B-Dynamic: An Efficient Algorithm for Dynamic User Equilibrium Assignment in Activity-Travel Networks

Gitakrishnan Ramadurai, and Satish Ukkusuri *

April 3, 2010

Abstract

Multi-dimensional choice in DTA - for example, a combined model of activity location, time of participation, duration, and route choice decisions - results in exponentially increasing choice alternatives. Any efficient algorithm for solving the multi-dimensional DTA problem must avoid enumeration of alternatives. In this paper an algorithm that does not enumerate paths is presented. The algorithm is a novel extension of Algorithm B (Dial 2006) to dynamic networks and hence referred to as Algorithm B-Dynamic. The DTA model proposed here uses a point queue model for traffic propagation that reduces computational complexity. The activity participation decision dimensions are incorporated through utility functions which are a linear function of duration and schedule delay (early or late arrival penalty). Numerical examples are then presented to illustrate both the steps of the algorithm and its' capabilities. Overall, the algorithm performed well for up to medium-sized networks. Further, the algorithm scales fairly well with increasing demand levels.

*The first author is affiliated with the Department of Civil Engineering, Indian Institute of Technology, Madras, Chennai 600032, India and the second author is affiliated with the School of Civil Engineering, Purdue University, West Lafayette, IN 47907, U.S.A. Part of the work was carried out when the authors were at the Department of Civil and Environmental Engineering, Rensselaer Polytechnic Institute, Troy, New York 12180, U.S.A. The email addresses of the authors are, respectively, gitakrishnan@iitm.ac.in, and sukkusur@purdue.edu.
1 Introduction

Dynamic traffic assignment models (Peeta and Ziliaskopoulos 2001) have been well studied over the past two decades. Initial DTA models primarily considered determining route choices and were subsequently extended to capture two choice dimensions - route and departure time choice (Friesz et al. 1993, Ran et al. 1996, Huang and Lam 2002, Szeto and Lo 2004, Zhang and Zhang 2007). Urban transport modeling involves several additional dimensions of individual choice including activity participation, location, time of participation, and duration. To capture behavioral realism better there is a need to consider these additional choice dimensions within a dynamic traffic assignment framework. In Ramadurai and Ukkusuri (2008), activity location, time of participation, duration, and route choice decisions were jointly modeled in a single unified representation framework referred to as Activity-Travel Networks (ATNs). ATNs are Supernetworks (Sheffi 1985) where activities are represented as links in the network. A major hurdle for extending the Supernetwork concept to dynamic networks considering activities is that the resulting multi-dimensional choice problem leads to combinatorially increasing choice dimensions. Therefore existing algorithms that depend on path enumeration such as route-swapping algorithm are difficult to implement even for moderately sized networks.

In this paper, an algorithm that does not require path enumeration is presented. The algorithm is a novel extension of Algorithm B (Dial 2006) to dynamic networks and hence referred to as Algorithm B-Dynamic. The rest of the paper is organized as follows: the literature in static traffic assignment algorithms is reviewed first followed by algorithms for solving dynamic traffic assignment (DTA), and DTA considering additional choice dimensions. Given the extensive literature on these topics and in the interest of length, the reviews are brief and illustrative; not comprehensive. The next section describes Dial’s Algorithm B in detail that provides the platform for Algorithm B-Dynamic, described next. Numerical examples are then presented to illustrate both the steps of the algorithm and its’ capabilities.
2 Literature Review

2.1 Algorithms for Static Traffic Assignment

The well accepted static traffic assignment formulation of Beckmann et al. (1956) has been solved using several different techniques. The most common behavioral frameworks in static traffic assignment are User Equilibrium (UE) and System Optimal (SO) (Wardrop 1952, Dafermos and Sparrow 1969). Broadly the algorithms may be classified as link based or path based. Link based solution algorithms store the flows in each link while path based solution algorithms store the flow in each path separately. The link flow is an aggregation of path flows that traverse a link. Since the number of links in a network are much lesser than the number of paths between any two O-D pairs, the link based solution algorithms are more memory efficient. For this reason, link based methods have been more widely accepted and implemented in practice.

Two of the most common link based solution algorithms for static traffic assignment problem are the Method of Successive Averages and the Frank-Wolfe method (Sheffi 1985). These algorithms iteratively improve the link flow solutions by solving a linearized problem of the original traffic assignment. While the major advantage of these solution algorithms is their lean memory requirements, it is well known they have poor convergence properties. At higher levels of accuracy, the tail of the convergence graph flattens out (Bar-Gera 1999).

In contrast to the link based solution algorithms, path based solution algorithms do not suffer from poor convergence and have been shown to obtain more precise solutions in lesser time. Path based algorithms were among the original methods proposed to solve the static traffic assignment problem. The overall concept of path based methods is to shift flows from more expensive paths to less expensive paths till all the paths are in equilibrium. The method was originally suggested by Dafermos (1968) and Dafermos and Sparrow (1969). Bertsekas (1976) proposed a method of shifting flow between independent path segments which was implemented in Jayakrishnan et al. (1994).

All of the above path based methods required the storage of paths across iterations. As mentioned
earlier, path storage requires substantial memory and in large networks may be infeasible. Dial (2006) proposed an algorithm (Algorithm B) that does not require explicit storage of paths. He makes use of acyclic sub-networks referred to as bushes to perform flow shifts for equilibration. Dial’s Algorithm B is fast since it is path-based and at the same time is efficient since it does not require paths to be stored. To date, Algorithm B is the fastest algorithm available for solving the static traffic assignment problem. Bar-Gera (1999) developed an origin based algorithm that is similar to Algorithm B. Bar-Gera (1999) differentiates his method as being origin based and hence distinct from the link based and route based methods. Dial (2006) reports that Algorithm B’s running times are better than Bar-Gera’s origin based algorithm.

