Trafforithm
A Traffic-aware Shortest Path Algorithm in Real Road Networks with Traffic Influence Factors

Lin Qi and Markus Schneider
Department of Computer & Information Science & Engineering, University of Florida, Gainesville, Florida, U.S.A.
{*lqi, mschneid*}*@cise.ufl.edu*

Keywords: Shortest Path Problem, Road Networks, Traffic Influence Factors, Modeling, Algorithm.

Abstract: The shortest path computation between two given locations in a road network is an important problem that finds applications in a wide range of fields. There has been a lot of research efforts targeting at the preciseness and performance of finding shortest paths in road networks. However, rarely of them have really taken into account the influence of traffic factors such as traffic lights, road conditions, traffic jams and turning cost. In other words, existing approaches are rather purely based on the topology of the network, but forgot that there are multiple factors in a real road network that impact the accuracy of the algorithm. The contribution of our paper is twofold. First, we present a generic two-layered framework for moving objects in road networks environment and demonstrate the important role of traffic factors on path finding and route planning. Second, we develop an efficient parallel shortest path algorithm in road networks with the consideration of traffic influence factors. Detailed analysis presented shows that our parallel TRAFFic-aware shortest path algORITHM (*Trafforithm*), is accurate and practical.

1 INTRODUCTION

Shortest path problems are among the most popular network flow optimization problems, with interesting applications in a range of fields. Given two locations *s* and *e* in a road network, a distance query returns the minimum network distance from *s* to *e*, while a shortest path query computes the actual route that achieves the minimum distance. These two types of queries find important applications in practice, and a plethora of solutions have been proposed in past few decades.

The basic concept of shortest path problem is to model the specific problem in a suitable graph and to compute a path with the minimum travel cost to solve it. This is also known as a *routing algorithm*. While it is relatively simple to come up with an algorithm that just solves the problem, it is much harder to develop an efficient, yet precise algorithm. Furthermore, the problems vary in different scenarios. For instance, in a real-life road network, car journeys and public transportations are regulated by traffic rules, restricted by traffic lights, and are greatly influenced by traffic jams. In addition, travelling along a route with less left turns can be less time-consuming compared to another route of the same length but with more left turns. We call these factors as *Traffic Influence Factors (TIFs)*. As indicated previously, most of current approaches neglect the influence of traffic factors, which lacks some practical value. The reason is that the shortest path may not always be the fastest path (i.e. least cost) because of the influence of traffic factors. The detailed discussion regarding TIFs will be given out later in Section 3.2. By considering this kind of effect, we can also provide users with different route choices to avoid traffic congestions.

To address this problem, we would like to propose our traffic-aware shortest path algorithm, *Trafforithm*, which takes traffic conditions and turning costs into consideration. People may argue that this kind of information can only be stored aside from the network topology, and hence it would lower the performance of computing shortest paths when requesting information. However, actually instead of regarding traffic information as outside auxiliary contents, we model this kind of information inside our data model with a user-defined attribute, namely *theme_attr*. In this field, various kinds of information can be stored as a bit sequence, an integer, or it may have a complex type whose values represent combinations of n values. Examples of thematic attributes range from speed limits of a road to the capacity of an oil pipe. Therefore we could store the traffic information of the

Qi L. and Schneider M..
Trafforithm - A Traffic-aware Shortest Path Algorithm in Real Road Networks with Traffic Influence Factors.
DOI: 10.5220/0005350701050112
In *Proceedings of the 1st International Conference on Geographical Information Systems Theory, Applications and Management* (GISTAM-2015), pages 105-112
ISBN: 978-989-758-099-4

road network to this field attached to each point in the network. With the help of our model, we then derive some retrieval functions to peel off the hard-coded information stored in our network model.

The rest of the paper is organized as follows: we summarize the related work in Section 2. Section 3 shows the application-side analysis which would stimulate our model needs and query supports. Section 4 presents our generic two-layered framework for road networks. The problem definition and the body of the algorithm are given in Section 5. Some key ideas on how to evaluate the proposed algorithm are presented in Section 6. Finally, Section 7 concludes the paper together with some directions of possible future improvements.

