# Hierarchical Time-Dependent Shortest Path Algorithms for Vehicle Routing under ITS

Mark Mahyar Nejad, Lena Mashayekhy, Ratna Babu Chinnam, and Anthony Phillips [*]

**Abstract**

The development of efficient algorithms for vehicle routing on time-dependent networks is one of the major challenges in routing under Intelligent Transportation Systems. Existing vehicle routing navigation systems, whether built-in or portable, lack the ability to rely on online servers. Such systems must compute the route in a stand-alone mode with limited hardware processing/memory capacity given an origin/destination pair and departure time. In this paper, we propose a computationally efficient yet effective hierarchical algorithm to solve the time-dependent shortest path (TDSP) problem. Our proposed algorithm exploits community-based hierarchical representations of road networks, and it recursively reduces the search space in each level of the hierarchy by using our proposed search strategy algorithm. Our proposed algorithm is efficient in terms of finding shortest paths in milliseconds for large-scale road networks while eliminating the need to store preprocessed shortest paths, shortcuts, lower bounds, etc. We demonstrate the performance of the proposed algorithm using data from Detroit, New York, and San Francisco road networks.

*Keywords*: time-dependent shortest path, routing under ITS, hierarchical search strategies, community structure detection.

[*]M. M. Nejad is with the School of Industrial and Systems Engineering, University of Oklahoma.
L. Mashayekhy is with the Department of Computer and Information Sciences, University of Delaware.
R. B. Chinnam is with the Department of Industrial and Systems Engineering, Wayne State University.
A. Phillips is with the Research and Advanced Engineering, Ford Motor Company.
E-mail: mark.nejad@ou.edu, mlena@udel.edu, r_chinnam@wayne.edu, aphilli8@ford.com

## 1 Introduction

The quickly expanding Intelligent Transportation Systems (ITS) coverage around the world can be a key enabler for efficient vehicle route planning and for reducing the effects of traffic congestion on travel times. ITS provides valuable information for a time-dependent road network, such as time-varying travel times for traversing road segments at high resolution (Nejad et al., 2011). Routing algorithms must exploit these traffic information feeds efficiently, both to plan the route in advance and to update it en route. In general, an efficient routing algorithm should strike a balance among preprocessing time, query time, optimality gap, and storage/processor memory requirements. In addition, the scalability of the routing algorithm for handling large-scale road networks while maintaining reasonable response times is an important property. Depending on the form of implementation of the routing application, however, some of the aforementioned features may be prioritized over others. In this paper, we focus on large-scale deterministic time-dependent transportation networks. The need for fast responses to ITS information puts the speed-up techniques for shortest path problems (SPP) on time-dependent networks at the heart of computational needs for routing. In addition, a vast majority of vehicle routing navigation systems, whether built-in or portable, lack the ability to rely on online servers and must compute the route in a stand-alone mode with limited hardware processing/memory capacity. This last aspect is the primary focus of this paper to design computationally efficient yet effective hierarchical search strategies and algorithms to solve the time-dependent shortest path problem (TDSP).

**Definition 1 (Time-Dependent Shortest Path (TDSP))** *Given a time-dependent network, an origin $O$, a destination $D$, and a start time, the* time-dependent shortest path *is a path with the minimum travel time among all paths from $O$ to $D$ starting at the specified starting time.*

The TDSP problem is an adaptation of SPP to time-dependent networks. Cooke and Halsey (1966) first studied the TDSP problem using dynamic programming. Dreyfus (1969) studied the generalization of Dijkstra's algorithm for determining TDSP with the same time complexity as the SPP problem. Ahn and Shin (1991) and Kaufman and Smith (1993) proved that the TDSP problem is polynomially solvable. See Foschini et al. (2011) for a recent study on the complexity of the TDSP problem.

Dynamic programming methods are prevalent in the literature for the TDSP problem. Such methods suffer from the curse of dimensionality in dealing with the scale and complexity of transportation networks. They require overly long query times for computing the route and for offering rerouting options once the vehicle is en route. On the other hand, naïve algorithms that arbitrarily limit the degree of ITS "look ahead" to a small neighborhood ahead of the vehicle to reduce the state space can lead to a higher optimality gap.

An approach to speeding up the computation of shortest paths is pre-computing the optimal paths, short-cuts, or lower bounds for all $OD$ pairs or a subset at different time windows (Bierlaire and Crittin, 2004; Song and Wang, 2011). Methods based on ALT (A*, Landmarks, Triangle inequality) employ landmarks to find lower bounds in order to direct the search in a reduced search space (Goldberg and Harrelson, 2005; Goldberg et al., 2007, 2009). Bidirectional ALT further reduces the search space by adding a backward search from the destination to reduce the search space that has to be explored by the forward search (Nannicini et al., 2008; Goldberg et al., 2009). In ALT-based methods, there is a tradeoff between choosing well-positioned landmarks and preprocessing time. These methods, however, require large memory space, rendering them ineffective for large road networks as well as for vehicles not relying on online routing services.

There are extensive studies on designing routing algorithms for stochastic networks, which each road segment has stochastic traversal times. There are two versions of the shortest path problem on stochastic networks, the expected shortest path problem (Gao and Chabini, 2006), where all information on the arc weights is available before starting the trip; and the shortest path with recourse problem (SPR) (Provan, 2003; Waller and Ziliaskopoulos, 2002), where only local traffic information is available. SPR is more realistic in routing applications since in reality all information on traffic network dynamics is not available. While it is desirable to consider the stochastic nature of the traffic networks, solving stochastic routing problems is generally complex and prohibitive for real-time routing on large-scale road networks. Hence, we focus on large-scale deterministic and time-dependent transportation networks.

In this paper, we propose an algorithm capable of solving TDSP in milliseconds on large-scale dynamic road networks without the need for storing memory-intensive precomputed paths, short-cuts, or bounds. In particular, we propose new search strategies that exploit the hierarchical structure of efficient road network representations.

Hierarchical approaches have been used in routing algorithms for large road networks, and have proven to be effective on both static networks (Fernández-Madrigal and González, 2002; Jagadeesh et al., 2002; Jung and Pramanik, 2002; Fu et al., 2006; Bauer and Delling, 2009; Rajagopalan et al., 2008; Hilger et al., 2009; Bauer et al., 2010; Song and Wang, 2011) and dynamic networks (Chou et al., 1998; Schultes, 2008; Buriol et al., 2008; Geisberger et al., 2012; Delling and Nannicini, 2012). A hierarchical search can dramatically reduce the search space. This is due to the fact that the search will take place predominantly at higher levels of network representations that tend to be sparse, with far fewer nodes and arcs. These methods mostly employ hierarchical representations based on the fixed topology and functional classification of road networks. Functional classification categorizes streets and highways into classes based on the character of service they are intended to provide. The classification is rooted in the road network design and helps determine the speed category and travel time of passing through the road under free-flow conditions. One issue inherited with a majority of hierarchical routing algorithms in the literature is enforcing the vehicle to travel over higher-level arcs (e.g., highways) without considering the traffic state of those arcs. Although the speed limit is higher at higher levels, and the optimal route might pass through higher levels under free-flow conditions, this route may not necessarily be optimal under different traffic conditions. Therefore, incorporating just the fixed topology of road networks and its functional classes may not be adequate for efficient hierarchical routing.

Instead of a functional class representation, we employ an emerging concept in analyzing complex networks called "community structure detection" (Clauset et al., 2008; Newman, 2011) to form hierarchical community-based representations of road networks efficiently (Newman, 2004; Blondel et al., 2008). We present a model of the hierarchical representation to aid the computational performance of our proposed algorithm for TDSP. While it has been shown that the community detection methods are effective for path-finding in static networks (Song and Wang, 2011), there are no studies for time-dependent networks. Our proposed algorithm for solving TDSP employs new hierarchical search strategies to reduce the state space without compromising optimality gap.

## 1.1 Our Contribution

We propose a hierarchical time-dependent shortest path algorithm (HTNGD) to solve the deterministic TDSP problem on large-scale networks. HTNGD uses community-based hierarchical representations of road networks, and it recursively reduces the search space in each level of the hierarchy by using our proposed search strategy algorithm, TNGD. We perform extensive experiments in order to investigate the performance of HTNGD. We use time-dependent A* (TA*) as a benchmark when we investigate the performance of HTNGD, and we compare HTNGD with the most successful speedup techniques in the literature. The results show that the overhead memory requirement and the pre-processing time of HTNGD are the lowest, and its query time is in terms milliseconds. These properties make HTNGD suitable for deployment in vehicle routing navigation systems that do not rely on online servers.

### 1.2 Organization

The rest of the paper is organized as follows. We explain hierarchical community-based representation of road networks in section 2. Section 3 describes the proposed algorithms for solving TDSP. Section 4 presents experimental results from applying the proposed algorithm on Detroit, New York, and San Francisco road networks. Finally, section 5 offers some concluding remarks and directions for future research.