The algorithm presented in this paper (B-Dynamic) is similar in principle to Dial’s Algorithm B. However, the current problem context is more complex since traffic dynamics and multiple decision dimensions are considered. Dial’s Algorithm B and the proposed B-Dynamic algorithm is discussed in detail later. Other dynamic traffic assignment algorithms are reviewed below.

2.2 Algorithms for Dynamic Traffic Assignment

Unlike static traffic assignment, the area of DTA does not have a single well defined problem statement. The primary goal of DTA is to improve on traffic dynamics when compared to static assignment (Peeta and Ziliaskopoulos 2001). Several alternative frameworks for DTA have been developed and can be broadly classified as analytical or simulation-based. It is important to note that simulation-based DTA approaches often have an overall analytical framework; however, when the constraints to describe traffic propagation in the analytical formulation are complicated and cannot be captured analytically, a traffic simulator is utilized to replicate the traffic dynamics (Peeta and Ziliaskopoulos 2001). Given this definition, the model proposed in the current paper can also be termed as a simulation-based DTA model. Therefore, the review here is restricted to simulation-based DTA models. In particular, the following simulation models which are commercially available and widely used are discussed - DYNASMART (Mahmassani 2001), DynaMIT (Ben-Akiva et al. 2001), and Dynameq (Florian et al. 2008). A more comprehensive
review of both analytical and simulation-based DTA models is available in Peeta and Ziliaskopoulos (2001).

DYNASMART (Mahmassani 2001) (DYnamic Network Assignment-Simulation Model for Advanced Road Telematics) is a simulation assignment model that combines a microscopic level of representation of individual travel decisions with a macroscopic description of traffic flow. A time-dependent origin-destination (O-D) matrix is assumed to be given and based on assignment rules the demand is assigned to routes. Several alternative assignment rules are modeled including system optimal (SO) objective, user equilibrium (UE) objective, boundedly rational path switching, and a current best path assignment when a traveler can consult instantaneous traffic information while at the origin. Traffic simulation is modeled at a macroscopic level but at the same time individual or groups of vehicles are tracked by the simulation. Link movements are modeled using a modified Greenshield speed-density relationship and node transfers are used to compute delays when transferring between links. The framework may be described as a path based framework with a multi-user class K-shortest path algorithm providing available paths for processing. The solution algorithm is iterative with all-or-nothing assignment employed for drivers with SO and UE objectives, and route switching rules for boundedly rational users. Given, the mixed behavior models included in the simulation there is no single overarching analytical formulation or equilibrium criteria that can be used to describe the framework. However, it should be possible to isolate the different behavioral models to develop equilibrium frameworks.

DynaMIT (Ben-Akiva et al. 2001) (Dynamic Network Assignment for the Management of Information to Travelers) uses demand and supply simulators in real-time dynamic traffic assignment to generate consistent user optimal guidance information. DynaMIT combines demand and supply simulation tools at different levels of aggregation. For example, individual responses to information needs to be captured at a disaggregate level while OD estimation is performed at an aggregate level. Traffic flow is modeled as a hybrid effect resulting from capacity restrictions on the roadways, deterministic queuing at bottlenecks, and macroscopic speed-density relationships. Traffic simulation is performed in two phases. First, in the update phase, macroscopic parameters such as speed and density are updated. Second, in the advance
phase, vehicles are advanced to new positions at a microscopic level. Since the behavioral framework adopted is at a disaggregate level based on discrete choice models, DynaMIT does not lend itself to equilibrium based DTA analysis.

Dynameq (Florian et al. 2008) is a simulation based dynamic traffic assignment model that incorporates user equilibrium behavior. Traffic flow is modeled as a discrete-event procedure that moves individual vehicles similar to microscopic models (car following, lane changing and gap acceptance effects are captured). Time-dependent OD matrices are assumed as input. Paths are initialized based on an incremental assignment technique and updated in subsequent iterations with a new set of dynamic shortest paths. After a pre-specified number of iterations, only used paths are retained and no new paths are updated. The algorithm employs the method of successive averages (MSA) to update path flows between iterations.

From the review above, it can be seen that there are several alternative models and paradigms for DTA. Unlike static traffic assignment, there is no consensus on a single unified method for formulating the problem. Further, even among the DTA formulations that solves for a dynamic extension of the UE (DUE) principle (Friesz et al. 1989), there are no guarantees or checks whether the solution obtained is an UE solution. Both DYNASMART and Dynameq perform UE assignment over only a sub-set of the total paths. This does not preclude the possibility of better paths remaining unused. While the need to restrict the number of paths may arise from the need to store all paths or due to scalability, in the current paper the model generates better paths considering the entire network. This is feasible since only used paths are stored. Further, the convergence criterion used in the current paper is a direct measure of the DUE condition. Earlier papers have utilized a gap measure to determine convergence. A gap measure may not necessarily result in an UE solution (Dial 2006).

### 2.3 Algorithms for DTA Models With Additional Choice Dimensions

The present study incorporate additional activity-related dimensions into the DTA framework. Therefore, the following studies that incorporate DTA with additional choice dimensions are reviewed in this

Abdelghany et al. (2001) develop a model where the unit of analysis is trip chains instead of trips. They develop an iterative simulation-assignment algorithm that is consistent with UE behavior. The simulation model embedded in the framework for traffic flow propagation is DYNASMART. An interesting variation employed in this paper is to assign each segment in a trip chain independently using shortest path for each segment. Abdelghany et al. (2001) argue that the alternative of allowing individuals to optimize over the entire trip chain could result in individuals choosing longer routes in one segment to save time on another segment. In contrast, in this paper, the entire trip chain including activity participation is modeled as an analogous route choice problem. Therefore, individuals optimize over their complete activity-travel sequence. Abdelghany et al. (2001) use MSA to update route flows and use a convergence criterion based on difference in number of vehicles assigned to a path over successive iterations. It must be noted that solution obtained with such a convergence measure may not necessarily satisfy equilibrium conditions.

