Hierarchichal TDSP

Keywords: Time dependent shortest path; traffic

1 Shortest path on time dependent hierarchical networks

In this section, we propose an efficient algorithm to find near-optimal shortest paths on a time dependent hierarchical network. The hierarchical search strategy is a well-studied topic, and Chou et al. (1998), Liu (1997), Jagadeesh et al. (2002), Shapiro et al. (1992), Car and Frank (1994), and Song and Wang (2011) provided various algorithms and heuristics to solve it. Though most of them are either for static networks or involves a lot of manual effort (for example, segregating all the roads into different grids). The method proposed in this section avoids any manual work, and thus, is scalable. It also leverages the fact that side streets are usually less affected by traffic than highways and other main arteries, and creates the hierarchy accordingly.

In this paper, we consider a special class of networks for truck routing called *FIFO networks*, in which vehicles travel on the arcs in a first-in-first-out manner. This is a pragmatic assumption to make for a logistic network, since most of the road networks, in reality, exhibit FIFO property. If $a_{ij}(t)$ is the arrival time at node j, departing from node i at time t on arc (i, j), the fundamental property of a FIFO network is given by:

$$a_{ij}(t^{'}) \geq a_{ij}(t) \quad \forall \quad t^{'} \geq t \tag{1}$$

In other words, $a_{ij}(t)$ is a non-decreasing quantity for a FIFO network. Dean (2004) has discussed about FIFO networks and its properties and benefits in greater detail. He also proposed an exact Label-Setting (Dijkstra's) Algorithm to find optimal shortest path on time dependent networks, that we are going to use in our algorithm.

1.1 FIFO model used in this paper

In a typical city, the roads that experience traffic are usually motorways, state highways and other main roads. Side streets and neighborhood roads almost never get much traffic because people use these roads to enter the main streets as soon as possible. Keeping this in mind, following assumptions have been made for the network in consideration:

- No traffic on side streets, and travel speed doesn't change with time.
- Motorways, state highways, and other main arteries follow the following congestion pattern (τ: Travel time without any congestion):

Time (Hour) of the Day	Travel Time
0 - 7	τ
8 - 9	2τ
11 - 15	1.25τ
17 - 18	2τ
20 - 24	τ

Travel time at any instant between 7AM and 8AM is a linear interpolation between the points (7,τ) and (8,2τ). For example, travel time at 7:45AM is 1.75τ. Same is true for instances

between 9AM - 11AM, 3PM - 5PM and 6PM - 8PM. Congestion is modeled this way to mimic real traffic pattern. In a typical city, roads are usually empty until 7AM. Traffic starts increasing after 7AM as people start going for work and school. The city experiences peak traffic at around 8AM, for an hour. After 9AM, it again starts decreasing until it reaches a steady state of mild traffic throughout the day. It again peaks in the evening, and then decreases until there is little to no traffic, after 8PM.