## 2 Hierarchical Representation of Road Networks

Complex networks have attracted a great deal of attention across many fields of science (Guimera and Amaral, 2005; Palla et al., 2005, 2007). A recently proposed concept in analyzing complex networks is their "community structure" (Newman and Girvan, 2004; Clauset et al., 2008). Many networks can be decomposed into communities such that the densely connected subsets of nodes form communities with only sparser connections between them. A wide variety of methods have been lately developed for detecting communities in networks (see Fortunato (2010) for a recent review).

Road networks are commonly represented by directed graphs where streets form the arcs, and intersections are considered as nodes. To capture the dynamics of road networks, arc traverse times can be considered as arc "weight." Community detection methods can be employed to decompose the weighted road network to effectively represent the network structure and its connectivity (Nejad et al., 2012). Hierarchical search strategies can exploit this community structure for solving the

TDSP problem.

There are two approaches to build hierarchical representations of networks in the literature (Fortunato, 2010): agglomerative and divisive. In agglomerative, a bottom-up approach, the detected communities in a network become an input to another iteration of community detection method (Pons and Latapy, 2005). In divisive, a top-down approach, all nodes are considered as one community, then it splits into communities in lower levels of the hierarchy (Radicchi et al., 2004). In both approaches, each hierarchy forms a directed graph itself with fewer arcs and nodes as we go up the levels. These higher levels are abstractions of their lower-level graphs.

To model each level of the hierarchy, we consider the graph in level $h$ as $G^h(V^h, A^h, W^h)$ where $V^h$ is a set of nodes, $A^h$ is a set of arcs, and $W^h$ is a set of arc weights. Suppose that $G^h$ is partitioned into $k^h$ communities $C_i^h(V_i^h, A_i^h, W_i^h)$, where $i = 1, \ldots, k^h$ with the following properties:

$$\begin{cases} \bigcup_{i=1}^{k^h} V_i^h = V^h, \\ \bigcup_{i=1}^{k^h} A_i^h \subseteq A^h \end{cases} \tag{1}$$

where $\forall p, q, V_p^h \cap V_q^h = \emptyset$ , $A_p^h \cap A_q^h = \emptyset, 1 \leq p, q \leq k^h$, and $p \neq q$. In the rest of the paper, we refer to community $C_i^h(V_i^h, A_i^h, W_i^h)$ as $C_i^h$.

In each community $C_i^h$, a subset of $A^h$, $A_i^h$, connects its nodes, $V_i^h$ such that $A_i^h$ represents intra-community arcs. In addition to these arcs, $A^h \ / \bigcup_{i=1}^{k^h} A_i^h$ is a subset of arcs representing the intercommunity arcs, which connect pairs of communities in level $h$. For each arc in $A^h \ / \bigcup_{i=1}^{k^h} A_i^h$ that connects two communities $C_p^h$ and $C_q^h$, we define $w_{C_p^h C_q^h}^h$ as the travel time between centers of those communities. We set a virtual vertex as the center of a community. In the case of road networks, the coordinates of the center is the average of coordinates of all vertices within the projection of that community to the lowest level. Therefore, we set travel time $w_{C_p^h C_q^h}^h$ as the distance between the virtual vertices divided by the maximum speed limit. Note that the projection of each community $C_i^h$ to the lowest level covers a subset of nodes in $G^1$.

Each community in level $h-1$ is represented by a node in level $h$. That means each community $C_p^{h-1}$, $1 \leq p \leq k^{h-1}$, is represented by a node $v \in V^h$. If $v$ is a vertex ($v \in V_q^h$) that belongs to $C_q^h$, $1 \leq q \leq k^h$, then $C_q^h$ is a super-community of $C_p^{h-1}$ and $C_p^{h-1}$ is a sub-community of $C_q^h$. In each level, a node represents a sub-community. In general, $\bigcup_{i=1}^{k^{h-1}} C_i^{h-1} = V^h$. Thus, there is a

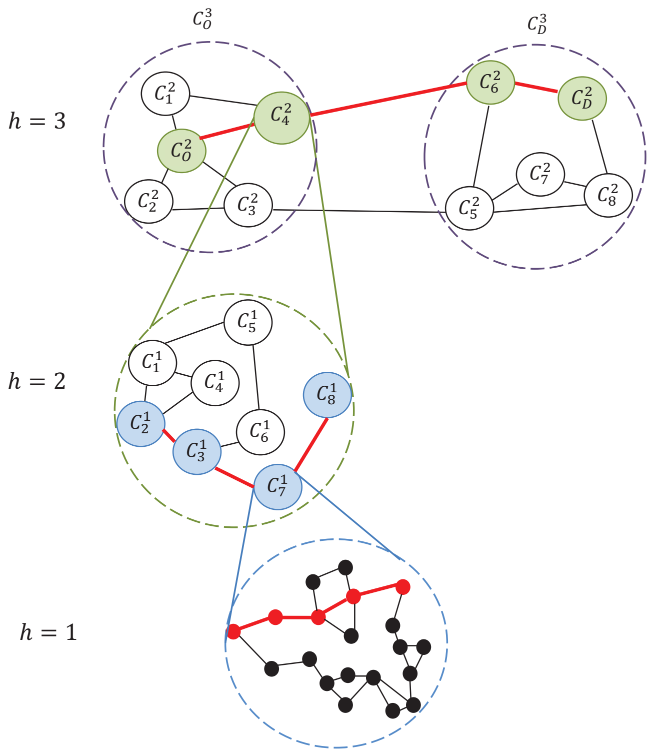

Figure 1: Illustrative example for the hierarchical representation of a network

one-to-one correspondence between $V^{h+1}$ and $C^h$, where $C^h = \bigcup_{i=1}^{k^h} C_i^h$.

In all levels of the hierarchy, $V^h$ is the set of communities of level $h-1$, where $h \neq 1$. If $h = 1$, $G^1$ represents the actual road network, where $V^1$, $A^1$ and $W^1$ represent sets of road intersections as communities, road segments, and road segment travel times, respectively. In our proposed time-dependent model of the road network, we denote $w_{ij}^t$ as the travel time of the arc $(i, j) \in A^1$ connecting $i \in V^1$ to $j \in V^1$, where $t$ is the arrival time at node $i$.

Fig. 1 shows a highly stylized example to illustrate the hierarchical representation of an undirected and an unweighted network with three levels of hierarchy. The graph in level 3 (i.e., $G^3$) consists of 10 nodes that are partitioned into two communities $C_O^3$ and $C_D^3$. Each node in this level is a community in level 2. For example, $C_4^2$ is represented as a node in level $h = 3$ which along with four other nodes forms community $C_O^3$ in level 3. Therefore, community $C_O^3$ is its super-community. In addition, $(C_O^2, C_4^2)$ is an intra-community arc within community $C_O^3$, and $(C_4^2, C_6^2)$ is intercommunity arc that connects two communities $C_O^3$ and $C_D^3$. Community $C_4^2$ in level 2 consists of 8 nodes (sub-communities), $C_1^1, \ldots, C_8^1$. For example, $C_1^1$ is a sub-community of $C_4^2$. We show the projection of community $C_7^1$ in level 1, which is a part of the actual graph $G^1$.

A modularity measure was first introduced by Newman and Girvan (2004) to measure the strength of partition of a network into communities. This measure gives a value, $\psi$, between -1 and 1 for a partition based on the density of arcs inside communities in comparison with the density of arcs between communities. A higher value of $\psi$ indicates a better partitioning of the network. $\psi$ is a property of a network and a specific partition of the network into communities. For simplicity, we assume nodes $i$ and $j$ belong to communities $C_i$ and $C_j$, respectively. In the case of weighted directed networks, the modularity measure for all arc $(i,j)$ and a given partition is defined as follows:

$$\psi = \frac{1}{m} \sum_{(i,j)} \left[ b_{ij} - \frac{d_i^{in} d_j^{out}}{m} \right] \delta(C_i, C_j) \tag{2}$$

$$\delta(C_i, C_j) = \begin{cases} 1 & \text{if } C_i = C_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$m = \sum_{(i,j)} b_{ij} \tag{4}$$

where $b_{ij}$ represents the closeness weight of the arc between $i$ and $j$, and $d_i^{in}$ ($d_j^{out}$) is the sum of the incoming (outgoing) arc closeness weights attached to vertex $i$ ($j$). It is worth mentioning that $b_{ij}$ indicates closeness or similarity between nodes $i$ and $j$ that can give useful information about communities. Not all weights on network arcs are necessarily appropriate for determining community structure. In traffic networks, the inverse of travel time between nodes $i$ and $j$ can be used as the value for $b_{ij}$ in order to find densely connected subsets of nodes as communities. For example, if the travel time between two nodes is long, it does not mean that these nodes are similar so they may be assigned to different communities.