Abdelghany and Mahmassani (2003) solve the DTA problem considering departure time, route choice, and sequence of activities simultaneously. The model is formulated as a stochastic dynamic user equilibrium problem and solved using an iterative method that employs MSA. The algorithm employs K-shortest travel time paths between any two stops in the trip chain. Subsequently a superset of all combinations of these shortest paths are enumerated. A stochastic networking loading method is employed which is different from the all or nothing assignment in Abdelghany et al. (2001). The convergence criteria is similar to Abdelghany et al. (2001) and hence suffers from the drawback mentioned above. Further since the algorithm requires path enumeration (albeit restricted to combination of K-shortest paths in each trip segment) it is anticipated the algorithm performs slower compared to Abdelghany et al. (2001). Consequently, the test problem solved in Abdelghany and Mahmassani (2003) (22 nodes, 68 links) is much smaller compared to the test problem in Abdelghany et al. (2001) (178 nodes, 441 links).
Lam and Huang (2003) develop combined activity/travel choice models considering activity location, route, and departure time dimensions. Ramadurai and Ukkusuri (2008) extend the analysis by considering all the above dimensions and duration of activity participation. Both papers employ a route swapping algorithm to solve the problem. The route swapping algorithm requires the enumeration of paths and is likely to be computationally infeasible even for medium-sized networks. Further, the route swapping method has known convergence issues (Nagurney and Zhang 1997, Lam and Huang 2003, Ramadurai and Ukkusuri 2008, H. R. Varia July 2004) and under the general convergence criteria may not converge to an equilibrium solution.

The algorithm proposed in this paper is analogous to Algorithm B developed by Dial (2006) but for the dynamic equilibrium problem in activity-travel networks (ATNs; Ramadurai and Ukkusuri (2008)) considering activity location, duration, departure time, and route choices. The algorithm does not explicitly enumerate paths and is hence efficient compared to the methodologies described above. Dial’s Algorithm B is described in detail in the next section.

3 Algorithm B (Dial 2006)

In Algorithm B, the network is decomposed into acyclic sub-networks rooted at the origin. This origin specific acyclic sub-network, referred to as a ‘bush’, contains arcs that carry all flow from the given origin to all destinations. The basic principle of Algorithm B is to ensure the min- and max-cost paths for each origin specific bush are within the \( \epsilon \) tolerance limit. This is achieved iteratively by equilibrating every origin rooted bush and updating it to include any new min-cost path and equilibrating again till convergence is achieved. Equilibration of a bush, in turn, is achieved by computing the min- and max-cost paths, shifting flows between these two paths\(^1\) so that they are equilibrated, and repeating the process till all paths are equilibrated.

\(^1\)The shifting of flows in Algorithm B is actually between non-overlapping path segments; however for clarity in presentation the details are not presented here. The reader is referred to Dial (2006) for details.
Dial (2006) identified four major advantages of Algorithm B compared to earlier static traffic assignment algorithms: Algorithm B (a) Provides improved precision, (b) Greater efficiency - primarily because it avoids storing or enumerating paths, (c) Employs a direct measure of solution quality to check for convergence, and (d) Provides substantial benefits while solving pivot-point traffic assignment problems. Pivot-point traffic assignment problems arise when solving the traffic assignment problem repeatedly for trip matrices that are only marginally different. Using the solution from the original matrix for a warm start, the algorithm is able to quickly re-compute the new equilibrium solution for the perturbed demand matrix. The proposed methodology - B-Dynamic - seeks to exploit these advantages of Algorithm B by extending it to the dynamic context.

Algorithm B has the following steps:

1. Initialization: For each origin create an initial feasible bush.

2. For each origin, perform the following operations on the corresponding origin specific bush:

   (a) Build max- and min-path trees

   (b) Equilibrate bush by shifting trips from max- to min-paths to make their costs equal.

   (c) Improve bush by including any cheaper path in the entire network that is not already in the bush.

   (d) Repeat above three steps if the bush has been improved in previous step; otherwise move to next origin.

3. Terminate when all origin specific bushes are equilibrated; otherwise return to Step 2.

It must be noted that in the static traffic assignment problem, the link cost functions are given by analytical expressions. The bush equilibration step described above can therefore be carried out precisely. Dial (2006) proves the convergence of the algorithm to the well known unique solution in the static traffic assignment problem. The simulation based DTA problem on the other hand does not have analytical expressions for determining link cost and proof of convergence cannot be obtained. Nevertheless
the overall structure of Algorithm B can be adopted to the equilibrium assignment problem in ATNs. However, the implementation details for each step are different.

4 Algorithm B-Dynamic

Algorithm B-Dynamic is presented in this section. The overall model framework is presented first. Individuals start at an origin, participate in zero or more activities en-route and eventually reach a destination. Individuals derive non-negative utility from participating in activities en-route, and negative utilities due to congestion delay and schedule delay (early or late arrival penalty). A linear specification is assumed for the utilities. The model framework presented here is very similar to Ramadurai and Ukkusuri (2008). However, a major difference in the current paper is the inclusion of schedule delay in the linear utility specification.

A positive integer value of demand for every demand pattern is assumed to be given. Demand is said to be satisfied if all activity-travel sequences represented by the routes chosen start at its origin, pass through all activities present in the demand pattern, and end at its destination. A feasible set of flows is one that satisfies the total demand for every demand pattern. The behavioral framework adopted is dynamic user equilibrium - all used activity-travel route sequences have the maximum utility while unused routes have equal or lesser utility. The problem is to determine a feasible set of flows that satisfy the equilibrium conditions. Additional details on the model specification and the input parameter details can be found in Ramadurai (2009).