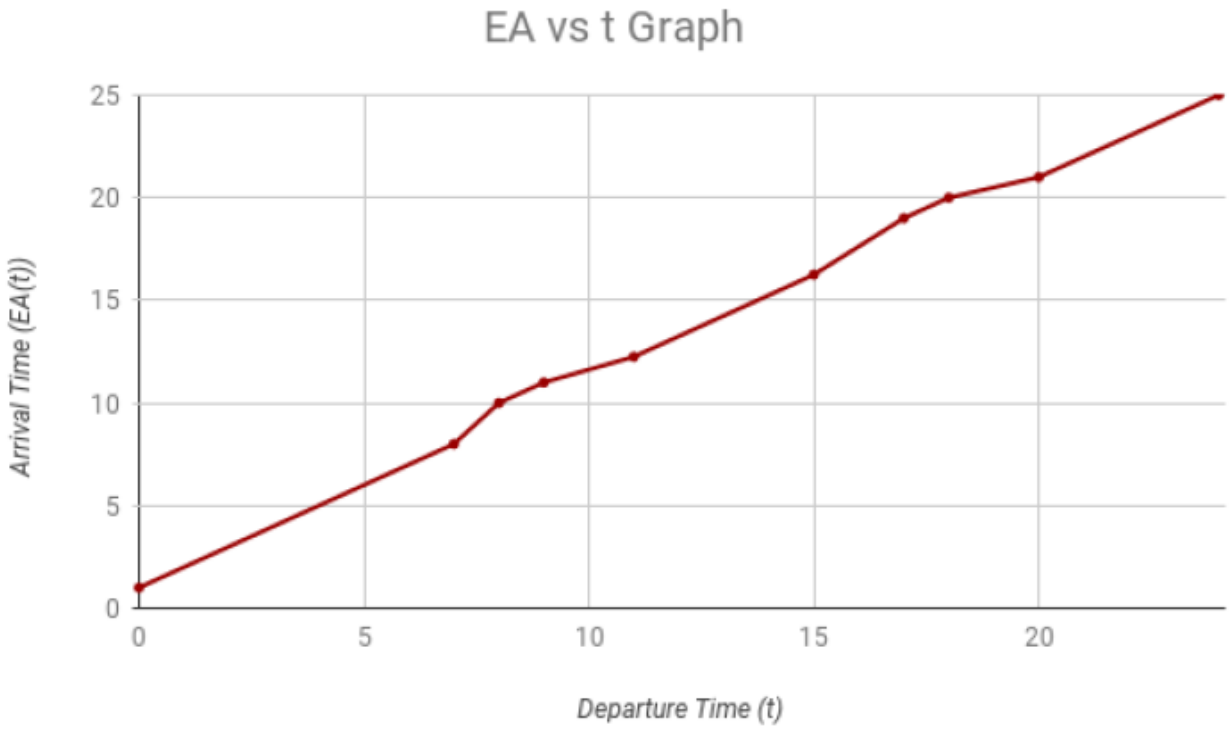

Figure 1: A typical arrival time graph as a function of time (For $\tau = 1$ hour)

1.2 Hierarchical network structure

Two-level hierarchical network is considered for this problem. The upper level consists of all the roads that experience traffic such as motorways, state highways and major arteries. The lower level consists of all the other roads that experience little to no traffic such as side streets and neighborhood roads. Since the traffic condition is assumed to be dependent on the type (or class) of road, the segregation of roads into two levels is trivial. A typical example of network hierarchy for North Buffalo is shown in figure 2. As we can see, the major roads in the upper level partition the network into many natural grids or pockets. Therefore, if going from one node to another within a grid, one doesn't need to travel on major roads (which experience traffic). The shortest path, in this case, can be found by using standard Dijkstra's algorithm. To find shortest path between two nodes which are across grids requires strategy, and is discussed in detail in section 1.5.

1.3 Dijkstra's algorithm for time-dependent networks

Dean (2004) provided a label-setting algorithm for FIFO networks, that we'll discuss here. Since the input to a label setting algorithm must be a directed network, we can substitute the undirected