We employ the Louvain method (Blondel et al., 2008), which is an agglomerative approach for constructing hierarchical representation of the network. This method not only extracts a hierarchical community structure, but exhibits excellent computational performance even for large-scale directed networks. The Louvain method is a heuristic method based on the gain in modularity,

$\Delta\psi_i$, by adding (removing) a vertex $i$ into (from) a community $C$ in each iteration of their proposed method. The gain in modularity, $\Delta\psi_i$, for directed and weighted networks is defined as follows:

$$\Delta\psi_i = \left[\frac{\sum_{j,\ k\in C} b_{jk} + \sum_{j\in C} b_{ij} + \sum_{j\in C} b_{ji}}{m} - \left(\frac{\sum_{j\in C,\ k\ \notin C} b_{jk} + d_i^{out}}{m}\right)\left(\frac{\sum_{j\in C,\ k\ \notin C} b_{kj} + d_i^{in}}{m}\right)\right]$$

$$- \left[\frac{\sum_{j,\ k\in C} b_{jk}}{m} - \frac{(\sum_{j\in C,\ k\ \notin C} b_{jk})(\sum_{j\in C,\ k\ \notin C} b_{kj})}{m^2} - \frac{d_i^{in} d_i^{out}}{m^2}\right] \tag{5}$$

where $\sum_{j,\ k\in C} b_{jk}$ is the sum of the weights of intra-community arcs of $C$, $\sum_{j\in C,\ k\ \notin C} b_{jk}$ is the sum of the weights of the arcs incident to vertices in $C$, and $\sum_{j\in C} b_{ij}$ is the sum of the weights of the arcs from $i$ to vertices in $C$. Each vertex $i$ is added to one of its neighboring communities that has the highest modularity gain.

Our proposed algorithm is not limited to any specific community structure detection methods; other community structure detection or graph partitioning methods can be applied. In the next section, we propose our hierarchical search method using the proposed hierarchical graph model.

## 3 Hierarchical Time-Dependent Shortest Paths

We propose a new hierarchical search algorithm for solving the TDSP problem on dynamic road networks with discrete and deterministic time-varying travel time. The algorithm exploits the hierarchical representation of the road network, as outlined in section 2.

We first introduce a Time-dependent Neighborhood Goal Directed (TNGD) search algorithm. The task of TNGD is to determine a spectrum of promising communities for exploration in each level of the hierarchy. We then propose a Hierarchical Time-dependent Neighborhood Goal Directed (HTNGD) algorithm that recursively employs TNGD to solve the TDSP problem. HTNGD efficiently searches over the entire hierarchical representation of the road network.

### 3.1 Time-dependent Neighborhood Goal Directed (TNGD) Search Algorithm

We consider a graph $G^h(V^h,\ A^h, W^h)$ as described in section 2 to find a spectrum of communities between $C_O^h$ and $C_D^h$ in level $h$, where $C_O^h$ and $C_D^h$ are the communities containing $O$ and $D$,

Table 1: Notation

| | |
|---|---|
| $t_O$ | Trip start time |
| $C_O^h$ | Origin community in level $h$ |
| $C_D^h$ | Destination community in level $h$ |
| $\alpha$ | Spectrum control parameter |
| $f_v$ | Estimated minimum total travel time among all paths passing through community $v$ from $C_O^h$ to $C_D^h$ |
| $g_v$ | Minimum arrival time from $C_O^h$ to community $v$ starting at time $t_O$ |
| $e(v, C_D^h, g_v)$ | Lower bound estimate on travel time to go from $v$ to $C_D^h$ assuming the arrival time to $v$ is $g_v$ |
| $S$ | Set of visited communities |
| $N$ | Set of nominated communities for the selection of the next community |
| $CS_{OD}^h$ | Core set in level $h$ |
| $Q^h$ | Spectrum of communities in level $h$ |

respectively. We define a spectrum $Q^h$ as follows.

**Definition 2 (Spectrum)** *A spectrum $Q^h$ is a set of communities in level $h$ such that the projection of that spectrum to the lowest level of the hierarchy structure contains at least one path from $O$ to $D$.*

In this subsection, we describe how TNGD finds $Q^h$. The likelihood of obtaining the shortest path in the spectrum can be increased by increasing the size of the spectrum.

TNGD algorithm is designed in a way that it returns a spectrum of communities connected through intercommunity arcs. It finds a set of connected communities, the core set $CS_{OD}^h$, connecting $C_O^h$ and $C_D^h$ with the shortest path through the community centers with the condition that there is at least one intercommunity arc for every consecutive pair of communities on the path. Note that this shortest path is at a particular level $h$, and the communities along this path identify the candidate communities for exploration at the lower level. Communities in the core set $CS_{OD}^h$ build a spectrum $Q^h$.

To increase the likelihood of finding the shortest path on the actual road network represented by $G^1$, TNGD can extend the initial spectrum $Q^h$ by adding neighbor communities of the core set $CS_{OD}^h$. However, this comes at a cost of increasing run time. Hence, TNGD employs a parameter $\alpha$ to strike a good balance between efficiency (search cost) and effectiveness (path optimality).

If $\alpha = 1$, TNGD includes all additional communities with a direct intercommunity arc to the core set, leading to a spectrum of communities $Q^h_{\alpha=1}$. If $\alpha = 2$, TNGD extends the spectrum $Q^h_{\alpha=1}$ by including once again all additional communities with a direct intercommunity arc to the current spectrum. This recursive procedure can be applied for any particular integer $\alpha \geq 1$. If $\alpha = 0$, TNGD returns just the core set. At the lowest level of the hierarchical representation, there is no need to build a spectrum; hence, $\alpha$ is set to zero.

We define a set of notations assuming a time-dependent network in Table 1. The proposed TNGD algorithm is given in Algorithm 1. The description of the TNGD algorithm is as follows:

TNGD starts with $G^h, C^h_O,\ C^h_D,\ h$, and $\alpha$ as input parameters. The objective of TNGD is to find a spectrum of communities in the level $h$ using the parameter $\alpha$. The algorithm uses $S$ and $N$ to store a set of visited communities and a set of communities to visit in the next iteration, respectively. TNGD initializes $S = \emptyset$, $N = C^h_O$, $f_{C^h_O} = \infty$, and the core set $CS^h_{OD} = \{C^h_O,\ C^h_D\}$ (line 1). It also initializes $g_{C^h_O}$ to departure time $t_O$, and $g_u$ to infinity for all communities $u$ in level $h$ except for $C^h_O$ (line 2). TNGD updates $g_u$ to minimum arrival time from $C^h_O$ to community $u$ (lines 3-22). Note that if $t_O = 0$, $g_u$ is minimum travel time. TNGD selects a community $v$ from $N$ with minimum total travel time (line 4). Estimated minimum total travel time $f_v$ is the sum of the minimum travel time from $C^h_O$ to community $v$ and the heuristic estimate of lower bound on travel time to go from an intermediate community $v$ to the destination community $C^h_D$, assuming the arrival time to $v$ is $g_v$ (i.e., $f_v \geq g_v + e(v, C^h_D, g_v)$). Then, it removes $v$ from the nominated set $N$ and adds it to the visited set $S$ (lines 6-7). TNGD updates $N$, $g_u$, and $f_u$ for each neighbor community $u$ of community $v$ (i.e., with a direct intercommunity arc) where either $u$ is not in the nominated set or there is a shorter path using $v$ to reach to $u$ (lines 12-16). If the travel time from the origin community to reach the neighbor community $u$ passing through $v$, $g_v + w^{g_v}_{vu}$, is smaller than the current travel time of the neighbor $g_u$, TNGD updates the travel time to the smaller time (lines 12-14). Note that $w^{g_v}_{vu}$ is the time-dependent travel time of the arc $(v, u)$, where the arrival time to $v$ is $g_v$. Then, TNGD updates the nominated set $N$ (line 13) by adding community $u$ to $N$. TNGD computes $e(u, C^h_D, g_u)$, which is a lower-bound estimate on travel time to go from $u$ to $C^h_D$, assuming the arrival time to $u$ is $g_u$, and then updates $f_u$ (lines 14-15). TNGD can use any lower-bound function to calculate the value of $e(u, C^h_D, g_u)$, for example, travel time from $u$ to $C^h_D$ under free-flow condition can be used as a lower bound (e.g., a vehicle cannot travel from $u$ to $C^h_D$

**Algorithm 1** Time-dependent Neighborhood Goal Directed (TNGD) Algorithm ($\mathbf{TNGD(G^h, C^h_O, C^h_D, h, \alpha)}$)

```
1: $v \leftarrow C^h_O,\ S = \emptyset,\ N = \{v\},\ f_v = \infty,\ CS^h_{OD} = \{C^h_O,\ C^h_D\}$
2: $g_v = t_O,\ g_u = \ \infty,\ \forall u \in V^h,\ u \neq v$
3: while $N \neq \emptyset$ do
4:   $v \leftarrow \arg\min_{n \in N} f_n$
5:   if $v \neq C^h_D$ then
6:     $N \leftarrow N \backslash \{v\}$
7:     $S \leftarrow S \cup \{v\}$
8:     for all $u$ where $(v, u) \in A^h$ do
9:       if $u \in S$ then
10:        Continue;
11:      else
12:        if $u \notin\ N$ or $g_v + w^{g_v}_{vu} < g_u$ then
13:          $N \leftarrow N \cup \{u\}$
14:          $g_u \leftarrow g_v + w^{g_v}_{vu}$
15:          $f_u \leftarrow g_u + e(u, C^h_D, g_u)$
16:        end if
17:      end if
18:    end for
19:  else
20:    Break;
21:  end if
22: end while
23: Construct $CS^h_{OD}$
24: {Build a spectrum}
25: $Q^h \leftarrow CS^h_{OD}$
26: $y \leftarrow CS^h_{OD}$
27: while $\alpha > 0$ do
28:   for all $v \in y$ do
29:     for all $u$ where $(v, u) \in A^h$ do
30:       if $u\ \notin Q^h$ then
31:         $Q^h \leftarrow Q^h \cup u$
32:       end if
33:     end for
34:   end for
35:   $y \leftarrow Q^h \backslash y$
36:   $\alpha \leftarrow \alpha - 1$
37: end while
38: Output: $CS^h_{OD},\ Q^h$
```

faster than when it is under free flow condition). While it is desirable to use a tight lower bound such as minimum travel time, calculating such a tight lower bound increases the execution time of the algorithm. If such bounds are calculated offline, the algorithm requires large memory space to save such lower bounds, which is not in alignment with our goal to decrease the need to store preprocessed shortest paths, shortcuts, lower bounds, etc.