The dynamic assignment problem considering multiple choice dimensions presents several challenges when compared to the static traffic assignment problem. First, the temporal evolution of flows in the different links has to be tracked. Second, there are no analytical equations to compute travel delays; travel delays are a result of complex traffic interactions. Third, additional choice dimensions such as departure time and duration of activity participation increase the complexity. Fourth, existence of equilibrium in dynamic traffic assignment considering additional choice dimensions is not always guaranteed. And
finally, the utility value of choosing an activity-travel sequence is dependent on the functional specification which could be non-smooth and discontinuous. These prevent an algorithm for solving dynamic traffic assignment from converging smoothly.

Before discussing the details of B-Dynamic, the reader should note an important argument. While it is possible to compute shortest path on dynamic networks utilizing a time-expanded graph and using static shortest path algorithms on the expanded graphs, dynamic traffic assignment on a time expanded graph employing Algorithm B will not work. This is because the topology of a bush changes within an iteration. For example, when shifting flow from a path, let the travel time on a path reduce. Now, the time expanded bush topology will change; two points in space connected using a longer temporal arc will now be connected with a shorter temporal arc. The equilibration step explained in Algorithm B is over a given fixed bush. Therefore, if the topology changes within an iteration, the equilibration step fails.

The alternative bush structure to be employed in B-Dynamic may be viewed as a temporally ordered bush. Algorithm B differentiated bushes based on the origin only. However, in ATNs each origin-destination pair is further characterized by activity participation decisions. The origin-destination-activity participation combination is referred to as a demand pattern. In B-Dynamic, each bush corresponds to a demand pattern. Further, for ease in implementation, the following additional assumptions are imposed:

1. Origin nodes have no incoming arcs.
2. Destination nodes have no out-going arcs.
3. Activity durations are fixed for every activity arc.
4. There are no capacity restrictions on activity arcs.

The reader may note that assumptions one and two are not restrictive; given a transportation network additional origins and destinations could be added with connecting arcs with zero travel time. The last two assumptions, however, are restrictive. They prevent the modeling of spatial restrictions and congestion
effects at activity participation locations. Relaxing these assumptions complicate the model significantly and are therefore required.

Finally an important note on the convergence criterion used here. By design, if the algorithm converges the resulting solution is an equilibrium solution. Under conditions when an equilibrium cannot be reached, the algorithm terminates with the current “best” solution that cannot be further improved by the algorithm. Given the lack of understanding on the properties of DUE it is not possible to precisely determine the reasons for a non-equilibrium solution being obtained. The results of the solution when the algorithm does not converge include the minimum and maximum utility values of currently used paths and the utility of the next best path in the entire network that can be no better than the best path that is currently used (which results in the “best” solution described above).

4.1 Algorithm Structure

1 Initialization: For each origin, destination, and activity sequence combination create an initial bush.

2 Equilibration: For each origin, destination, and activity sequence combination,

   2a Construct dynamic min- and max-utility paths from corresponding bush.

   2b If difference in utility between min- and max-utility paths is greater than $\epsilon$, shift flow from min- to max-utility path such that their utility difference is less than $\epsilon$. Else, skip to [2d].

   2c Re-compute link delay functions. Return to [2a].

   2d Check if the max-utility path on the entire network is greater than the max-utility path of the bush. If yes, augment the bush with new max-utility path. Return to [2a]. Else, continue to [3].

3 Termination: For each origin, destination, and activity sequence combination, check if the max-utility path on the entire network is lesser than or equal to the max-utility path of the bush. If yes, terminate. Else, return to [2].

Each of the steps above are discussed in detail below.
4.1.1 Initialization

The initialization step may be any feasible set of network flows. This could be a simple all-or-nothing assignment on the free-flow shortest path or could be a warm-start from a previously computed solution. A warm-start is expected to significantly reduce the computation effort similar to that observed with Algorithm B (Dial 2006). The current implementation of the algorithm utilizes an all-or-nothing assignment for each demand pattern with re-computation of link delay functions after each demand pattern has been assigned. This minimizes overlap in the paths chosen for the demand patterns that could have overlapping arcs.

4.1.2 Equilibration

The equilibration step is divided into several sub-procedures: a) construction of dynamic min- and max-utility paths, b) shifting flow from min-utility path to max utility path till the two are in equilibrium, c) re-computation of link delay functions, and d) update of bush with better paths from the entire network.

Equilibration is carried out for every activity-sequence combination from all origins to destinations. Dynamic min- and max-utility paths are constructed for each bush (corresponding to a unique origin, destination, activity-sequence combination). Constructing dynamic min- and max-utility paths in a bush requires a simple tree traversal and is achieved in linear time.

The equilibration of the max- and min-utility paths (say $P$ and $P'$) in each bush is an important step in the algorithm. This is achieved by solving the following equivalent minimization problem.

$$\min(f(x) - f'(c - x))^2$$

(1)

such that,

$$0 \leq x \leq c$$

(2)

where, $x$ is the flow on one of the paths, and $c$ is the total flow that is currently using the two paths.
\(f(x)\) is the utility derived by using path \(P\) and \(f'(c - x)\) is the utility derived from using path \(P'\). The problem is a minimization problem in one variable and can be solved using existing methods. However, the functions \(f\) and \(f'\) do not have analytical expressions. They are obtained from the traffic flow model explained below. The traffic flow model has to be repeatedly simulated by changing the value of \(x\). However, instead of simulating the entire network, only the links that constitute the paths are utilized for a localized simulation. This significantly reduces the computational time though possibly at the cost of accuracy. This localized simulation is likely to be very efficient and the overall path equilibration step converges within reasonable computation time. Further, for certain traffic flow models (such as the one employed below where downstream capacity does not affect link exit flow) the accuracy is not significantly affected by ignoring adjoining links.