2 RELATED WORK

In this section, we will introduce the related work and classify them into several categories. Tons of works and contributions based on the shortest path problem and route planning were proposed and carefully compared. *Dijkstra's* algorithm (Dijkstra, 1959) is the fundamental shortest path algorithm. It computes the shortest paths from a single source node to all other reachable nodes in the graph by maintaining tentative distances for each node. A straightforward improvement of Dijkstra's algorithm is *bidirectional* search (Dantzig, 1962). An additional search from the target node is performed in backward direction, and the whole search process terminates as soon as both directions meet. *A* search* (Hart et al., 1968) algorithm integrates a heuristic into a search procedure. Instead of choosing the next node with the least cost, the choice of node is based on the cost from the start node plus an estimate of proximity to the destination. (Awasthi et al., 2005) proposed a rule based method for evaluating the fastest path on urban networks. This method uses a statistical approach based on clustering to compute rules useful in predicting fastest paths. (Qi and Schneider, 2014) came up with two algorithms to address the issue of shortest path computation delay, leading to a realtime response system. (Kanoulas et al., 2006) proposed a traffic speed pattern named CapeCod by classifying the time based on traffic flow. The paper used A* algorithm to solve the fasted problem in various departure times. A mining algorithm for travel time evaluation was proposed in (Lu et al., 2008), which used the mined knowledge to predict the future traffic situations. And a continuous fastest path planning algorithm (Lu et al.,) is proposed. It takes into account the effects of traffic influence factors to avoid the traffic congestions. Compared to our strategy, this approach requires two rounds of pre-computation, and does not consider the turning cost. From network analysis standpoint, researchers as in (Jiang, 2007) and (Jiang and Liu, 2011) target at topological analysis as well as semantic analysis of urban networks. The key idea in (Jiang and Liu, 2011) is to seek a route with fewest turns based on the concept of natural roads. However, they did not distinguish left turn cost from right turn cost, which is inaccurate in shortest and fastest path computation.

Due to its ubiquitous usage over the web and real life, shortest path search on graphs has also become a major topic of interest over the last decade in commercial systems. A good case study from (Demiryurek et al., 2010) is presented based on commercial systems such as Navteq, TIGER, and TOMTOM etc, Google maps (Google,) and Navteq (Navteq,) have been actively providing real road network data set for testing. TOMTOM (TOMTOM,) has developed a technology called *IQ routes* in the sense that the actual speed and travel time per roadstretch is measured per time of day.

3 MOTIVATION

This section would give the motivation behind our model and TIFs. First, we will introduce our model motivation and state why database support is essential in Section 3.1. In Section 3.2, we will develop the definition of TIFs, and analyze how they influence our shortest path query results.

3.1 Model Motivation and Database Support

The Geometric aspects of a road network comprise junctions, routes, boundary points, and crossover points that are embedded in space and characterized by precise locations. In order to model the real road network correctly and precisely, we regard the network as a collection of points, and each point is attached with a label. In our road network model, we assign thematic attribute values to all points in the network. These attribute values can be used to identify network components like routes and interaction points. We call these values labels, and each point in the Euclidean plane is mapped to a set of labels.

The major reason for the lack of a spatial network data type in spatial databases is that it has not been formalized due to the inherent complex nature of spatial networks. The lack of model for spatial network has led to the development of only a very

small number of operations on them. Since the intention is to incorporate the spatial network data type in a database, we have to create an efficient implementation concept of spatial networks and incorporate it inside the database. Based on the model of the spatial network, we also explore ways of querying the spatial network using a SQL-like language, such as network algebra in (Kanjilal and Schneider, 2011) and SQL extentions for moving objects in spatial networks in (Qi and Schneider, 2012).

3.2 Traffic Influence Factors (TIFs): Definition and Importance

Traffic Influence Factors (TIFs) are a collection of factors that have influence on traffic route planning and result in effect on traveling times in a road network environment. TIFs have so far not been extensively included in this area and it is important to consider them because they have a huge influence on the travelling time of a certain path. In reality, people would like to find the *Fastest Path* (FP) to the destination rather than purely the shortest path in most cases. However, there are cases that a shortest path may not be the fastest one since more left turns and passing more numbers of traffic lights are more time consuming than the distance difference. As a result, the TIFs should be considered while computing the real shortest paths between the source and destination.

The factors we will consider here include the *right-turn effect*, the number of traffic lights and other factors. The number of right turns along the path plays a very important role of the total traveling time.

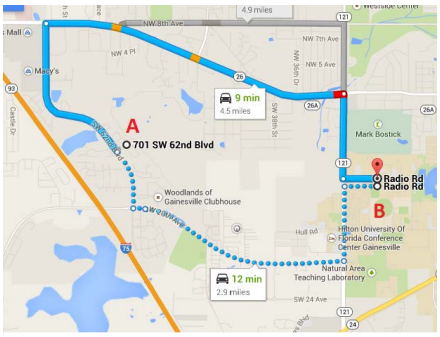

Figure 1: Right-turn effect illustration.