TNGD stores communities forming the minimum total travel time path from $C_O^h$ to $C_D^h$ as a core set $CS_{OD}^h$ (line 23). The core set $CS_{OD}^h$ only contains communities in the level $h$. TNGD initializes the spectrum $Q^h$ by the obtained core set (line 25). To avoid removing some promising communities, the algorithm extends the search space by adding neighbor communities to the selected communities in the core set (lines 27-37), yielding spectrum $Q^h$ of the core set $CS_{OD}^h$. In doing so, TNGD uses a temporary set $y$ initialized with the core set (line 26). For each community in $y$, TNGD adds to the spectrum its neighbor communities which do not belong to the spectrum (lines 28-34). Then, TNGD updates $y$ to the set of newly added communities to the spectrum (line 35) and decrements $\alpha$ (line 36). Using $y$ decreases the amount of computation to build the spectrum since TNGD does not need to consider communities that are already belong to the spectrum. The output parameters of TNGD are the core set and the spectrum.

TNGD always finds the shortest path in each level as long as the estimated travel time obtained by the heuristic function is a lower bound of the actual travel time. The goal of proposing TNGD is to reduce the search space in each level of hierarchy by eliminating communities that would not be traversed by the optimal path. In the case of the lowest level where $h = 1$, $\alpha$ is always set to 0. As a result, TNGD in the lowest level becomes a time-dependent goal directed algorithm exploring only a subset of nodes selected by the projection of higher spectrums instead of the whole actual network.

## 3.2 Hierarchical Time-dependent Neighborhood Goal Directed (HTNGD) Algorithm

We now propose the Hierarchical Time-dependent Neighborhood Goal Directed (HTNGD) algorithm that incorporates a new hierarchical search strategy. HTNGD recursively employs TNGD, starting with the highest level of the hierarchy in which $O$ and $D$ fall into two distinct communities. The spectrum of communities resulting from TNGD is recursively projected to the level below, identifying the collection of communities to be searched at the level below. The process terminates at the lowest level, with TNGD identifying the shortest path.

The proposed HTNGD algorithm is given in Algorithm 2. The full details of HTNGD are outlined below. The algorithm receives an $OD$ pair and $\alpha$ as input parameters. It finds the

**Algorithm 2** Hierarchical Time-dependent Neighborhood Goal Directed Algorithm (**HTNGD**$(\mathbf{O}, \mathbf{D}, \alpha)$)

```
1:  for all levels $h$ from top to bottom do
2:     Find community $C_O^h$ containing $O$ in $G^h$
3:     Find community $C_D^h$ containing $D$ in $G^h$
4:     if $C_O^h = C_D^h$ then
5:        {in the same community}
6:        Continue; {go to the lower level}
7:     else
8:        {in different communities}
9:        if $h = 1$ then
10:          $(CS_{OD}^1,\ Q^1) = TNGD(G^1, C_O^1,\ C_D^1,\ 1,\ 0)$
11:       else
12:          $(CS_{OD}^h,\ Q^h) = TNGD(G^h, C_O^h,\ C_D^h,\ h,\ \alpha)$
13:          {changes in the lower level graph}
14:          for all communities $C_i^h \in Q^h$ do
15:             for all sub-communities $C_j^{h-1}$ of $C_i^h$ do
16:                for all $C_p^{h-1}$ where $C_j^{h-1}C_p^{h-1} \in A^{h-1}$ do
17:                   $C_q^h \leftarrow$ super-community $C_p^{h-1}$ in $G^h$
18:                   if $C_q^h \notin Q^h$ then
19:                      Update $w_{C_j^{h-1}C_p^{h-1}}$ to $\infty$ in $G^{h-1}$
20:                   end if
21:                end for
22:             end for
23:          end for
24:       end if
25:    end if
26: end for
27: Output: Shortest path $CS_{OD}^1$
```

communities $C_O^h$ and $C_D^h$ in the highest level of hierarchy (lines 2-3). If $O$ and $D$ are located within the same community at this level, the algorithm proceeds to the next lower level for the route search (lines 4-6). This procedure continues until $O$ and $D$ fall into different communities. Then, the algorithm executes the TNGD on $G^h$ to find the spectrum $Q^h$ (lines 8- 12).

To eliminate communities that do not belong to the current spectrum from the search space, we set the weights of the intercommunity arcs going out of the spectrum $Q^h$ to infinity (lines 13-23). To do so, for each community in the current spectrum, the algorithm first finds communities that fall into the projection of that community at the lower-level denoted sub-communities. Then, for the selected sub-communities, it finds their neighbor communities with direct intercommunity arc (line 16). If the communities of these neighbors at the level above (denoted super-communities) are not in the spectrum, the algorithm sets the weight of their intercommunity arcs to infinity (line 19).

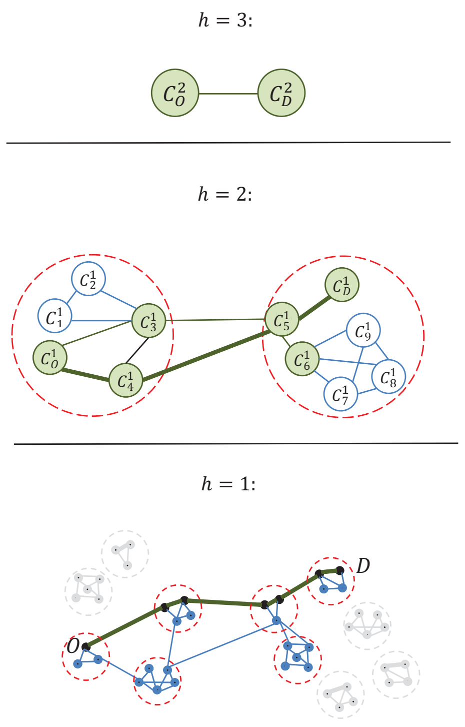

Figure 2: Illustrative example for HTNGD

These changes are tracked in $G^{h-1}$.

The algorithm then proceeds to the lower level and repeats the process until it reaches the lowest level of the hierarchical graph that is the actual road network. However, instead of finding the optimal path in the whole road network, it only searches nodes that are part of the projection of spectrum from level $h = 2$. At this lowest level, HTNGD sets $\alpha$ to zero and employs TNGD to find the optimal path from $O$ to $D$ within the reduced search space.

We consider a highly stylized example to illustrate how HTNGD works with $\alpha = 1$. Fig. 2 shows three levels of hierarchy, where the top level only has two nodes. We consider $O$ and $D$ to fall into $C_O^2$ and $C_D^2$, respectively. All sub-communities of these two communities are shown in the second level, $h = 2$. In this level, $O$ and $D$ fall into $C_O^1$ and $C_D^1$, respectively. HTNGD calls TNGD to find the core set, $CS_{OD}^1 = \{C_O^1, C_4^1, C_5^1, C_D^1\}$. Since $\alpha$ is set to one, the spectrum $Q^1$ contains the immediate neighbor communities of $CS_{OD}^1$. Therefore, $Q^1 = \{C_O^1, C_3^1, C_4^1, C_5^1, C_6^1, C_D^1\}$. HTNGD eliminates communities not included in $Q^1$ from further search space, $C_1^1, C_2^1, C_7^1, C_8^1$, and $C_9^1$. Then, HTNGD projects $Q^1$ onto the lowest level of the hierarchy, $G^1$. Finally, HTNGD finds the shortest path between $O$ and $D$ using the reduced search space at this lowest level, $h = 1$. The optimal

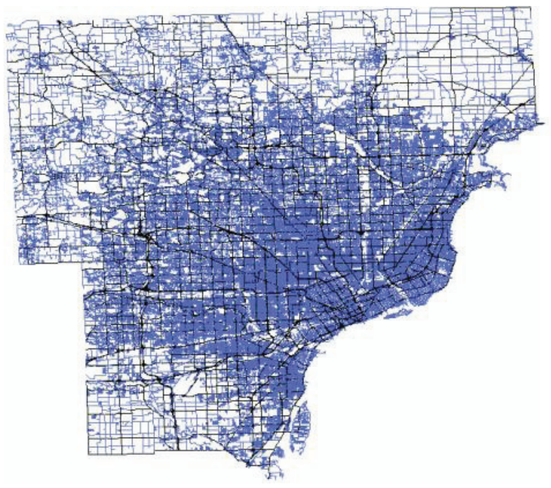

Figure 3: Metro Detroit road network, 465,938 road segments (arcs), 168,806 cross sections (nodes)

shortest path is shown by a bold line in Fig. 2.