### 4.1.3 Traffic Flow Model

Once the paths have been equilibrated the entire network is utilized to perform a traffic flow simulation.

In this study, a simple point queue model of traffic flow is employed. Each link has a free-flow traversal time and a maximum fixed rate of traffic discharge. The average link delay faced by vehicles departing at any given time instant can be obtained from the cumulative arrival-departure curve profiles. An example arrival-departure profile for a link is shown below in Figure 1. Since time is discretized, the curves are step-functions that have quantum jumps over fixed time intervals. The average delay of vehicles entering a link at time \(t\) is given by the area in the shaded region divided by the total number of vehicles entering the link at time instant \(t\).

An important point to note in the above implementation of computing the link-delay function is its independence from path attributes and reduced dependence on other downstream links in the network (since availability of capacity downstream is not modeled in a point queue). Since the link-delay function is essentially at a link level, they can be quickly recomputed for any changes to inflow into a link. For example, when shifting flow from the min- to max-utility path, step 2c in the algorithm requires recomputing the link-delay only on the affected links. The link independence assumption allows us to
ignore other links in the network that are not on the min- and max-utility paths thus increasing the efficiency of the algorithm. However, this requires additional storage of flow values. Flow from different demand patterns are stored separately to allow for partial modifications to the cumulative curves. More complicated traffic flow models that consider jam density (such as the Cell Transmission Model (Daganzo 1994, 1995)) can be modeled as part of this step. However the link independence assumption may no longer hold in these cases. Enforcing the link independence assumption in these more complex traffic flow models is an approximation that will have to be tested rigorously through numerical techniques.

4.1.4 Modified TDSP

A key step in the B-Dynamic algorithm is the computation of min- and max-utility paths. This is achieved using a modification to the time dependent shortest path (TDSP) algorithm (Ziliaskopoulos and Mahmassani 1993). The TDSP algorithm provides shortest path labels from an origin \(o\) to all nodes or to a destination \(d\) from all nodes. Since the network is time-dependent and every arc traversal takes a positive unit of time, the network is acyclic. The shortest path computations can be performed
efficiently in acyclic networks (Ahuja et al. 1993).

Given a network $G(N, A)$ where $A$ includes both travel and activity arcs.

Notation:

$\lambda_i[t, l]$: Shortest-path label for node $i$ at time-period $t$ and activity-combination lexicon $l$

$L$: Set of all activity-combination lexicon.

Let there be two different activities Shop and Eat out; the lexicon set is given by

$L = \text{None, Shop, Eat out, Shop+Eat out}$

$SE$: Scan eligible list is a 2-tuple consisting of (node $i$, time interval $t$) as opposed to just the node. This may significantly reduce the number of computations that have to be made since only arcs in future time intervals need to be modified. Recall that maintaining a scan eligible node list would require all time intervals (including redundant past time intervals) to be scanned.

$FS_i$: Forward star at node $i$.

$AA_i$: Set of activity arcs at node $i$. The id $k$ of the activity arc is stored.

$l_k$: Label of activity arc $k$, for example Shop, Eat out

$d_k$: duration of activity arc $k$.

$u_k(t)$: Utility of participating in activity arc $k$ when starting the activity at time $t$. This utility is assumed to depend on the arrival time (schedule delay penalty) and the duration of activity participation.
\[ u_k(t) = \rho_{l_k} d_{l_k} - \beta_{l_k} s - \gamma_{l_k} \bar{s}, \] where \( \rho_{l_k} \) is the utility derived per unit time from participating in activity \( l_k \), \( \beta_{l_k} \) is the unit measure of disutility if arriving early to an activity, \( \gamma_{l_k} \) is the unit measure of disutility if arriving late to an activity, and \( s \) and \( \bar{s} \) are the measures of deviation (early and late respectively) from the preferred arrival time at the activity.

\[ u_{ij}(t): \] \((dis)utility of traveling on arc \( i \rightarrow j \) leaving node \( i \) at time \( t \). \( u_{ij}(t) = -\alpha d_{ij}(t) \), where \( \alpha \) is the utility value of travel time.

\[ d_{ij}(t): \] duration of traversing arc \( i \rightarrow j \) leaving node \( i \) at time \( t \)

The primary difference between the modified TDSP and the traditional TDSP is: for every (node \( i \), time interval pair \( t \)) in the scan eligible list, both travel arcs \(((i, j) : j \in FS_i)\) as well as activity arcs \((k \in AA_i)\) are scanned. The algorithm is presented below.

**Step 1** Initialization

\[ \lambda_i(t, l) = \infty \quad \forall \quad (i, t, l) \in (N, T, L) \setminus (o, 0..T, None) \]

\[ \lambda_o(0..T, None) = 0 \]

Insert \((o, 0..T)\) into SE list.

**Step 2** If SE is empty, then go to step 3.

Else, remove top (node \( i \), time \( t \)) pair from SE list.

For each activity combination \( l \in L \)

For each arc \((i, j) \in FS_i\)

If \( \lambda_j[t + d_{ij}[t], l] > \lambda_i[t, l] + u'_{ij}[t] \)

Then, \( \lambda_j[t + d_{ij}[t], l] = \lambda_i[t, l] + u'_{ij}[t] \)

\[ PRED_j[t + d_{ij}[t], l] = [(i, j), t] \]
Insert \((j, t + d_{ij}[t])\) into SE list.

