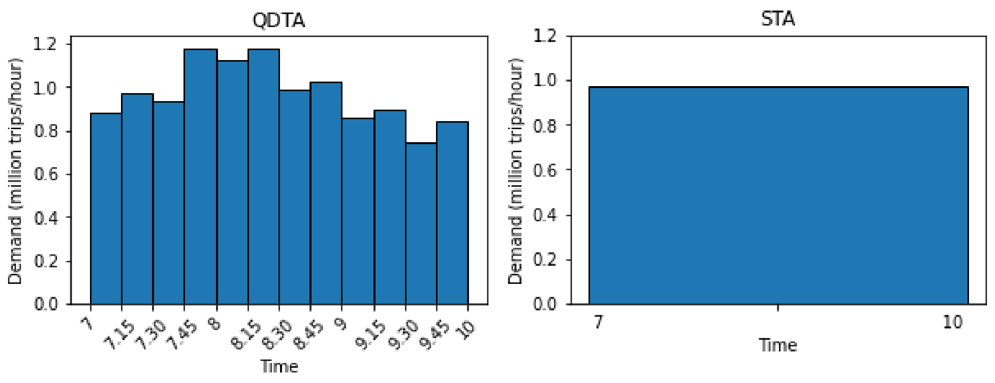


Figure 7: Temporal demand interaction and model capabilities

portion of the network, where a link is considered congested if its volume over capacity ratio is greater than or equal to 1.0. For QDTA analysis, congestion is calculated for 7.45-8 AM time period when the demand is the highest.

Table 6: System Level Metrics (all roads)

Category	VMT (in millions)		Average VOC		VHD (in thousands)	
	STA	QDTA	STA	QDTA	STA	QDTA
FC2	16.44	18.11	0.52	0.69	17.57	96.84
FC3	4.67	5.01	0.23	0.30	2.67	14.43
FC4	4.96	5.25	0.14	0.17	1.13	6.42
FC5	2.43	2.60	0.03	0.48	0.38	17.78
Total	28.51	30.98	-	-	21.95	135.49

Table 7: Congested Network Metrics (roads with VOC >= 1.0)

Category	Length (km)		VMT (in millions)		Average VOC	
	STA	QDTA	STA	QDTA	STA	QDTA
FC2	131	553	1.77	7.56	1.15	1.26
FC3	38	88	0.17	0.42	1.18	1.28
FC4	14	56	0.04	0.15	1.19	1.21
FC5	5	28	0.01	0.06	1.17	1.25
Total	188	725	2.0	8.19	-	-

The key distinction between the two models is how they estimate the variation and magnitude of network congestion patterns. As seen in Table 7, QDTA predicts more congested roadway (length), and higher VMT and average VOC on the congested roadways compared to STA. These data are in accordance with our expectation that non-uniform demand distributions result in higher travel times due to the convexity properties associated with link performance functions. Furthermore, STA (by definition) is not able to recreate any temporal congestion dynamics, whatsoever. Fig. 8 shows the VOC over time by functional class (left) and VOC comparison for the two models by functional class (right). For all classes of roads the QDTA predicts congestion dynamically over time whereas STA either overestimates or underestimates the values irrespective of the demand dynamics. This difference is highly pronounced for FC2 where STA significantly underestimates the peak hour congestion. As FC2's are high capacity highways and freeways, static average demand is not able to capture the evolving congestion dynamics which is captured by QDTA.

It can also be seen from Table 6 that the total system delay is significantly higher in QDTA versus STA. This is due to the dynamic demand distribution and the modeling capability in QDTA that allows interaction of demand between time intervals, thus allowing for a more realistic modeling of traffic. Fig. 9 shows the dynamic variation in the number of trip legs over time in 15 minute intervals from 7am to 10am. For STA, the system is modeled as if the demand (3.5M trips whose start times are within the 7-10am range) are spread evenly across all three hours, resulting in an average of about 300k trips per 15 minute interval. As described in Section 4, for QDTA the total number of trips modeled in an interval is composed of two parts: the original (existing) demand $\boldsymbol{d}^o(i)$ and the residual demand $\boldsymbol{d}^r(i)$ carried over from previous intervals. While the sum of the existing and residual trips appears to be much larger than the static demand case, in reality each trip only contributes a fraction of its entire route to each time segment due to the route