## 4 Experimental Results

We study the performance of our proposed algorithm on the road networks of metropolitan Detroit, New York, and San Francisco. We use two sources for extracting their directed graphs. The first source is NAVTEQ (NAVTEQ, 2013) for Metro Detroit. It consists of coordinates of intersections, road segment distances, and speed limits. We extract the graph with its features using ArcGIS Desktop 10. Fig. 3 shows the full road network of Metro Detroit. The second source is the center for Discrete Mathematics and Theoretical Computer Science (DIMACS) at Rutgers University (DIMACS, 2013). It consists of coordinates of intersections, distance graph, and travel time graph for New York and San Francisco. Table 2 shows the number of nodes and arcs of these three road networks. All algorithms are implemented in C++. Experiments are conducted on an Intel 2.53 GHz with 3GB RAM Linux platform.

Table 2: Properties of selected road networks

| | No. of nodes | No. of arcs |
|---|---|---|
| Detroit | 168,806 | 465,938 |
| New York | 264,346 | 733,846 |
| San Francisco | 321,270 | 800,172 |

## 4.1 Generating Time-Dependent Networks

Given the unavailability of time-dependent arc travel times for all arcs of the road networks under study (e.g., ITS coverage is mostly limited to highways), we adopt the following procedure for generating such data. Many transportation studies (e.g., Nannicini et al. (2008); Delling and Nannicini (2012)) have also employed similar artificially generated time-dependent travel time datasets.

Given that the travel time index (TTI) varies by time of day, we rely on the latest TTI as reported by the Texas Transportation Institute for the cities under study to calibrate the traffic speeds for individual arcs at one-minute resolutions for a typical weekday (Schrank and Lomax., 2012). TTI corresponds to the ratio of travel time in a particular period to the travel time at free-flow condition. For example, a value of 1.3 for a certain time of day indicates that a 20-minute trip under free-flow condition takes an average of 20 $\times$ 1.3=26 minutes in that period. Note that a TTI of 1 corresponds to the free-flow of traffic without any congestion. Therefore, we equate this to posted speed limits for individual arcs. During rush hours, TTI significantly exceeds 1 and corresponds to reduced traffic speeds. For instance, for the Metro Detroit region the TTI is 1.2 and 1.28 during morning and afternoon rush hours, respectively. Fig. 4 shows the TTI for the Metro Detroit region in more detail. We adjusted the traffic speeds for every arc of the network, as a function of time of day, to match the average TTI profile at a one-minute resolution. To generate a representative time dependent travel network, we employed the following approach. We selected coordinates for ten stationary congestion spots covering the Detroit Metro network. Based on the distance proximity between the nearest congestion spot and the mid-point of each arc, the travel time index profile for the arcs (at a one-minute resolution) is generated as follows:

$$w_{ij}^t(1 + (TTI^t - 1)\frac{1}{0.25\lambda_{ij} + 1}) \tag{6}$$

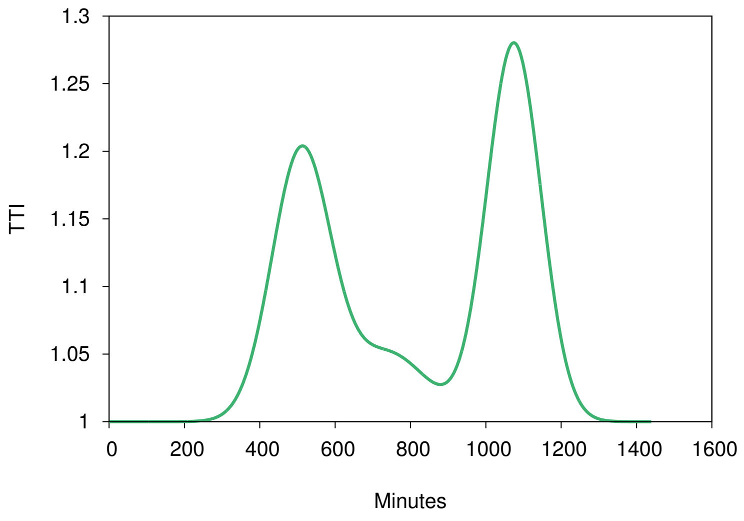

Figure 4: Travel time index (TTI)

where $\lambda_{ij}$ is the distance proximity between the nearest congestion spot and the mid-point of the arc from $i$ to $j$. As designed, different arcs of the network exhibit different travel time index profiles based on their proximity to the congestion spots (nearby arcs will experience the full impact of recurrent congestion and distant arcs will mostly maintain free flow travel conditions). Note that the intention here is not to mimic real-world traffic dynamics but to generate a time dependent network to objectively evaluate the proposed algorithm.

## 4.2 Experimental Setup

We construct the hierarchical representation using travel times under free-flow. Table 3 reports the number of communities identified in each level of the hierarchy using the hierarchical community detection algorithm. Louvain's community detection algorithm establishes the same number of hierarchy levels in New York and San Francisco while extracting one more level for Detroit. This is because the Metro Detroit network is sparser than the other two networks. In the first level ($h = 1$), each community contains just a single node from the network. As the level increases in the hierarchy, more nodes are merged to construct each community. Therefore, there are fewer communities at the higher levels. This algorithm finds the hierarchical communities for each studied road network in less than a second.

For all levels of the hierarchy, we build the hierarchical representation $G^h(V^h,\ A^h, W^h)$ as

Table 3: Number of communities in each level revealed by the community detection algorithm

| $h$ | Detroit | New York | San Francisco |
|---|---|---|---|
| 1 | 168,806 | 264,347 | 321,271 |
| 2 | 67,136 | 79,261 | 93,100 |
| 3 | 21,508 | 18,968 | 23,104 |
| 4 | 5,833 | 4,007 | 5,085 |
| 5 | 1,453 | 952 | 1244 |
| 6 | 457 | 438 | 672 |
| 7 | 368 | - | - |

explained in section 2. At the lowest level, $w^h_{C^h_i C^h_j}(t) = w^t_{ij}$, where $w^t_{ij}$ is the time-dependent travel time of going from node $i$ to node $j$ and $t$ is the arrival time at node $i$. However, in our experiments, for the higher levels, $w^h_{C^h_i C^h_j}(t)$ is the estimated lower bound of travel time from $C^h_i$ to $C^h_j$ based on the straight-line distance between centers of those communities and the speed limit. These estimates can be replaced with more precise information when available, and they may lead to further improvements in computational efficiency. The fixed topology of road networks gives routing algorithms for vehicular networks the benefit of using coordinates; other networks may not have such a privilege. We employ a haversine distance to estimate the distance between any given pair of nodes or communities. Haversine distance $d$ of two vertices $i$ and $j$ is computed using the following formula:

$$\begin{aligned} a =& \sin^2\left(\frac{lat_i - lat_j}{2}\right) + \\ & \cos\left(lat_i\right)\cos\left(lat_j\right)\,\sin^2\left(\frac{long_i - long_j}{2}\right) \end{aligned} \tag{7}$$

$$c = 2\ \mathrm{atan2}(\sqrt{a},\ \sqrt{1-a}) \tag{8}$$

$$d = R\ c \tag{9}$$

where $R$ is earth's radius (3,961 miles).

We set up an extensive experimental evaluation of our proposed routing algorithm. To analyze effects of $OD$ pairs distance on the proposed algorithm, our tests are executed on five different classes of $OD$ pairs distance: less than 5 miles, 5 to 10 miles, 10 to 20 miles, 20 to 30 miles, and 30 to 40 miles.

We first evaluate the HTNGD using 1,000 randomly selected $OD$ pairs in each class from the road networks of Detroit, New York, and San Francisco, resulting in a total of 15,000 $OD$ pairs (i.e., 1000 $\times$ 5 classes $\times$ 3 cities). We randomly select trip start times throughout the day from 1,440 (i.e., 24 hours $\times$ 60 minutes/hour) time windows. We also perform sensitivity analysis for the spectrum control parameter $\alpha$ over five different values of $\alpha$ for the Detroit dataset. The selected values for $\alpha$ are as follows: 1, 2, 3, $L - h$, and $2(L - h)$, where $L$ is the number of levels and $h$ is the level of hierarchy in the algorithm. To analyze the performance of HTNGD under different traffic conditions, we only consider the Detroit dataset. We choose two distinct traffic conditions: free-flow (early morning) and high traffic (afternoon rush-hour).

## 4.3 Evaluation of HTNGD

As noted earlier, vehicle routing navigation systems, whether built-in or portable, lack the ability to rely on online servers and have to compute the route, given an origin/destination pair and departure time, in a stand-alone mode with limited hardware processing/memory capacity. This mostly renders methods that store preprocessed shortest paths, shortcuts, and lower bounds impractical due to their massive memory requirements. The proposed HTNGD algorithms are explicitly designed to overcome these limitations.