Else, go to next node \(j\)

End Loop

End Loop

For each activity combination lexicon \(l \in L\)

For each activity arc \(k \in AA_i : l_k \notin l\)

If \(\lambda_i[t + d_k, l + l_k] > \lambda_i[t, l] + u'_k[t]\)

Then, \(\lambda_i[t + d_k, l + l_k] = \lambda_i[t, l] + u'_k[t]\)

\(PRED_i[t + d_k, l + l_k] = [k, t]\)

Insert \((i, t + d_k)\) into SE list.

Else, go to next arc

End Loop

End Loop

**Step 3** Stop

In terms of computational complexity, the TDSP algorithm will need to be repeated once for every activity combination possible or at most \(|L|\) times. The number of combinations of activities is given by \(2^m\), where \(m\) is the number of activities. However, it is reasonable to assume that the number of activity-stops in a single tour are not likely to be a very large number. When number of activity stops is 5, the TDSP algorithm has to be repeated at most \(2^5 = 32\) times. Further, the actual computation time is likely to be lesser since several of the activity arcs become redundant if the activity has already been participated in. For example, if eat-out is an activity the individual participates in, then while computing the labels that include eat-out activity all subsequent eat-out activity arcs can be ignored from the computation.

Before presenting results from example networks an important algorithm implementation technique needs to be discussed. Since flow in the bush in B-Dynamic is differentiated based on the activity arcs it has already traversed, it is important to keep track of the list of arcs the flow has already visited.
This is achieved by storing a (one step) temporal ordering of arcs visited with unique, modified arc id values. Whenever a new path is added to the bush, it is checked whether there are any arcs in the bush that are starting at the corresponding departure time. If the departure time already exists then the first link on the path that is not already included in the bush is identified. This link is included to the bush and is labeled with the smallest available unique modified arc id. Every subsequent link in the path is sequentially added to the bush with corresponding increments to the modified arcs ids. A map that references the next modified arc id given the current one as well as an overall index that tracks the original arc ids to the modified arc ids are maintained. The two indices together allow the temporal ordering of arcs in the bush to be preserved. The case when no arcs in the bush start at a departure time, requires the construction of a new branch in the bush and then following the steps as above. Further, in addition to the temporal ordering that ensures the links in a path are visited in the right order, it is important to ensure the right amount of flow reaches the link in the right temporal order. Flows in arcs in the bush are determined by two separate indices referred to as the demand split and flow split. The demand split index stores the flow or demand that is assigned to each departure time. The flow split index stores the fraction of flow that needs to be assigned to different downstream arcs given the current arc. The flow split index are referenced by the modified arc ids - not the original arc ids. The implementation details are presented using a step-by-step example in Section 5.1.

5 Example Networks

The B-Dynamic algorithm has been coded using C++. All the numerical experiments were run on a Windows PC with a 32-bit processor. The code was compiled using Microsoft Visual C++ 2008 compiler. The code reads input from a generic input file. The input includes the number of time intervals, nodes, travel and activity arcs, and demand patterns, travel arc characteristics (start node, end node, queue discharge capacity, and free flow travel time), activity arc characteristics (node, activity type, and duration), and finally the demand pattern characteristics (origin, destination, demand, value
of travel time, activity participation type and corresponding schedule delay parameters).

To demonstrate and describe the algorithm in detail several numerical experiments were conducted. The first example is a small network to explain in detail the various steps in the algorithm. The second example is on the well known, medium-sized Sioux Falls network to demonstrate the efficiency of the algorithm in solving larger networks. The third set of examples demonstrate the scalability of the algorithm with respect to network size and congestion levels.

5.1 Example 1. Small Network to Illustrate Step by Step Progress

A simple 4 node network (Figure 2) is used to illustrate the step by step progress of the algorithm. There are 4 travel arcs and 4 activity arcs in the network. Travel arcs are represented with solid lines in the figure while activity arcs are shown using dotted lines. All four activity arcs are of the same type. The two arcs on the same node have durations of 1 and 2 time units. There are two demand patterns that start at node 1 and end at node 4; only one of the demand patterns requires participating in the activity en-route.

The step by step progress of the demand pattern that includes the activity participation (DP1) is traced here. In the initialization step, the shortest path given free flow is any of the paths that passes through the arcs with the longer duration. The algorithm picked the path given by arcs (0, 11, 2) with a departure time of 8. The evolution of the bush, demand split, flow split, and modified arc index matrices are shown in Figure 3. At the end of the initialization step, the bush for DP1 has three arcs with the modified arc id labels as shown in the figure. The bush is stored as a nested map structure. The first level key is the node id. The second key is the previous arc’s modified id, while the third key is the current arc’s modified id. In the case of the first arc from origin, the previous arc’s id is the departure time. The final value stored in the bush structure is the flow split - this represents the percentage of flow from previous arc that reaches the current arc. All demand departs at time interval 8. The utility of the path after assigning all flow was $-24.4$ units.

In the next iteration, the path given by arcs (1, 13, 3) with a departure time of 8 was chosen as the
shortest path. This is a symmetric path passing through node 3 to the path first chosen. The utility of this shortest path was $-4.4$ which is clearly better than utility of the current used path $(-24.4)$. The new path is added to the bush with the three arcs in the path assigned modified arc ids $(3, 4, 5)$ as shown in Figure 3. The next step in the algorithm after updating the bush is to equilibrate the two paths. In the beginning the first and only path had all the flow (176 units). After the path equilibration step, flow is distributed in the ratio 45.4% in path 1 and 54.5% in path 2. The corresponding utility values for both paths after equilibration is $-14.4$. Therefore, by taking flow out of path 1, its utility increased while path 2 which received this flow decreased in utility value. The flow values mentioned may be one of the several solutions that provide the same utility value. The corresponding bush structure and flow split values are shown in figure 2. At the end of this second iteration, the other demand pattern also had two equilibrated paths with utility values of $-24.4$. Another iteration of the shortest path algorithm failed to produce any more better paths for both demand patterns. Therefore the algorithm converged and terminated.
Figure 3: Bush Evolution
It must be mentioned that the particular input values were such that the algorithm was able to converge after the second iteration. However, this is not always the case. For example, when the demand values were modified to \((124, 126)\) instead of \((174, 176)\), the algorithm terminated only after 14 iterations and no equilibrium flows were found. The final utility values for the demand pattern without activities were: minimum utility path = \(-16.1\) and maximum utility path = \(-9.7\). The final utility values for demand pattern with activities were: minimum utility path = \(-22.7\) and maximum utility path = \(-8.6\).