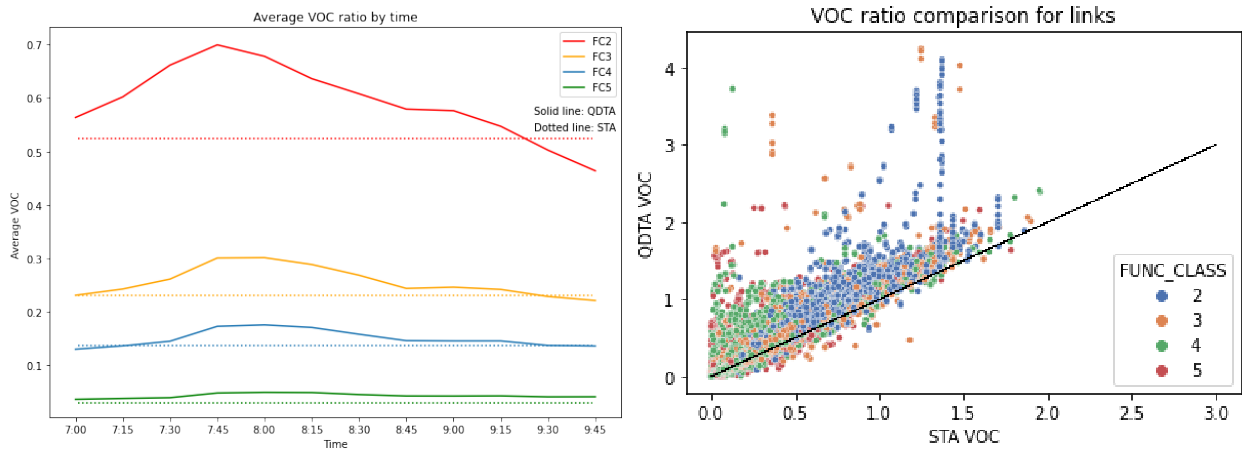


Figure 8: Congestion profiling for QDTA and STA. Figure on the left shows the average VOC ratio for all links over time categorised by functional classes. The figure on right is a scatter plot showing the VOC ratio comparison for every link in the network for the 2 models at 7:45-8 am.

truncation mechanism. For trips that span multiple time intervals, different (non-overlapping) segments of the trip's path are loaded onto network in each time interval. Thus, in QDTA each trip leg only contributes flow to the links it traverses within the time segment, whereas for STA every leg contributes flow to the entire route. For the time interval 7.45-8 AM, STA has 188 km of congested network in comparison to QDTA with 725 km of congested network. Fig. 10 shows the Bay area network with congestion locations for the two modeling cases.

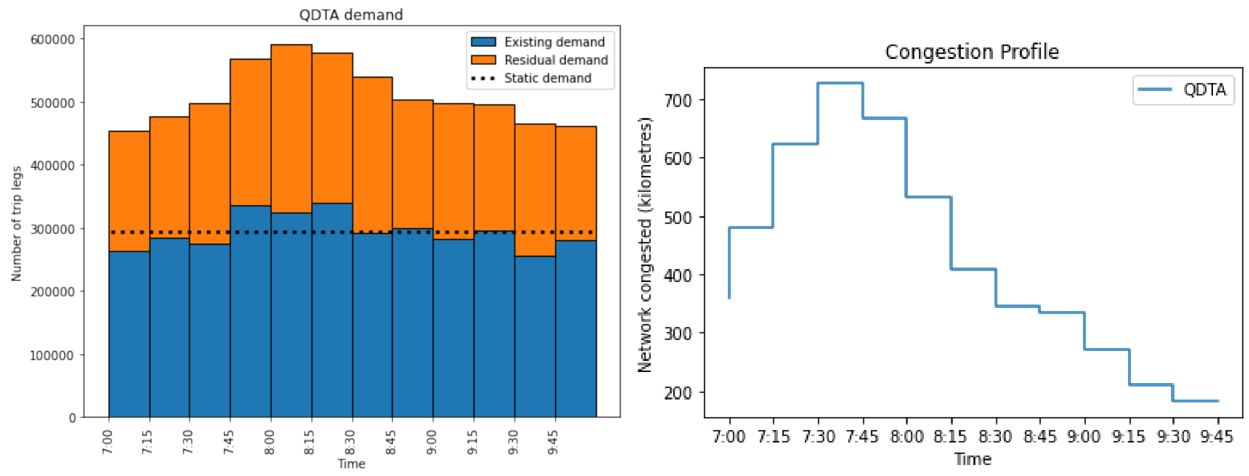


Figure 9: QDTA demand modeling with residuals (left). The length of network congested over time. A link is considered congested if the VOC ratio is greater than or equal 1.

7.2 Validation

Validation was performed for QDTA results to test the effectiveness of representing the real world traffic environment. We conducted validation for traffic volume, speed and system metrics using multiple data sources. Stage 1 of the validation procedure involves checking the traffic counts for eight link corridors. The traffic count for each link was compared against the field data for the entire day in 15 minutes increments. The field data for city roads and highways were collected from the city of San Jose and Caltrans PeMS website [69] respectively for the year 2019. Each corridor provided information regarding traffic volumes and speeds by time of the day and direction. A coefficient of determination (R^2) of 0.7 is typically used as a satisfactory criterion for link count checks. The Fig. 11 shows R^2 values for the eight corridors under consideration. The modeled corridors indicate a close match with the field data with the lowest R^2 value observed being 0.67 for Zanker Road.