In this subsection, we evaluate the performance of the proposed HTNGD algorithms in time-dependent road networks generated for Detroit, New York, and San Francisco. We compare the results of HTNGD to an adaptation of A* algorithm for time-dependent networks. The reader is referred to (Chabini and Lan, 2002) for such adaptations. Time-dependent A* algorithms do not require storage of preprocessed shortest paths, shortcuts, or lower bounds, and hence, qualify for fair comparison with the proposed HTNGD algorithms.

$TNGD(G^1, O, D, h = 1, \alpha = 0)$ works as an adaptation of A* on the time-dependent network $G^1$. This means that TNGD with $\alpha = 0$ on the whole network is a time-dependent A* algorithm (TA*). However, HTNGD on the lowest level calls TNGD with $\alpha = 0$ on the reduced search space. Therefore, for fair analysis of the performance of HTNGD, we compare HTNGD with TA*. Note

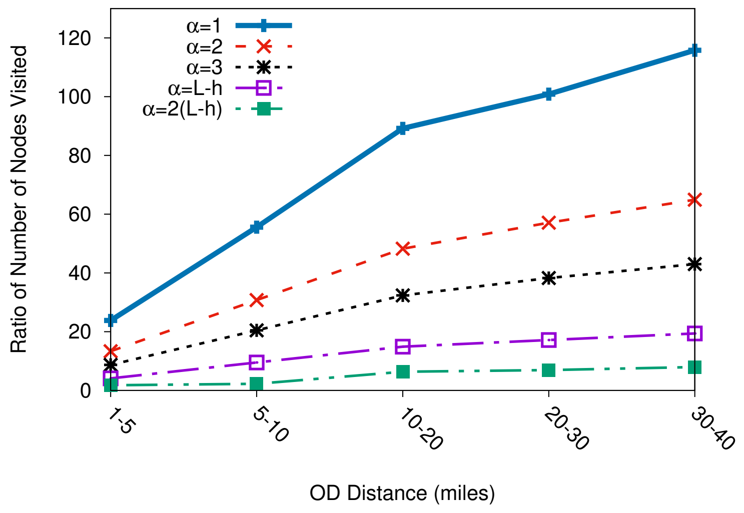

Figure 5: Average number of nodes visited by TA* compared to HTNGD during search

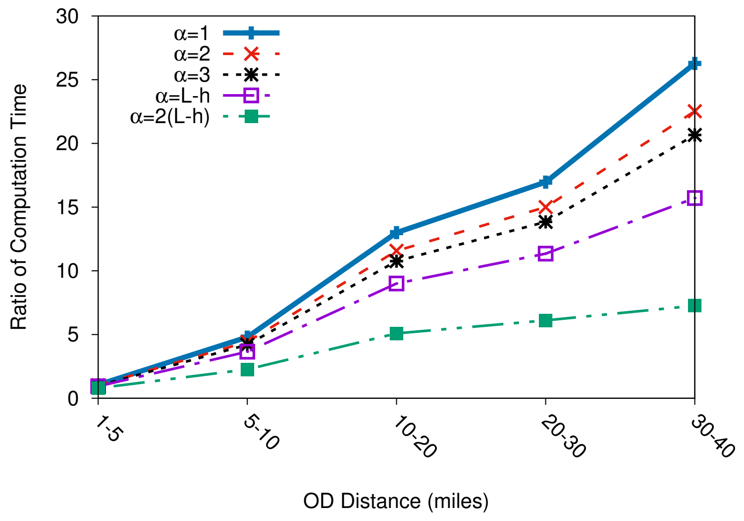

Figure 6: Speedup factor for HTNGD vs TA*

that TA* always finds the optimal shortest path as long as the estimated travel time obtained by the heuristic function is a lower bound of the actual travel time. This is always the case in our proposed TA*.

The ratio of the number of visited nodes in HTNGD compared to TA* on the described test sets are presented in Fig. 5. With an increase in the distance between $OD$ pairs, both HTNGD and TA* explore more nodes to find the path. However, as shown in the figure, HTNGD visits many fewer nodes than TA*. This is primarily attributable to the hierarchical search and projection strategy of HTNGD. For example, for the $OD$ distance class of 10-20 miles and $\alpha = L - h$, TA*

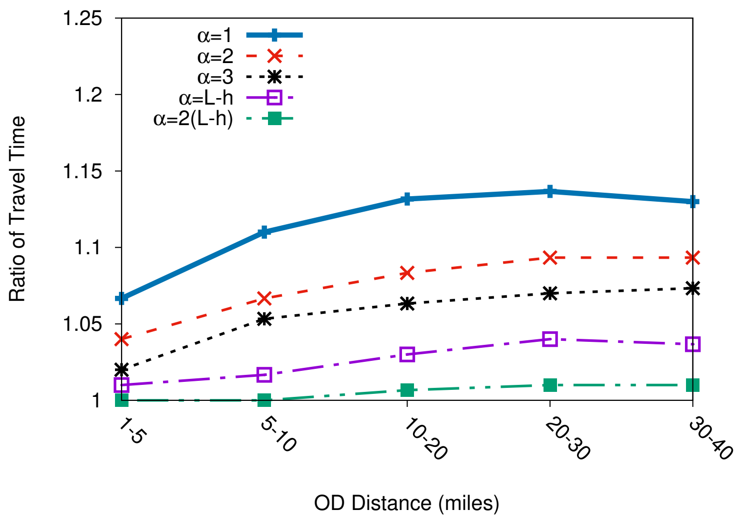

Figure 7: Average travel time of the path provided by HTNGD compared to TA*

explores 14.92 times more nodes than HTNGD. This ratio goes upto 89.21 in the case of $\alpha = 1$ for the same $OD$ class.

Fig. 6 compares the computational time differences of HTNGD over TA*. The results show significant computational efficiency of HTNGD over TA*. For the case of $\alpha = L-h$, HTNGD is 9.0 times faster than TA* for the $OD$ distance class of 10-20 miles, and is over 15.70 times faster for longer distances. In the case of $\alpha = 1$, HTNGD is 26.27 times faster than TA* for longer distances. The results of Fig. 5 and Fig. 6 show that the decrease in the number of visited nodes leads to a faster execution time of the HTNGD. This is due to the fact that the decrease in the number of visited nodes reduces the search space leading to a faster execution time.

In addition, we study the optimality of the path identified by HTNGD. We compare the total travel time of the paths obtained by HTNGD and TA* in Fig. 7. The results vary based on different values of $\alpha$. As noted earlier, proper selection of $\alpha$ is critical in the tradeoff between computation time and optimality gap. The optimality gap is the difference between the travel time of the optimal path and the path obtained by HTNGD. Fig. 7 shows the ratio of travel time of HTNGD to TA*. HTNGD with $\alpha = (L-h)$ results in a trip travel time that is 3.0% more than that of TA* for the $OD$ distance class of 10-20 miles. If needed, one can further decrease the optimality gap by increasing the value of parameter $\alpha$, but at the cost of increasing the execution time.

Figs. 8-10 describe the results with $\alpha = 2(L-h)$ for the Detroit dataset in more detail. These figures present the distribution of the results with minimum, 10 percentile, average, 90 percentile,