While non-existence (or existence) of equilibrium solutions have not been shown theoretically, it can be explained from an algorithmic view point. The utility of a given path is computed based on average utility values experienced in each travel and activity arc (using the modified TDSP method). A small change in travel time in the first arc due to a small additional flow on a path could result in significant additional delays in subsequent links of the path. This cascading effect could result in the new utility value with small additional flow being significantly different from the previous utility value with original flow. Further, the link level utilities are discrete, step functions and the total utility is a sum of several step functions. Existence of equilibrium in static traffic assignment has generally required total travel times to be a monotonous and continuous function of flow; these assumptions generally do not hold in simulation-based DTA models. The issue of solution existence and the computational framework and definitions employed for computing utility values are important research questions that needs to be addressed in future.

5.2 Example 2. Sioux Falls Network

A second example network analyzed is the well known medium sized Sioux Falls network. The network is shown in Figure 4. The network has a total of 30 nodes and 82 travel arcs. Nodes 14 - 17 and node 23 have activity arcs that are not shown in the figure. Two different activity types are represented. Nodes 14 and 16 carry activity type 1 arcs (durations 2, 4, 6, and 8 time units). Nodes 15, 17, and 23 carry activity type 2 arcs (durations 5 and 10 time units). There are a total of 14 activity arcs. Nodes 25-27 are origins and nodes 28-30 are destinations. There are a total of 9 O-D pairs. Three of the O-D pairs do not have
any activity participation, 4 O-D pairs participate in one activity (2 in each type), and the remaining 2
O-D pairs participate in both activity types.

The algorithm converged after 45 iterations and took a reasonable time of 57 minutes. The average
running times of different steps of the algorithm are: traffic flow modeling step - 0.157 seconds, modified
time dependent shortest path computation in entire network - 1.62 seconds, and path equilibration step
- 2.69 seconds. The main results are presented in Table 1. The table lists for each demand pattern, the
final utility values for the min- and max-paths and the number of arcs in the final bush. In all but two
demand patterns the difference between the min- and max-utility paths is less than 3 units. In two cases,
the values match - these demand patterns are in perfect equilibrium. The number of arcs in each demand
pattern’s bush varies from 14 to 45. The origins and destinations themselves are not all equidistant. And
few of the demand patterns had additional activity arcs. Assuming an average of 7 to 9 arcs in a path,
the demand patterns have in the range of 2 to 6 different paths in them which suggests a good dispersion
of the flow through various parts of the network.

Figure 5 plots the evolution of utility values over several iterations for the minimum and maximum
utility of paths for each demand pattern. The graphs for demand patterns 4, and 6 and 9 were similar
to demand pattern 3 and 5 respectively and have not been included. As can be observed from the plots,
the convergence is not smooth. This is because the values plotted is for the current iteration’s minimum
and maximum utility value and not the value for a fixed path. The discrete nature of the problem and
the possibility that a delay by one time unit at the origin could reflect in delays of several time units at
destination owing to the discrete step function description of congestion delay are reasons for the non-
smoothness. Alternative methods for computation of delay in the shortest path algorithm may improve
the smoothness of convergence and could also reduce the number of iterations required. The graphs also
indicate intermediate iterations when the difference between the minimum and maximum utility values
were better than or close to the values in the final step. For example about iteration 30, almost all
Figure 4: Sioux Falls Network
Table 1: Results for Sioux Falls Network

<table>
<thead>
<tr>
<th>Pattern Id</th>
<th>Demand</th>
<th>Min. utility</th>
<th>Max. utility</th>
<th>Number of Arcs in Bush</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-24.4</td>
<td>-22.7</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>-20</td>
<td>-8.4</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>-10.4</td>
<td>-10.4</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>-3.7</td>
<td>-3.1</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>-33.2</td>
<td>-18</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>-1.2</td>
<td>2.9</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>-14</td>
<td>-14</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>-10.4</td>
<td>-9</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>-0.1</td>
<td>1.6</td>
<td></td>
<td>35</td>
</tr>
</tbody>
</table>

demand patterns appear to have converged. An alternative criterion for termination could be a more relaxed value for the $\epsilon$ tolerance level between the minimum and maximum utility paths.

Figure 6 plots the cumulative arrival-departure curves of few of the congested links. The links close to the origin and the links adjacent to the nodes containing activity arcs were more congested. In this implementation, a point queue model of traffic flow has been used. This explains the fixed rate of departure in all the cumulative departure curves. More sophisticated traffic flow models that reflect the capacity restrictions in downstream links similar to density-based models will provide more accurate and interesting insights. The cumulative curves can also be utilized to compute link delays and total systems delays. The total system travel time for the current problem was 48669.45 time units.
Figure 5: Convergence of Algorithm - Sioux Falls Network
Figure 6: Cumulative Arrival-Departure Curves for Select Congested Links
5.3 Example 3. Comparison of Run Times for Different Network Sizes and Congestion Levels

Several examples were run to study the scalability of the algorithm with respect to network size and congestion levels. Square grid networks were employed for these experiments. In all these examples, 3 origins and 3 destinations (9 O-D pairs) were used. Origins were connected to the lower left extreme node and the middle nodes on the lower and left edges of the square grid. Destinations were connected to the upper right extreme node and the middle nodes on the upper and right edges of the square grid. The nodes on the diagonal were defined as locations for participating in an activity; five different activity arcs with durations of 2, 4, 6, 8, and 10 time units were defined at each of these diagonal nodes. Nine demand patterns were defined - each corresponding to an O-D pair and all demand patterns required participation in the activity at one of the diagonal nodes. Figure 7 presents the layout of one of the grid networks used in these experiments.