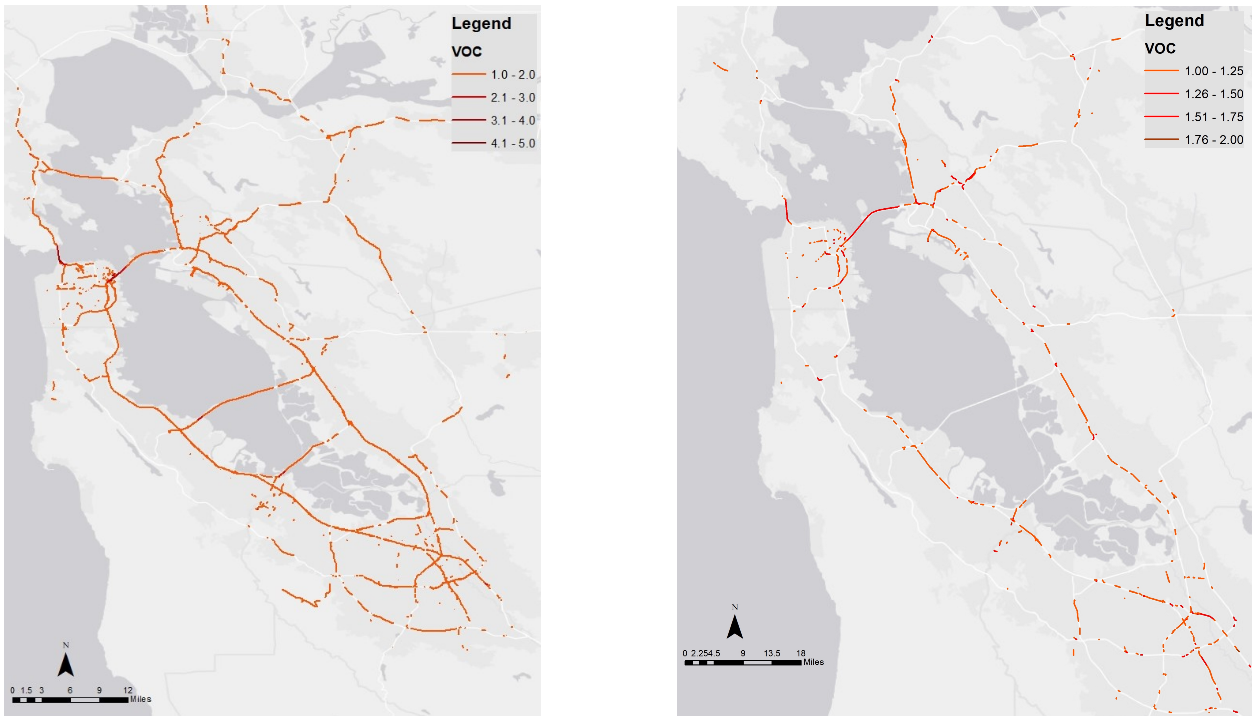


Figure 10: VOC ratios for congested links for QDTA (left) and STA (right) is shown for time period 7.45 - 8 AM. The links that are predicted to be congested in STA has even higher congestion in QDTA. STA produces 188 km of congested network in comparison to QDTA with 720 km of congested network. Only links with VOC greater than or equal to 1 is shown.

Stage 2 is speed comparison with Uber Movement speed data for San Francisco region for Q4, 2019 [70]. Links from Uber network were matched to our network for 139,495 links (20% of total). The speeds were compared for 8 am to 9 am for different speed limits. Figure 12 shows the speed distributions from QDTA results and Uber on links with 60 mph and 70 mph speed limit. Figure 13 shows the average speeds from Mobiliti and Uber across all speed limits. We believe the observed discrepancies in the speed distributions may be improved in future work with the addition of real world traffic signal location and timing data and further refinement of the link flow congestion and timing models.

Final stage of validation includes system level metrics comparisons, network validation, and error checking. Model visualization is used to check for unusual activities in traffic flows and odd roadway network attributes. Error checking and model verification consist of several smaller tasks such as checks for link geometry and connectivity, number of lanes, speeds, ramps and intersection geometry. Since our travel demand data was obtained from SFCTA, which conducts their own validation, we did not conduct additional behavior checks. We conducted system metric checks for VMT and total demand and validated them against the 2017 Environmental Impact report for the Bay Area [71] in Table 8.