For example in Figure 1, it shows that driving along the solid route will be faster than the dotted one even with 30 percent longer distance. One of the reasons is that the solid route consists of three right turns as the dotted one consists of three left turns.

4 A GENERIC TWO-LAYERED FRAMEWORK FOR ROAD NETWORKS

In this section, we lay out our proposed generic two-layered framework for road networks. We would give a formal and clear definition of our model. Besides, we will also show how our model describes the network and how it formulates the basis for distance and shortest path queries. Due to the page limit, we would briefly go through our model, one could find the detailed parts of the model could be found in (Kanjilal and Schneider, 2011), (Qi and Schneider, 2012).

A road network is a three dimensional spatially embedded and labeled graph. Road networks are consist of routes (i.e. roads), junctions (where two or more routes intersect), and crossovers (where two or more routes intersect but they do not geometrically join). With the help of label space, we could express a road network with the help of spatial mappings. A spatial mapping of A, where A is a label type, is a total mapping $\upsilon : R^3 \rightarrow 2^A$. The set of all spatial mappings of type A is denoted by $[A]$, that is, $[A] = R^3 \rightarrow 2^A$. When applied to a set X, the function υ is iteratively applied to all the elements of X, i.e. $\upsilon(X) = \{\upsilon(p) | p \in X\}$.

With the help of this framework, next let us move on to the label space. Any label we considered in our model is a tuple of the form $(id\_attr, access\_pt, theme\_attr)$. The id_attr is the route identifier and uniquely represents a particular route in a network. This can be a route name or a route number. The $access\_pt$ is a boolean field and a value of true for this field indicates that the particular point is an access point otherwise it is a false. A $theme\_attr$ value may have a simple type such as integer or string, or it may have a complex type whose values represent combinations of n values. Examples of thematic attributes range from speed limits of a road to the capacity of an oil pipe.

In this way, instead of regarding traffic information as outside auxiliary contents, we could model traffic information inside our data model. The next question is how to retrieve such information for computing in algorithms. Therefore we define a retrieval function called $Theme\_Attr$ to extract all information from $theme\_attr$. It takes as argument a spatial mapping υ of type A and computes the set of all $theme\_attr$ values by using the projection operator $\sqcap$. Since the $theme\_attr$ is assumed to be always the 4th attribute in a label, we use $\sqcap_{\{4\}}$ to retrieve its value. The function $Theme\_Attr$ is therefore defined

for $\upsilon \in [A]$ as:

$$Theme\_Attr(\upsilon) = \{\sqcap_{\{4\}}(l) | s \in \upsilon(R(\upsilon)), l \in s\} \quad (1)$$

5 Trafforithm: TRAFFIC-AWARE SHORTEST PATH ALGORITHM

In this section, we will first give the problem definition of shortest path problem in Section 5.1. Followed by that, the body of the algorithm as well as some implementation ideas and speedup techniques will be discussed in Section 5.2.

5.1 Problem Definition and Preliminaries

In this subsection, we will give the formal definition of our *Shortest Path Problem* (SPP). Let $G = (V, E, w)$ be a road network with a vertex set V, an edge set E and a weight function. Each edge $e \in E$ is associated with a weight $w(e) : e \to R$. In this case, the weight function will give us the travel cost (which refers to the travel time here) for each given edge. The traveling time of a trip could be estimated from the road speed restrictions and the road length, and we call this traveling time as *scheduled traveling time*. This graph G is directed, since some of the roads restrict the traveling directions for vehicles, and dynamic traffic factors like road constructions or other traffic regulations will also have effect on the traveling directions.

Given two vertices $s, e \in V$, the classical shortest path query asks for a path p, which consists of a sequence of edges $p =< e_1, e_2, ..., e_k >$ that connects s and t such that the condition $\sum_{i=1}^{k} w(e_i)$ is minimized. However, in real road networks, only considering the total traveling cost without taking into account of TIFs is not enough. Therefore, we would like to select the path with the minimum *path cost* instead of simply minimizing the scheduled cost. Path cost π for path p is defined as follows:

$$\pi(p) = \alpha(T_s^p) \cdot RTR(p) + \beta(T_s^p) \cdot NTL(p) + \gamma \cdot T_s^p + \theta \cdot \sum_{i=1}^{k} OTF(e_i) \quad (2)$$

where $RTR(p) = \frac{\#\ of\ right\ turns}{\#\ of\ total\ turns}$ represents the *Right Turn Ratio* over the total number of turns in this whole path, and $NTL(p)$ gives the total *Number of Traffic Lights* in this path. T_s^p is the total scheduled travel cost along this path. Note that the speed limit could vary along the whole path, therefore we need a summation of individual traveling time from each edge piece within the path. And finally, if there exist any other traffic influence factors among these edges, we sum them up and add to the equation as $\sum_{i=1}^{k} OTF(e_i)$. In other words, the real cost of a path is jointly determined by $RTR(p)$, $NTL(p)$, T_s^p and $\sum_{i=1}^{k} OTF(e_i)$. As discussed in the motivation section (Section 3), the number of right turns, the number of traffic lights, the total scheduled traveling time and other traffic influence factors such as road site and traffic jams all play very important roles in the real cost of a path. In this paper, we will only consider the first three components and neglect other factors OTF, however, this component presented in the formula gives a potential extension of our model and could be later on integrated easily.