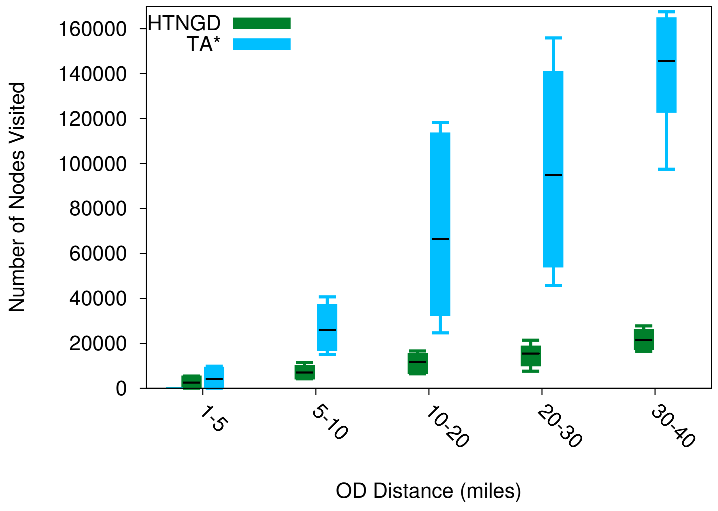

Figure 8: Number of visited nodes of HTNGD ($\alpha = 2(L - h)$) and TA*

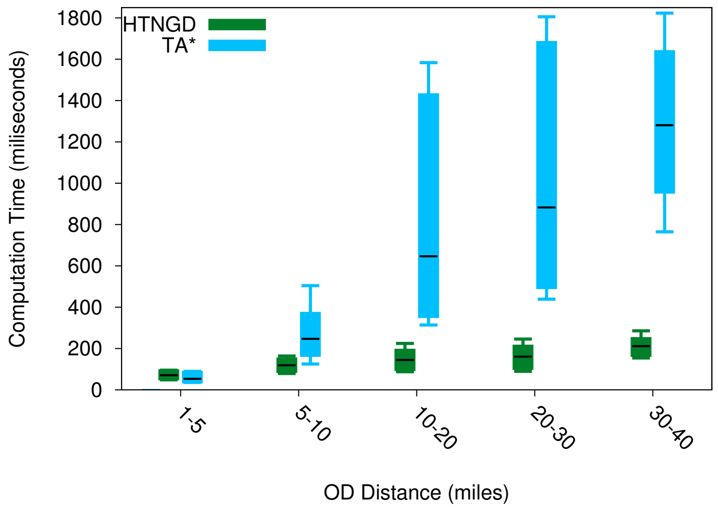

Figure 9: Computation time of HTNGD ($\alpha = 2(L - h)$) and TA*

and maximum values. Fig. 8 presents the distributions of the number of nodes visited during the search. TA* on average visits 6.04 times more nodes than HTNGD. Fig. 9 shows the distributions of the computation time of HTNGD compared to TA*. The computation time of HTNGD over all selected $OD$ pairs is on average 4.85 times faster than that of TA*. Clearly, our proposed algorithm performs even better than TA* for longer $OD$ distances (i.e., 6.05 times faster). The distributions of the obtained results for the total travel time are shown in Fig. 10. HTNGD results in a total travel time that is on average 1.1% longer than those of TA* for all classes of $OD$ pairs. For all the selected $OD$ pairs in the classes of 1-5 and 5-10 miles, HTNGD with $\alpha = 2(L - h)$ finds the

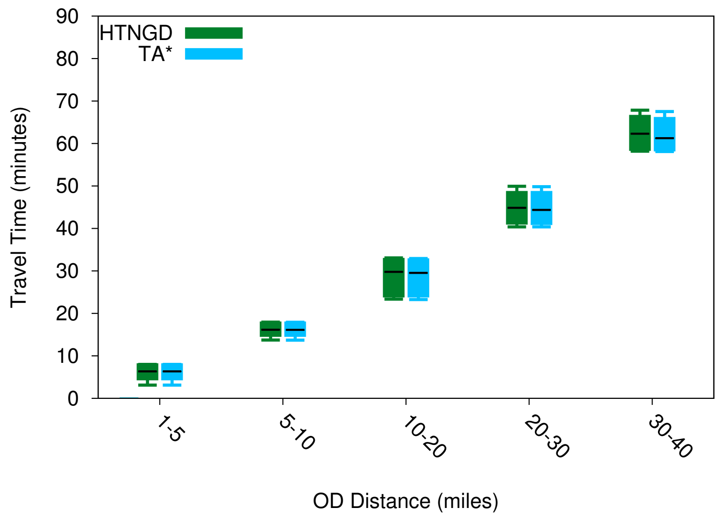

Figure 10: Travel time of HTNGD ($\alpha = 2(L - h)$) and TA*

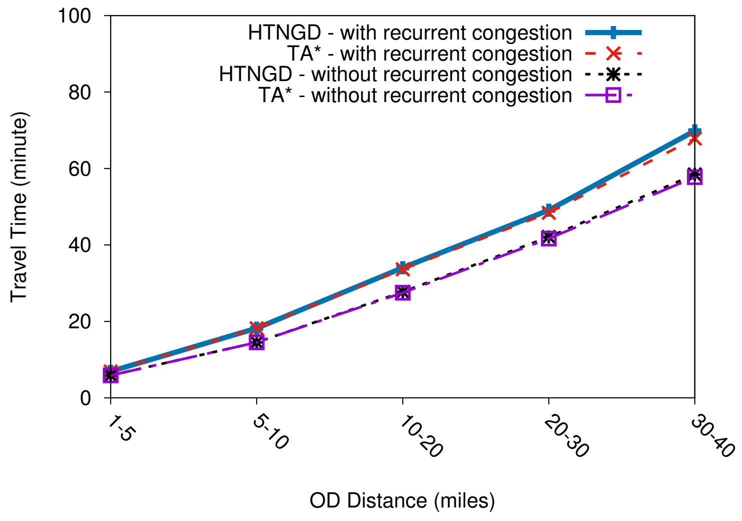

Figure 11: Average travel time of HTNGD ($\alpha = 2(L - h)$) and TA* in early morning vs afternoon rush-hour

optimal paths.

To investigate the impact of traffic conditions on the performance of our proposed routing algorithm, we now compare the performance of HTNGD with the 5,000 selected $OD$ pairs from the Detroit road network under two distinct traffic conditions: trip start times of midnight (closer to free-flow) versus 6:30PM (experiencing significant recurrent congestion). We study the effects of traffic conditions on our proposed algorithm with $\alpha = 2(L - h)$.

Fig. 11 presents the average travel time of HTNGD and TA*. As expected, the results show

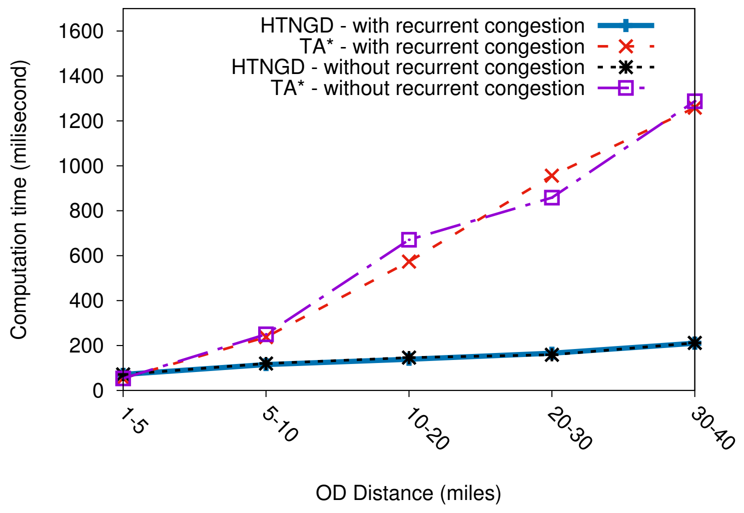

Figure 12: Average computation time of HTNGD ($\alpha = 2(L - h)$) and TA* in early morning vs. afternoon rush-hour

that the average travel time increases for both methods during rush-hour.

Fig. 12 shows the average computation time of HTNGD and TA* under free-flow and evening rush-hour. The results show that HTNGD in both test cases performs almost the same. In addition, TA* in both test cases has almost the same computation time. This is due to the fact that the complexity of both algorithms is independent of arc weights, here interpreted as the road segment travel time (with or without congestion). As a result, there are no significant changes in the performance of each algorithm in terms of computation time regarding the traffic congestion.

We compare HTNGD with the most successful speedup techniques in Table 4. The reader is referred to (Bauer et al., 2010; Geisberger et al., 2012) for information on other relevant algorithms. We analyze data from several papers, and compare pre-processing time, additional storage requirement based on byte per node, and query time. We also present the hardware used in each of the selected studies in the footnote of the table. TNR has the lowest query time, however, it requires 2,760 seconds for pre-processing, and 193 Bytes/node for additional storage space. HTNGD requires the least amount of pre-processing time and storage except than Dijkstra. In addition, query time of HTNGD is reasonable, and it is in terms of milliseconds. In this table, we present the average query time of HTNGD with $\alpha = 2(L - h)$. The query time of HTNGD can be reduced by choosing lower values for $\alpha$. Note that the query time of all the algorithms is less than one second.

From all these results, we conclude that HTNGD not only provides accurate route guidance,

Table 4: Comparison of various methods.

| Method | Data from | Pre-processing Time (s) | Storage requirement (Byte/node) | Query Time (ms) |
|---|---|---|---|---|
| Dijkstra[1] | Bauer et al. (2010) | 0 | 0 | 5,591.6 |
| TNR[4] | Geisberger et al. (2012) | 2,760 | 193 | 0.0033 |
| AF[3] | Hilger et al. (2009) | 129,360 | 25 | 1.1 |
| SHARC[1] | Bauer and Delling (2009) | 4,860 | 14.5 | 0.29 |
| HH | Schultes (2008) | 780 | 48 | 0.61 |
| CALT[1] | Bauer et al. (2010) | 660 | 15.4 | 1.34 |
| ALT[2] | Goldberg et al. (2009) | 780 | 70 | 120.1 |
| TDCALT[1] | Delling and Nannicini (2012) | 1,680 | 256 | 188.2 |
| HTNGD[5] | this paper | 0.98 | 10 | 141.3 |

[1]2.6 GHz AMD Opteron, SuSE Linux 10.2, 16GB RAM

[2]2.4 GHz AMD Opteron, Windows Server 2003, 16GB RAM

[3]2.2 GHz AMD Opteron, SuSE Linux 9.1, 4GB RAM

[4]2.0 GHz AMD Opteron, SuSE Linux 10.3, 8GB RAM

[5]2.53 GHz Intel, Fedora Linux 12, 3GB RAM

but also offers significant computational efficiency over other methods without large memory requirements.