Table 8: System level metrics validation

Metric	**QDTA**	**Field Data**	**Relative Error(%)**
VMT	150,634,403	158,406,800	-4.9
Daily Trips	19,167,301	21,227,800	-10

8 Conclusions

In this paper, we have presented a quasi-dynamic traffic assignment methodology to capture temporal dynamics in large-scale transportation networks and described its parallelization to run efficiently on distributed-memory high performance computing systems. Two key mechanisms, route truncation and residual demand, were implemented to provide more realistic demand profiles as the dynamic assignment interval is reduced and a greater percentage of vehicle trips span across multiple time segments. Our approach divides the simulated day into 15 minute intervals

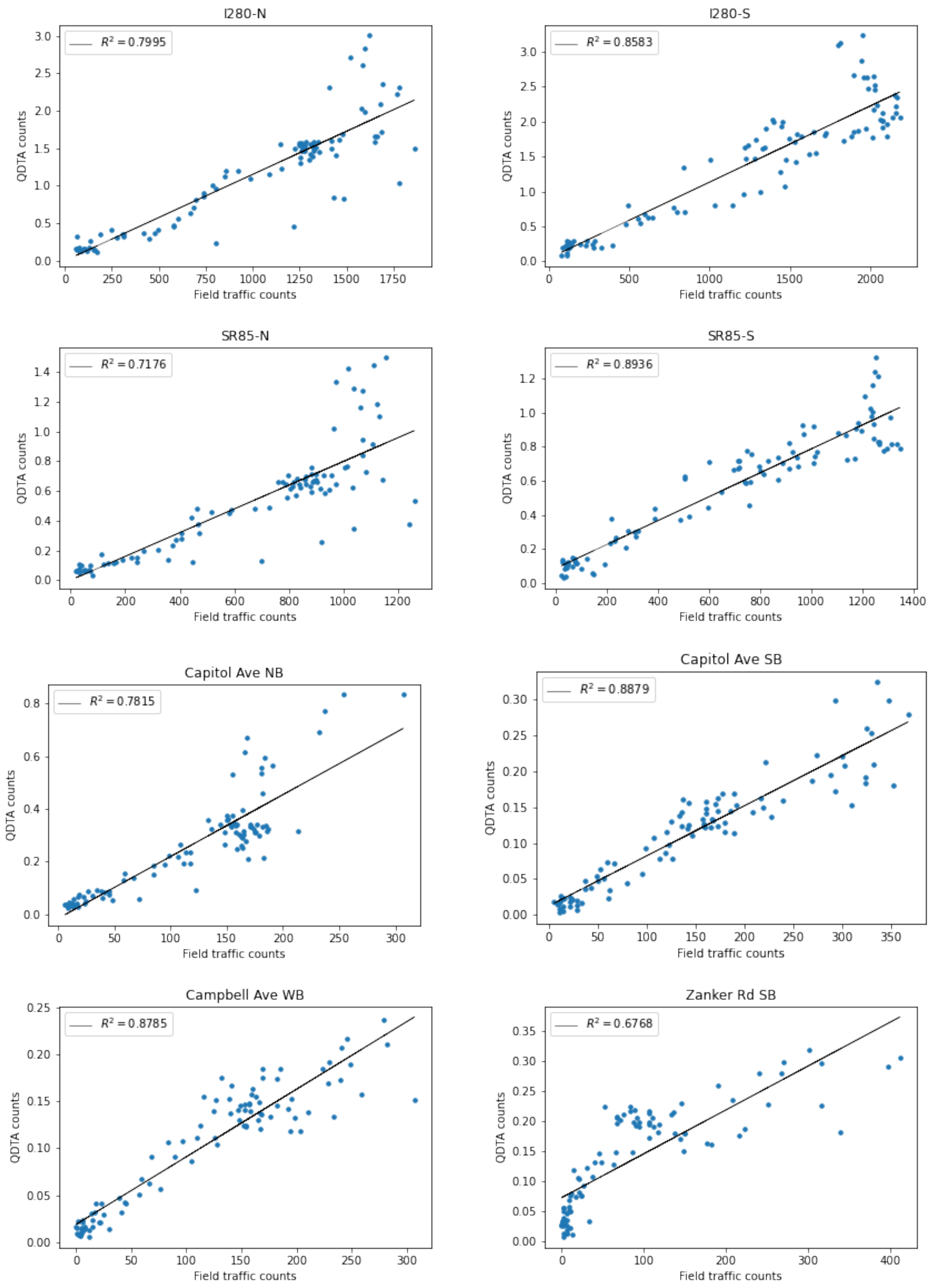


Figure 11: Validation of traffic counts in 15 minute increments for links in different functional classes. All links except one have satisfactory R^2 of greater than 0.7

and utilizes a modified static traffic assignment within each interval to assign the active traffic present in the network. The QDTA assignment step differs from a traditional STA in that the assigned routes are truncated to fit in the active time interval in each Frank-Wolfe iteration so that the resulting solution only includes traffic that occurs within the current interval. Furthermore, residual demand for each time interval is calculated based on estimated travel time on