The first set of experiments compared the run times of various steps of the algorithm with respect
to network size. The algorithm has three major steps: equilibrate paths, modified TDSP, and traffic flow model. The update bush step and the computation of TDSP within the bush are minor steps that take only fraction of the time compared to the above major steps. In all the experiments on run times, the algorithm is stopped after twenty iterations since the objective is to obtain average run times and not the total run time for the algorithm. The numbers presented are average run time for each of the step. The average is across different demand patterns and hence values for a particular demand pattern could be different from the average. Four different square grid networks were used: 5x5, 7x7, 11x11 and 17x17. The demand in each demand pattern was 200; therefore the total demand was 1800. Figure 8 plots average run time taken by the three major steps for different network sizes. Among the three steps, for small networks the most expensive step is path equilibration. But as the network size gets larger, the modified TDSP step’s run time increases at a much faster rate than the equilibrate path steps. For the 17x17 network, the modified TDSP step takes as much as four times longer than the equilibrate path step. The flow model step takes the least amount of time among the three steps and also does not scale poorly (average run time for 5x5 and 17x17 networks are 0.18s and 0.83s respectively).

The above results indicate that as network size increases, the modified TDSP step becomes increasingly burdensome. To improve the efficiency of the B-dynamic algorithm in large scale networks, it is important to improve the efficiency of the best utility path computation. Algorithms such as A-star or other faster heuristics for shortest path could be explored. Another promising methodology towards improving efficiency is the use of parallel processing. The TDSP algorithm has been successfully demonstrated to deliver significant efficiency improvement when implemented parallelly (Ziliaskopoulos and Kotzinos 2001). Within the context of Algorithm B-Dynamic the path equilibration step also allows parallel processing.

The second set of experiments on the scalability of the algorithm were done with varying demand levels. The 5x5 grid network was used for performing this set of experiments. The demand levels were varied from low to very high. The lowest demand level had 100 units of demand in each demand pattern (total 900 units). The next three levels had 200, 300, and 400 units for each demand pattern (total of
Figure 8: Comparison of Run Times for Different Network Sizes
1800, 2700, and 3600 units respectively). Since the smaller network was used here, the modified TDSP step took lesser time on an average compared to the path equilibration step.

Figure 9 plots the average run time for the three major steps for different demand levels. As expected there is very little variation in the time taken for computing the best utility paths across all demand levels. The modified TDSP step is flow independent and does not depend on congestion levels. On the other hand, both equilibrate paths and traffic flow model steps increased linearly with increasing demand. The traffic flow model step increased at almost the same rate as the demand levels. The average run times for the traffic flow model step were 0.09s, 0.18s, 0.29s, and 0.35s in increasing order of demand levels. For the path equilibration step, the average run times were 1.64s, 2.85s, 4.45s, and 5.89s in increasing order of demand levels. The path equilibration and traffic flow model steps of the algorithm scales linearly with respect to increase in demand levels while the modified TDSP step’s run time does not change with demand levels. However, in terms of overall run time of the algorithm to convergence the number of iterations required for convergence is expected to increase with increase in demand levels.

6 Summary

In this paper an efficient algorithm that obviates path enumeration was proposed for solving dynamic user equilibrium in Activity-Travel Networks. The algorithm is a novel extension of Algorithm B (Dial 2006) to dynamic networks and hence referred to as Algorithm B-Dynamic. The details of the various steps of the algorithm and numerical results from an implementation of the algorithm in C++ were presented. Overall, the algorithm performed well for up to medium-sized networks. Further, the algorithm scales fairly well with increasing demand levels. However, in its current form, the algorithm is not suitable for simulating large scale networks especially for real-time applications.

There are several improvements that could be implemented to improve the efficiency of the algorithm. More efficient algorithms for computing best utility paths in large scale networks could be explored. More promising is the possibility of implementing the algorithm using parallel processing. Parallel processing
<table>
<thead>
<tr>
<th></th>
<th>Very High</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equilibrate paths</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Flow model</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>TDSP</td>
<td>1.5</td>
<td>1.2</td>
<td>1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Figure 9: Comparison of Run Times for Different Demand Levels
could quicken two of the three major steps in the algorithm and could enable the implementation of B-Dynamic for real-time applications. Further, enabling warm starts for the initialization step could provide additional improvements in efficiency for real-time applications.

Other extensions include implementing more accurate traffic flow models. At present, the point queue based traffic flow model has been implemented. However, the point queue based model cannot model link spillovers as well as density effects. However, implementing more advanced models come at the expense of increase in computation time. Since the traffic flow model is the fastest step in the current implementation of the algorithm, the scalability of the algorithm and the accuracy of the path equilibration step needs to be re-examined when different traffic flow models are implemented.

**Acknowledgements** This work was partly supported by the US National Science Foundation Award No. 0826874: Incorporating Household Decision Making and Dynamic Transportation Modeling in Hurricane Evacuation: An Integrated Social Science - Engineering Approach. However, the authors are only responsible for the findings and conclusions in this paper.

**References**