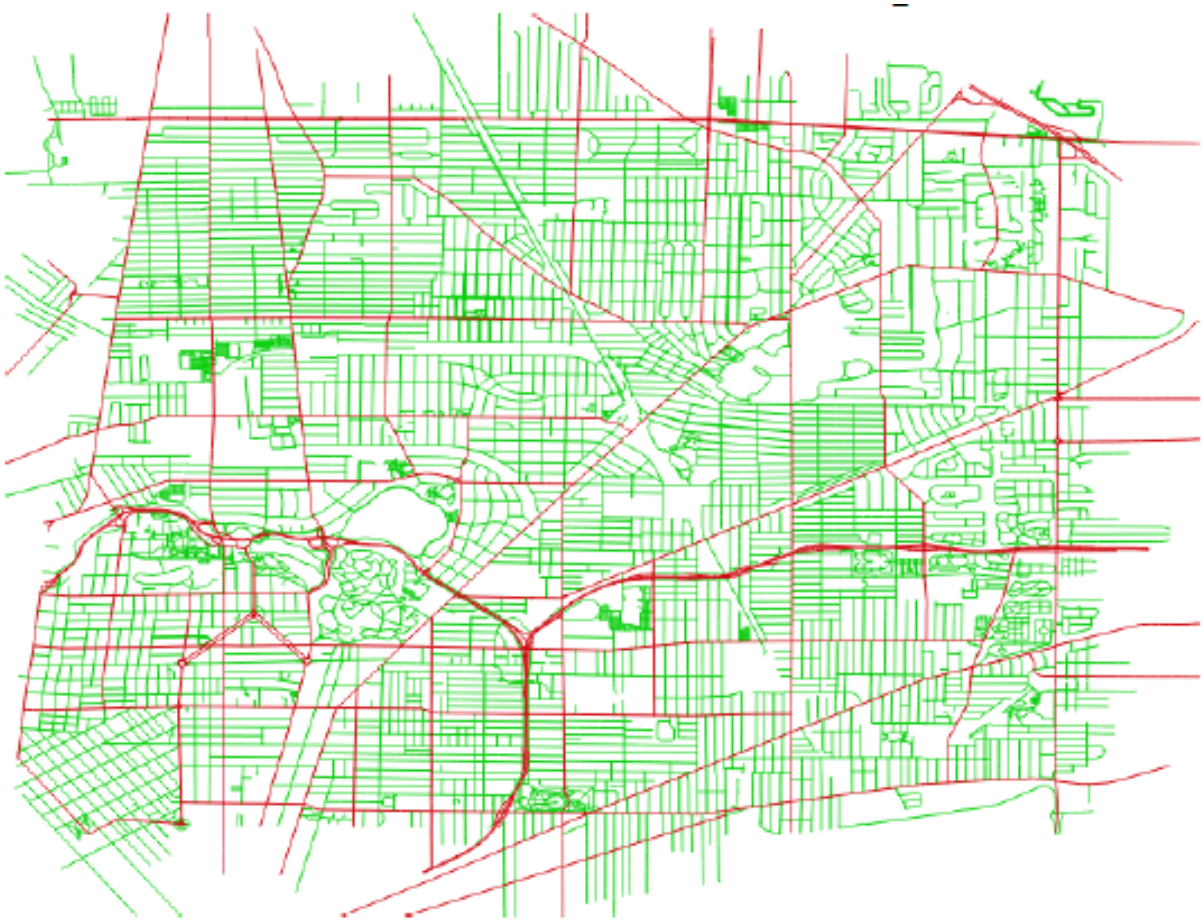

Figure 2: A hierarchical network structure of North Buffalo. Upper level roads are shown in red, and lower level ones are shown in green.

arcs (two-ways roads) in a city network with two directed arcs in opposite directions. Let the directed network be denoted by $G = (N, A)$, where N is the set of all the nodes, and A is the set of all the arcs. The arrival time function $a_{ij}(t)$ for every arc $(i, j) \in A$ gives the time of arrival at j, if departed from i at time t. Also, let $EA_{sd}(t)$ denotes the time of arrival at destination node d using shortest path, if started from source node s at time t. Then, using the label-setting algorithm, we can compute $EA_{s*}(t)$, which is the time of arrival at all the other nodes using shortest paths, starting from node s at time t. Pseudo-code for the algorithm is given below:

Algorithm 1: **Label-Setting (Dijkstra's) Algorithm**

Initialization:

$EA_{si}(t) \leftarrow \infty \quad \forall \quad i \in \{N : i \neq s\}$

$EA_{ss}(t) \leftarrow t$

$S \leftarrow N$

Main Loop:

While $S \neq \phi$

  Select $i \in S$ minimizing $EA_{si}(t)$

  $S \leftarrow S - \{i\}$

For all j such that $(i, j) \in A$
$EA_{sj}(t) \leftarrow min\{EA_{sj}(t), a_{ij}(EA_{si}(t))\}$

The arrival time function $a_{ij}(t)$ in *Algorithm 1* follows the FIFO model discussed in section 1.1, and is given as (τ: Travel time without any congestion):

$$a_{ij}(t) = \begin{cases} t + \tau, & t \in [0, 7) \\ t + (1 + \frac{t-7}{8-7})\tau, & t \in [7, 8) \\ t + 2\tau, & t \in [8, 9) \\ t + (2 - 0.75(\frac{t-9}{11-9}))\tau, & t \in [9, 11) \\ t + 1.25\tau, & t \in [11, 15) \\ t + (1.25 + 0.75(\frac{t-15}{17-15}))\tau, & t \in [15, 17) \\ t + 2\tau, & t \in [17, 18) \\ t + (2 - \frac{t-18}{20-18})\tau, & t \in [18, 20) \\ t + \tau, & t \geq 20 \end{cases} \tag{2}$$

Although *Algorithm 1* can be used to find the optimal shortest path between a source node and all the other nodes in a FIFO network, it's computationally very expensive to run it on a large graph. For example, a city like Buffalo has road network with ——- nodes and —— arcs. The purpose of hierarchical search algorithm proposed in section 1.5 is to overcome this problem.

1.4 Preprocessing

Before using the actual algorithm, one-time preprocessing needs to be done in order to make the algorithm run faster. We call all the nodes (or intersections) which are on the upper level network, as 'major nodes'. All the other nodes which only connect the lower level links are called 'minor nodes'. The idea behind hierarchical search algorithm is that if we want to find the shortest path from one minor node to another 'distant' minor node, we need to enter the upper level network as soon as possible through a nearby major node. Also, to get to the destination we need to exit the upper level network through a major node nearby destination node. The path from entry node to exit node can be found by applying *Algorithm 1* on the upper level network. Now, if the source node itself is a major node, it can be treated as the entry node. Same applies to the destination node.