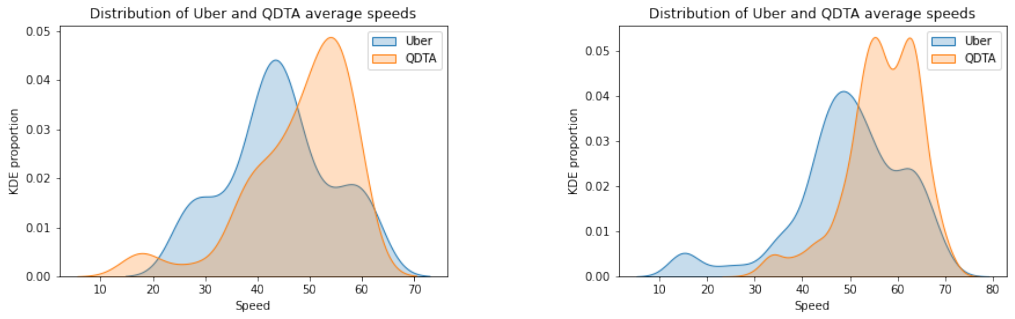


Figure 12: Kernel density plot comparing QDTA model and Uber speed distributions at 60 mph (left) and 70 mph (right).

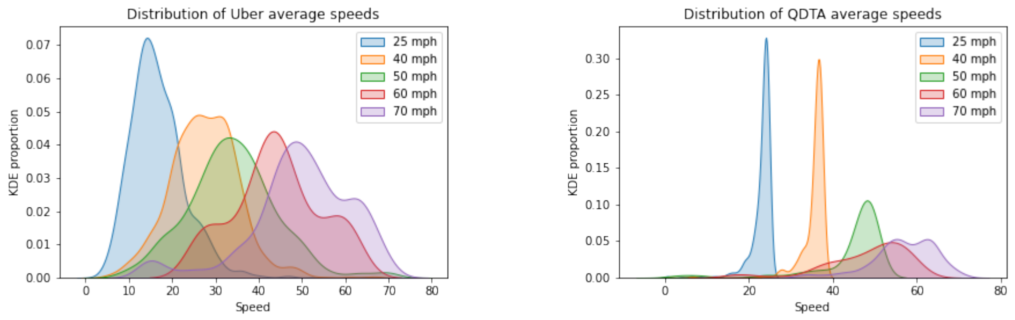


Figure 13: Distribution of average Uber (left) and QDTA model (right) speeds across specific speed limits between 8-9am in the morning shown using a kernel densities.

the links and then carried over to the next time interval. The combination of these techniques resolves traffic that is resident in the network for a longer time horizon than a single static assignment period and captures dynamic network behavior across multiple time segments. The model can be interpreted as an time expanded network approach where the expansion of the network represents the evolution with respect to time.

We have also described how the quasi-dynamic traffic assignment algorithm can be parallelized for efficient execution on high-performance distributed-memory computing platforms. The algorithm is parallelized through a combination of partitioning trip legs and network links across available compute threads to speed up the calculation of shortest paths, optimization cost functions, residual trip legs, and other program data. We described our parallelized line search algorithm which enables the identification of the optimal step size for each gradient descent iteration using Newton's method, taking less than 500 milliseconds for a network of 1 million links. Using the optimal step size provides a reduction of 16 percent in total execution time compared to using the method of successive averages, due to a decrease in gradient descent iterations required for convergence. We demonstrated that a quasi-dynamic traffic assignment of the San Francisco Bay Area (19 million trip legs, 0.5 million nodes, and 1 million links) using 96 15-minute time segments runs in under 4 minutes on 1,024 cores of the Cori supercomputer at NERSC, corresponding to a speedup of 36x compared to single core performance.

Finally, we presented an analysis of the traffic assignment results across functional classes, illustrating how the QDTA more accurately resolves the increased congestion patterns and dynamic behavior of the traffic system compared to a static traffic assignment approach, especially under peak congestion conditions. We presented a validation of the QDTA assignment counts and speeds compared to field data from CalTrans, San Jose, and Uber Movement, showing field count correlation values of 0.68 or greater. Future work includes evaluating a variety of optimization objective functions, include fuel optimization and system level optimizations for both travel time and fuel use. The QDTA results can be integrated into our computational platform Mobiliti [72], which also provides parallel discrete event traffic simulations for urban-scale regions to conduct hypothetical demand, infrastructure, and advanced traffic control scenario evaluations. We hope to also develop surrogate models using the results from the HPC implementation such that this capability can be made more widely available.