## 5 Conclusion

The expanding coverage of Intelligent Transportation Systems is necessitating the development of real-time algorithms for vehicle routing on time-dependent networks. This paper provides a new approach for solving the time-dependent shortest path (TDSP) problem on large-scale dynamic networks with deterministic time-varying travel time. In particular, we proposed a hierarchical time-dependent shortest path algorithm to solve the TDSP problem that can utilize community-based hierarchical representations of road networks. The proposed algorithm (HTNGD) generates routes in real-time in terms of milliseconds on large-scale networks without having to store a large number of pre-calculated shortest paths and lower bounds. A key property of the proposed algorithm is its low memory requirements. The significant reduction in memory requirements of HTNGD compared to that of other current methods makes HTNGD suitable to be incorporated in vehicle routing navigation systems. Extensive experimental evaluations of the proposed approach on Detroit, New York, and San Francisco road networks demonstrate the computational efficiency and accuracy of the proposed method. We plan to extend this research to energy-efficient routing of plug-in hybrid and pure electric vehicles.

## References

Ahn, B.-H. and J.-Y. Shin (1991). Vehicle-routeing with time windows and time-varying congestion. *Journal of the Operational Research Society*, 393–400.

Bauer, R. and D. Delling (2009). Sharc: Fast and robust unidirectional routing. *Journal of Experimental Algorithmics 14*, 4.

Bauer, R., D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner (2010). Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *Journal of Experimental Algorithmics 15*, 2–3.

Bierlaire, M. and F. Crittin (2004). An efficient algorithm for real-time estimation and prediction of dynamic od tables. *Operations Research 52*(1), 116–127.

Blondel, V., J. Guillaume, R. Lambiotte, and E. Lefebvre (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*(10), P10008.

Buriol, L., M. Resende, and M. Thorup (2008). Speeding up dynamic shortest-path algorithms. *INFORMS Journal on Computing 20*(2), 191–204.

Chabini, I. and S. Lan (2002). Adaptations of the a* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks. *IEEE Transactions on Intelligent Transportation Systems 3*(1), 60–74.

Chou, Y., H. Romeijn, and R. Smith (1998). Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. *INFORMS Journal on Computing 10*(2), 163–179.

Clauset, A., C. Moore, and M. Newman (2008). Hierarchical structure and the prediction of missing links in networks. *Nature 453*(7191), 98–101.

Cooke, K. L. and E. Halsey (1966). The shortest route through a network with time-dependent internodal transit times. *Journal of mathematical analysis and applications 14*(3), 493–498.

Delling, D. and G. Nannicini (2012). Core routing on dynamic time-dependent road networks. *INFORMS Journal on Computing 24*(2), 187–201.

DIMACS (2013). http://www.dis.uniroma1.it/ challenge9/download.shtml.

Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. *Operations Research 17*(3), 395–412.

Fernández-Madrigal, J. and J. González (2002). Multihierarchical graph search. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*(1), 103–113.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports 486*(3), 75–174.

Foschini, L., J. Hershberger, and S. Suri (2011). On the complexity of time-dependent shortest paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 327–341.

Fu, L., D. Sun, and L. Rilett (2006). Heuristic shortest path algorithms for transportation applications: state of the art. *Computers & Operations Research 33*(11), 3324–3343.

Gao, S. and I. Chabini (2006). Optimal routing policy problems in stochastic time-dependent networks. *Transportation Research Part B: Methodological 40*(2), 93–122.

Geisberger, R., P. Sanders, D. Schultes, and C. Vetter (2012). Exact routing in large road networks using contraction hierarchies. *Transportation Science 46*(3), 388–404.

Goldberg, A. and C. Harrelson (2005). Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 156–165.

Goldberg, A., H. Kaplan, and R. Werneck (2007). Better landmarks within reach. *Experimental Algorithms*, 38–51.

Goldberg, A. V., H. Kaplan, and R. F. Werneck (2009). Reach for a*: Shortest path algorithms with preprocessing. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge 74*, 93–139.

Guimera, R. and L. Amaral (2005). Functional cartography of complex metabolic networks. *Nature 433*(7028), 895–900.

Hilger, M., E. Köhler, R. H. Möhring, and H. Schilling (2009). Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge 74*, 41–72.

Jagadeesh, G., T. Srikanthan, and K. Quek (2002). Heuristic techniques for accelerating hierarchical routing on road networks. *IEEE Transactions on Intelligent Transportation Systems 3*(4), 301–309.

Jung, S. and S. Pramanik (2002). An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering 14*(5), 1029–1046.

Kaufman, D. E. and R. L. Smith (1993). Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems 1*(1), 1–11.

Nannicini, G., D. Delling, L. Liberti, and D. Schultes (2008). Bidirectional a search for time-dependent fast paths. *Experimental Algorithms*, 334–346.

NAVTEQ (2013). http://www.navteq.com/.

Nejad, M., L. Mashayekhy, and R. B. Chinnam (2012). Effects of traffic network dynamics on hierarchical community-based representations of large road networks. In *Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 1900–1905.

Nejad, M., L. Mashayekhy, A. Taghavi, and R. Chinnam (2011). State space reduction in modeling traffic network dynamics for dynamic routing under its. In *Proceedings of the 14th International IEEE Conference on Intelligent Transportation Systems*, pp. 277–282.

Newman, M. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E 69*(6), 066133.

Newman, M. (2011). Communities, modules and large-scale structure in networks. *Nature Physics 8*(1), 25–31.

Newman, M. and M. Girvan (2004). Finding and evaluating community structure in networks. *Physical review E 69*(2), 026113.

Palla, G., A. Barabasi, and T. Vicsek (2007). Quantifying social group evolution. *Nature 446*(7136), 664–667.

Palla, G., I. Derényi, I. Farkas, and T. Vicsek (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature 435*(7043), 814–818.

Pons, P. and M. Latapy (2005). Computing communities in large networks using random walks. *Computer and Information Sciences-ISCIS 2005*, 284–293.

Provan, J. S. (2003). A polynomial-time algorithm to find shortest paths with recourse. *Networks 41*(2), 115–125.

Radicchi, F., C. Castellano, F. Cecconi, V. Loreto, and D. Parisi (2004). Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America 101*(9), 2658–2663.

Rajagopalan, R., K. Mehrotra, C. Mohan, and P. Varshney (2008). Hierarchical path computation approach for large graphs. *IEEE Transactions on Aerospace and Electronic Systems 44*(2), 427–440.

Schrank, D. and T. Lomax. (2012). 2012 urban mobility report. Technical report, Texas A&M Univ. Syst., College Station, TX.

Schultes, D. (2008). *Route Planning in Road Networks.* Ph. D. thesis, Universität Karlsruhe.

Song, Q. and X. Wang (2011). Efficient routing on large road networks using hierarchical communities. *IEEE Transactions on Intelligent Transportation Systems 12*(1), 132–140.

Waller, S. T. and A. K. Ziliaskopoulos (2002). On the online shortest path problem with limited arc cost dependencies. *Networks 40*(4), 216–227.

## Biographies

Mark M. Nejad is an assistant professor in the School of Industrial and Systems Engineering at the University of Oklahoma. He received his Ph.D. in industrial engineering and M.Sc. in computer science from Wayne State University. His research interests include network optimization, algorithmic game theory, cloud computing, distributed data analytics, and electric vehicles. He received several best paper awards including: the 2014 INFORMS ENRE best student paper award, and runner-up awards for the 2014 POMS College of Sustainable Operations Best Student Paper Competition and the 2014 INFORMS Service Science Best Paper Award. He is a member of the IEEE, INFORMS, and IIE.

Lena Mashayekhy is an assistant professor in the Department of Computer and Information Sciences at the University of Delaware. She received her PhD degree in computer science from Wayne State University. Her research interests include cloud computing, parallel algorithms, sustainable computing, game theory, and electric vehicles. She has published more than thirty peer-reviewed papers in venues such as IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers. She is a member of the IEEE, ACM, and INFORMS.

Ratna Babu Chinnam is a Professor in the Department of Industrial & Systems Engineering at Wayne State University (Detroit, U.S.A.). He is the author of over 150 journal articles, conference proceedings, and technical reports in the areas of Supply Chain Management, Big Data Analytics, Sustainability, and Diagnostics/Prognostics. Most of his research is funded by the U.S. federal agencies (NSF, DOT, DOD, DOE, VA) and he carried out extensive funded collaborative research with international companies such as Ford Motor Company, General Dynamics, Chrysler and consulted for such companies as Dominos, Urban Science, Energy Conversion Devices, Federal Mogul, and Sirius Satellite Radio. He supervised over 24 PhD students that mostly occupy academic positions around the world. He is a member of Alpha Pi Mu, INFORMS, and NAMRI.

Anthony Phillips is a Senior Technical Leader in Research and Advanced Engineering at Ford Motor Company (Dearborn, U.S.A.). He received his B.A. degree in physics from Gustavus Adolphus College, St. Peter, Minnesota in 1990 and his M.S. and Ph.D. degrees in Mechanical Engineering Control Systems from the University of California, Berkeley in 1993 and 1995, respectively. His research interests include vehicle energy management, distributed system control and control system development tools and methods. He holds over 30 U.S. and international patents in automotive controls. He also serves as Ford's Corporate Advisory Board Member to the International Council of System Engineering.