In Figure 3, a path from the source node (S) to the destination node (D) in a two-level network hierarchy has been shown. There are 4 entry points (S1, S2, S3 and S4) corresponding to the source node. Similarly, there are 4 exit points (D1, D2, D3 and D4) corresponding to the destination node. In order to go from S to D, the vehicle travels on the lower level roads until it reaches S1. From there, it uses *Algorithm 1* to find a shortest path on the upper level network to D1. Then, it exits the upper level at D1 and travels to D using the lower level roads. In the preprocessing step, we need to find those entry and exit points for each minor node, and the corresponding distances from the minor node to those points. To incorporate diversity, we try to find 4 entry and 4 exit points (one in each direction) for every minor node. Steps to find the entry and exit points are following:

- *Step 1*: Let M be the set of all the minor nodes. For $m \in M$, start by finding an entry node in the North direction. To do that, import the complete network (without any hierarchy), and remove all the nodes and the arcs which are below m, from it. Arcs, having end points below m, can be considered below m. Let the network that is left be denoted by G_N.

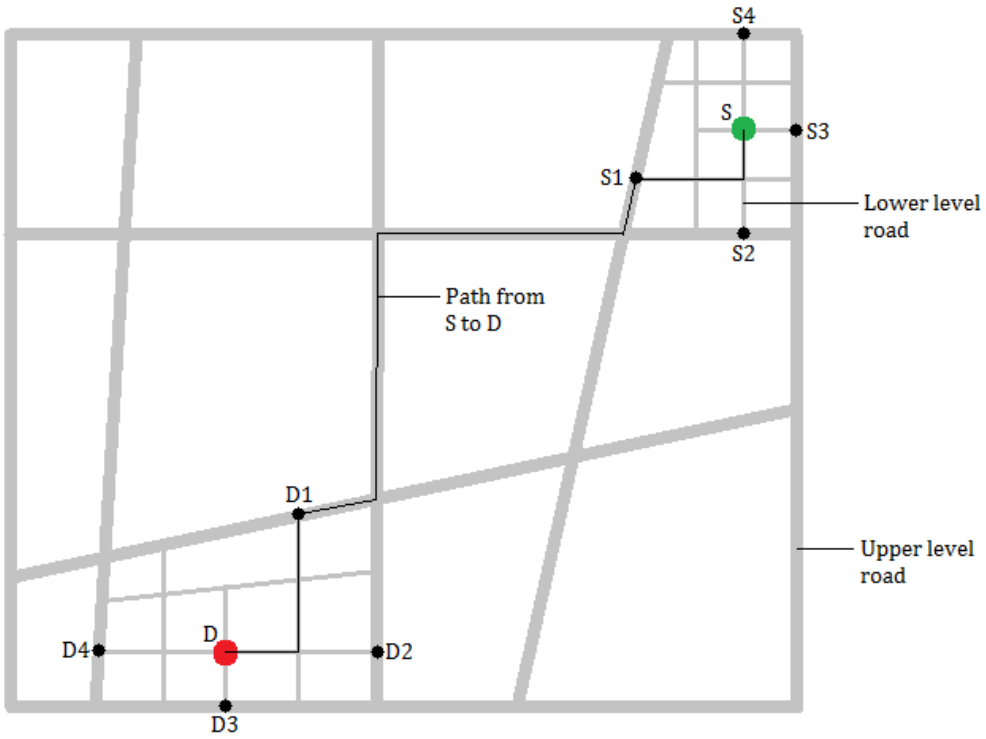

Figure 3: An illustration of two-level hierarchy, and how the shortest path algorithm works

- *Step 2*: Run the standard Dijktsra's algorithm on G_N, with m as the source node. Terminate the algorithm as soon as the first major node is permanently labeled. This major node (call it S1) is the entry point (to upper level network) for m in the North direction. We don't need G_N to be time-dependent for this step because the shortest route between m and S1 found by Dijkstra's algorithm will have all the lower level roads in it, and we have assumed no traffic on lower level roads. Therefore, the travel time from m to S1 would be constant, irrespective of start time at m. Store S1 and travel time to S1 in an array dedicated for m.

- *Step 3*: Run the backward Dijkstra's algorithm on G_N, with m as the destination node. Terminate the algorithm as soon as the first major node is permanently labeled. This major node (call it D1) is the exit point (from upper level network) for m in the North direction. Store D1 and travel time from D1 to m in the same array that is dedicated for m.