Acknowledgements

This report and the work described were sponsored by the U.S. Department of Energy (DOE) Vehicle Technologies Office (VTO) under the Big Data Solutions for Mobility Program, an initiative of the Energy Efficient Mobility Systems (EEMS) Program. The following DOE Office of Energy Efficiency and Renewable Energy (EERE) managers played important roles in establishing the project concept, advancing implementation, and providing ongoing guidance: David Anderson and Prasad Gupte. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] N. Tsanakas, J. Ekström, and J. Olstam. Estimating emissions from static traffic models: Problems and solutions. ISSN: 0197-6729 Pages: e5401792 Publisher: Hindawi Volume: 2020.

[2] M. G. H. Bell and Y. lida. *Transportation Networks*, chapter 2, pages 17–40. John Wiley and Sons, Ltd, 1997.

[3] S. C. A. O. GOVERNMENTS. Scag regional travel demand model and 2012 model validation. Technical report, SOUTHERN CALIFORNIA ASSOCIATION OF GOVERNMENTS, 2016.

[4] S. Flügel and G. Flötteröd. Traffic assignment for strategic urban transport model systems.

[5] M. Bliemer, M. Raadsen, E. de Romph, and E.-S. Smits. Requirements for traffic assignment models for strategic transport planning: A critical assessment. page 25.

[6] T. Iryo. Multiple equilibria in a dynamic traffic network. 45(6):867–879.

[7] Genetics of traffic assignment models for strategic transport planning. 37:56–78, January 2017.

[8] M. C. Bliemer, M. P. Raadsen, E.-S. Smits, B. Zhou, and M. G. Bell. Quasi-dynamic traffic assignment with residual point queues incorporating a first order node model. *Transportation Research Part B: Methodological*, 68:363–384, 2014.

[9] H. Tajtehranifard, A. Bhaskar, N. Nassir, M. M. Haque, and E. Chung. A path marginal cost approximation algorithm for system optimal quasi-dynamic traffic assignment. *Transportation Research Part C: Emerging Technologies*, 88:91–106, 2018.

[10] G. Fusco, C. Colombaroni, A. Gemma, and S. L. Sardo. A quasi-dynamic traffic assignment model for large congested urban road networks. *International Journal of Mathematical Models and Methods in Applied Sciences*, 7(4):341–349, 2013.

[11] S. Nakayama and R. Connors. A quasi-dynamic assignment model that guarantees unique network equilibrium. *Transportmetrica A: Transport Science*, 10(7):669–692, 2014.

[12] X. Zeng, X. Guan, H. Wu, and H. Xiao. A data-driven quasi-dynamic traffic assignment model integrating multi-source traffic sensor data on the expressway network. *ISPRS International Journal of Geo-Information*, 10(3):113, 2021.

[13] H.-K. Chen. *Dynamic travel choice models: a variational inequality approach*. Springer Science & Business Media, 2012.

[14] A. Nagurney and D. Zhang. *Projected dynamical systems and variational inequalities with applications*, volume 2. Springer Science & Business Media, 2012.

[15] B. Ran and D. Boyce. *Dynamic urban transportation network models: theory and implications for intelligent vehicle-highway systems*, volume 417. Springer Science & Business Media, 2012.

[16] D. Merchant and G. Nemhauser. A model and an algorithm for the dynamic traffic assignment problems. *Transportation science*, 12(3):183–199, 1978.

[17] D. Merchant and G. Nemhauser. Optimality conditions for a dynamic traffic assignment model. *Transportation Science 12.3*, 12(3):200–207, 1978.

[18] B. Ran and T. Shimazaki. A general model and algorithm for the dynamic traffic assignment problems. In *Transport Policy, Management & Technology Towards 2001: Selected Proceedings of the Fifth World Conference on Transport Research*, volume 4, 1989.

[19] B. Ran, D. Boyce, and L. LeBlanc. A new class of instantaneous dynamic user-optimal traffic assignment models. *Operations Research*, 41(1):192–202, 1993.

[20] T. Friesz, J. Luque, R. Tobin, and B.-W. Wie. Dynamic network traffic assignment considered as a continuous time optimal control problem. *Operations Research*, 37(6):893–901, 1989.

[21] B. Ran and D. Boyce. A link-based variational inequality formulation of ideal dynamic user-optimal route choice problem. *Transportation Research Part C: Emerging Technologies*, 4(1):1–12, 1996.

[22] D. Boyce, B. Ran, and L. Leblanc. Solving an instantaneous dynamic user-optimal route choice model. *Transportation Science*, 29(2):128–142, 1995.

[23] T. Friesz, D. Bernstein, T. Smith, R. Tobin, and B.-W. Wie. A variational inequality formulation of the dynamic network user equilibrium problem. *Operations Research*, 41(1):179–191, 1993.

[24] A. Bayen, A. Keimer, E. Porter, and M. Spinola. Time-continuous instantaneous and past memory routing on traffic networks: A mathematical analysis on the basis of the link-delay model. *SIAM Journal on Applied Dynamical Systems*, 18(4):2143–2180, 2019.

[25] S. Peeta and T.-H. Yang. Stability issues for dynamic traffic assignment. *Automatica*, 39(1):21–34, 2003.

[26] R. Ma, X. Ban, and J.-S. Pang. Continuous-time dynamic system optimum for single-destination traffic networks with queue spillbacks. *Transportation Research Part B: Methodological*, 68:98–122, 2014.

[27] X. Ban, J.-S. Pang, H. Liu, and R. Ma. Modeling and solving continuous-time instantaneous dynamic user equilibria: A differential complementarity systems approach. *Transportation Research Part B: Methodological*, 46(3):389–408, 2012.

[28] K. Han, T. Friesz, and T. Yao. A partial differential equation formulation of vickrey's bottleneck model, part i: Methodology and theoretical analysis. *Transportation Research Part B: Methodological*, 49:55 – 74, 2013.

[29] K. Han, T. Friesz, and T. Yao. A partial differential equation formulation of vickrey's bottleneck model, part ii: Numerical analysis and computation. *Transportation Research Part B: Methodological*, 49:75 – 93, 2013.

[30] K. Han, T. Friesz, and T. Yao. Existence of simultaneous route and departure choice dynamic user equilibrium. *Transportation Research Part B: Methodological*, 53:17 – 30, 2013.

[31] A. Bressan and K. Nguyen. Optima and equilibria for traffic flow on networks with backward propagating queues. *NHM*, 10(4):717–748, 2015.

[32] A. Bressan and K. Nguyen. Conservation law models for traffic flow on a network of roads. *NHM*, 10(2):255–293, 2015.

[33] M. Garavello, K. Han, and B. Piccoli. *Models for vehicular traffic on networks*, volume 9. American Institute of Mathematical Sciences (AIMS), Springfield, MO, 2016.

[34] M. Garavello and B. Piccoli. *Traffic flow on networks*, volume 1. American institute of mathematical sciences Springfield, 2006.

[35] H. Holden and N. Risebro. A mathematical model of traffic flow on a network of unidirectional roads. *SIAM Journal on Mathematical Analysis*, 26(4):999–1017, 1995.

[36] G. Bretti, R. Natalini, and B. Piccoli. Numerical approximations of a traffic flow model on networks. *NHM*, 1(1):57–84, 2006.

[37] M. Lighthill and G. Whitham. On kinematic waves. i. flood movement in long rivers. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 229(1178):281–316, 1955.

[38] P. I. Richards. Shock waves on the highway. *Operations research*, 4(1):42–51, 1956.

[39] A. Keimer, N. Laurent-Brouty, F. Farokhi, H. Signargout, V. Cvetkovic, A. Bayen, and K. Johansson. Information patterns in the modeling and design of mobility management services. *Proceedings of the IEEE*, 106(4):554–576, 2018.

[40] M. Gugat, A. Keimer, G. Leugering, and Z. Wang. Analysis of a system of nonlocal conservation laws for multi-commodity flow on networks. *Networks and Heterogeneous Media*, 10(4):749–785, 2015.

[41] A. Keimer, L. Pflug, and M. Spinola. Nonlocal scalar conservation laws on bounded domains and applications in traffic flow. *accepted in SIAM SIMA*, 2018.

[42] A. Aw and M. Rascle. Resurrection of "second order" models of traffic flow. *SIAM Journal on Applied Mathematics*, 60(3):916–938, 2000.

[43] H. Zhang. A non-equilibrium traffic model devoid of gas-like behavior. *Transportation Research Part B: Methodological*, 36(3):275 – 290, 2002.

[44] M. Ghali and M. Smith. A model for the dynamic system optimum traffic assignment problem. *Transportation Research Part B: Methodological*, 29(3):155–170, 1995.

[45] S. Peeta and H. Mahmassani. System optimal and user equilibrium time-dependent traffic assignment in congested networks. *Annals of Operations Research*, 60(1):81–113, 1995.

[46] M. Smith. A new dynamic traffic model and the existence and calculation of dynamic user equilibria on congested capacity-constrained road networks. *Transportation Research Part B: Methodological*, 27(1):49–63, 1993.