- *Step 4*: Repeat steps 1 - 3 to find the entry and exit points in other 3 directions as well. Remove nodes and arcs above m for South, to the left of m for East, and to the right of m for West. In some cases and for some directions, we may not get an entry or exit point if there are no major nodes in that direction or if the path from m to major nodes doesn't exist. In other cases, we may get the same major node as entry or exit points in two different directions.

- *Step 5*: Repeat steps 1 - 4 for all the minor nodes and store the results in a table, each row of the table representing a minor node and its entry/exit points.

- *Step 6*: For a major node n, store n itself as its entry and exit points, and append to the table created in Step 5. Do this for all the major nodes.

1.5 Hierarchical search algorithm

After preprocessing is done, we have everything we need for the algorithm. The algorithm follows a simple logic that if source and destination nodes are in the same grid, simply apply the standard Dijkstra's algorithm on the complete graph, as it will terminate quickly due to source and destination being in close proximity. Also, it won't have to consider time-dependency because all the roads inside a grid are lower level roads, and thus, shortest route from source to destination will consist of lower level roads with no traffic. Now if source and destination are in different grids, the shortest path follows the sequence: *source node* $\rightarrow$ *entry point of source node* $\rightarrow$ *exit point of destination node* $\rightarrow$ *destination node*. We already have the travel times for 'source node to entry points' and 'exit points to destination node' in the preprocessed data. We just need to find the time dependent shortest path from entry points of source node to exit points of destination node using *Algorithm 1*, and add the three travel sequences and their corresponding travel times to get the final result. Since there are up to 4 entry and exit points for each minor node, there can be up to 16 possible paths for each pair of source and destination nodes. We pick the one with the minimum travel time.

To check if the source node and the destination node are in the same grid, we use a function **ST_Intersects** available in PostGIS. This function essentially checks if two geometries/geography spatially intersect in 2D. For a pair of source and destination, if a straight line from source node to destination node doesn't intersects the upper level network, they are in the same grid. If they do intersect, it is highly likely that they are in different grids. The only case where they would intersect and still be in the same grid is if the grid is non-convex shaped. But if that's the case, it would be better still to switch to the upper level network than trying to travel around it be in the same grid. Therefore, in any case, switch to the upper level if they intersect. The pseudo-code for hierarchical search algorithm is given in *Algorithm 2*.

Algorithm 2: **Hierarchical Search Algorithm**

For a source node s, destination node d, complete graph G and upper level graph G_U,
If ST_Intersects is **FALSE**:
  Use Dijkstra's algorithm on G to find shortest path and terminate when d is labeled.
Else:
  $\{S_1, S_2, S_3, S_4\} \leftarrow$ Entry points of s
  $\{D_1, D_2, D_3, D_4\} \leftarrow$ Exit points of d
  For i from 1 to 4:
    For j from 1 to 4:
      $\tau_1 \leftarrow$ Travel time from s to S_i
      $\tau_2 \leftarrow$ Travel time from S_i to D_j obtained by running *Algorithm 1* on G_U
      $\tau_3 \leftarrow$ Travel time from D_j to d
      $T_{ij} \leftarrow \tau_1 + \tau_2 + \tau_3$
  Final travel time $\leftarrow min\{T_{11}, T_{12}, \ldots, T_{44}\}$

Algorithm 2 has been explained for a 'single source, single destination' case. But it can easily be extended to 'single source, multiple destinations' variant without increment in run-time, since Dijkstra's algorithm finds shortest paths to all the other nodes, starting from source node.

References

Adrijana Car and Andrew Frank. General principles of hierarchical spatial reasoning-the case of wayfinding. In the Proceedings of the 6th International Symposium on Spatial Data Handling, volume 2, pages 646–664, 1994.

Yu-Li Chou, H Edwin Romeijn, and Robert L Smith. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. INFORMS journal on Computing, 10(2): 163–179, 1998.

Brian C Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms. Rapport technique, Massachusetts Institute of Technology, 2004.

George Rosario Jagadeesh, Thambipillai Srikanthan, and KH Quek. Heuristic techniques for accelerating hierarchical routing on road networks. IEEE Transactions on intelligent transportation systems, 3(4): 301–309, 2002.

Bing Liu. Route finding by using knowledge about the road network. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 27(4):436–448, 1997.

Jacob Shapiro, Jerry Waxman, and Danny Nir. Level graphs and approximate shortest path algorithms. Networks, 22(7):691–717, 1992.

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