[47] S. Waller and A. Ziliaskopoulos. A chance-constrained based stochastic dynamic traffic assignment model: Analysis, formulation and solution algorithms. *Transportation Research Part C: Emerging Technologies*, 14(6):418–427, 2006.

[48] H.-K. Chen and C.-F. Hsueh. A model and an algorithm for the dynamic user-optimal route choice problem. *Transportation Research Part B: Methodological*, 32(3):219–234, 1998.

[49] J. Long, W. Szeto, H.-J. Huang, and Z. Gao. An intersection-movement-based stochastic dynamic user optimal route choice model for assessing network performance. *Transportation Research Part B: Methodological*, 74:182 – 217, 2015.

[50] B. Janson. Dynamic traffic assignment for urban road networks. *Transportation Research Part B: Methodological*, 25(2-3):143–161, 1991.

[51] B. Janson. Convergent algorithm for dynamic traffic assignment. *Transportation Research Record*, (1328), 1991.

[52] J. Ho. A successive linear optimization approach to the dynamic traffic assignment problem. *Transportation Science*, 14(4):295–305, 1980.

[53] R. Correa, I. de Castro Dutra, M. Fiallos, and L. F. G. da Silva. *Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications*, volume 67. Springer Science & Business Media, 2013.

[54] T. S. Systems. Aimsun. https://www.aimsun.com.

[55] I. o. T. S. SUMO, German Aerospace Center (DLR). http://www.sumo.dlr.de, 2018.

[56] S. Thulasidasan, S. Kasiviswanathan, S. Eidenbenz, E. Galli, S. Mniszewski, and P. Romero. Designing systems for large-scale, discrete-event simulations: Experiences with the fasttrans parallel microsimulator. In *2009 International Conference on High Performance Computing (HiPC)*, pages 428–437. IEEE, 2009.

[57] C. Sheppard et al. Modeling plug-in electric vehicle charging demand with beam, the framework for behavior energy autonomy mobility. Technical report, 05/2017 2017.

[58] J. Auld, M. Hope, H. Ley, V. Sokolov, B. Xu, and K. Zhang. Polaris: Agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations. *Transportation Research Part C: Emerging Technologies*, 64:101–116, 2016.

[59] W. Petprakob, L. Wijerathne, T. Iryo, J. Urata, K. Fukuda, and M. Hori. On the implementation of high performance computing extensionfor day-to-day traffic assignment. *Transportation research procedia*, 34:267–274, 2018.

[60] W. Himpe, R. Ginestou, and M. C. Tampère. High performance computing applied to dynamic traffic assignment. *Procedia Computer Science*, 151:409–416, 2019.

[61] M. Frank, P. Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[62] M. Patriksson. *The traffic assignment problem: models and methods*. Courier Dover Publications, 2015.

[63] J. Dibbelt, B. Strasser, and D. Wagner. Customizable contraction hierarchies. In *International Symposium on Experimental Algorithms*, pages 271–282. Springer, 2014.

[64] H. Sbayti, C.-C. Lu, and H. S. Mahmassani. Efficient implementation of method of successive averages in simulation-based dynamic traffic assignment models for large-scale network applications. *Transportation Research Record*, 2029(1):22–30, 2007.

[65] J. L. Gustafson. *Amdahl's Law*, pages 53–60. Springer US, Boston, MA, 2011.

[66] HERE Technologies. `https://www.here.com/`, 2019. [Online; accessed 06-Feb-2019].

[67] SFCTA. SF-CHAMP 6.1: ConnectSF Needs Assessment 2015 Base Year Model Run. Technical report, San Francisco County Transportation Authority, February 2019.

[68] European Commission, Joint Research Centre (JRC); Columbia University, Center for International Earth Science Information Network - CIESIN. Ghs population grid, derived from gpw4, multitemporal (2015). `http://data.europa.eu/89h/jrc-ghsl-ghs_pop_gpw4_globe_r2015a`, 2015. European Commission, Joint Research Centre (JRC) [Dataset].

[69] Caltrans, state of california. `https://dot.ca.gov/programs/traffic-operations/census/traffic-volumes`, 2019. [Online; accessed 03-January-2021].

[70] Uber movement. https://movement.uber.com/cities/san_francisco/downloads/speeds, 2019. [Online; accessed 03-January-2021].

[71] Environmental impact report plan bay area. https://www.planbayarea.org/2040-plan/environmental-impact-report, 2017. [Online; accessed 05-November-2020].

[72] C. Chan, B. Wang, J. Bachan, and J. Macfarlane. Mobiliti: scalable transportation simulation using high-performance parallel computing. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 634–641. IEEE, 2018.