Now let us switch gears to the coefficient functions. $\alpha(T_s^p)$ and $\beta(T_s^p)$ is the coefficient of $RTR(p)$ and $NTL(p)$ respectively. And γ is the factor of T_s^p. These coefficient functions directly reflect the importance of $RTR(p)$, $NTL(p)$ and T_s^p that they performed in the formulation of path cost. Note that $\alpha(T_s^p)$ is not a constant but rather a function of w.r.t. T_s^p. It is easy to understand that the time spent on making turns can be neglected in a long journey while it would be significant in a short trip. Therefore $\alpha(T_s^p)$ has a negative correlation with T_s^p, which is the total scheduled traveling time. One alternative formula of $\alpha(T_s^p)$ could be

$$\alpha(T_s^p) = 1/ln(1 + T_s^p) \quad (3)$$

Similarly, $\beta(T_s^p)$ should also have negative correlation with T_s^p. To be more specific, for instance, we could have

$$\beta(T_s^p) = 1/T_s^p \quad (4)$$

The only difference is the log scale is missing, since we have an assumption that the traffic lights influence is higher than the turns, therefore, without the log scale, $\beta(T_s^p)$ makes this influence more relevant and tighter to T_s^p than $\alpha(T_s^p)$. The exact expression of $\alpha(T_s^p)$, $\beta(T_s^p)$ should be learned and estimated from the real-world traffic data. Here we only give two examples that address their positions in our path cost concept. Finally, γ is a constant which emphasize or deemphasize the importance of T_s^p in the total expression, the real road network data could be fed to our path cost concept in order to learn the approximate sub-optimal expression of $\alpha(T_s^p)$ and $\beta(T_s^p)$. Without considering the $OTF(e_i)$, and if no ambiguities can arise, we sometimes omit the parameter p and simply use α, β, and γ. Therefore we have the following formula of path cost ready for our algorithm:

$$\pi(p) = \alpha \cdot RTR(p) + \beta \cdot NTL(p) + \gamma \cdot T_s^p \quad (5)$$

Note that given a path p, the way we compute path cost is to look behind and then determine the $RTR(p)$

and $NTL(p)$ in equation 5. We will exactly employ this formula to compute the path cost in our algorithm during the following sections.

5.2 Traffic-aware Shortest Path Algorithm

5.2.1 Graph Partitioning

The graph partitioning problem can be defined as follows: Given a graph $G = (V, E, w)$ with same settings in the SPP definition above and $|V| = n$, $|E| = m$. Partitioning G into subgraphs $G_1, G_2, ..., G_p$ means dividing V into a series of disjoint vertices subsets, namely $V_1, V_2, ..., V_p$, such that (i) $V_i \cap V_j = \emptyset, \forall i \neq j$ and (ii) $\cup_{1 \leq i \leq p} V_i = V$. Once the graph is partitioned into a number of subgraphs, the dependencies associated with the edges connecting between different subgraphs would play very important roles of communication. We call these edges as *Cutting Edges*, and the vertices that consist of these edges as *Accessing Nodes*. To sum up, a subgraph is a cluster of nodes and contains two kinds of nodes: *Inner Nodes*, whose all edges lead to nodes within the same subgraph; *Outer Nodes*, whose at least one edge connects to another subgraph. One could see that the accessing nodes are exactly the same with outer nodes. We will use the notation V_j^i and V_j^o, where $1 \leq j \leq p$, to represent inner nodes and outer nodes, and use the notation E_j^i and E_j^o for inner edges and cutting edges, respectively.

Partitioning is not simple. First of all, to take advantage of the graph structure, the cut cannot be random, as it would fluctuate performance. It would be ideal if we could equally cut the whole graph into equivalent pieces. However, in real road networks, the nodes are not distributed uniformly. Therefore, a good partitioning is done subject to several optimal constraints (Šimek and Šimeček, 2011), (Tang and Y. Zhang, 2008):

- An appropriate subgraph size must be chosen, or, an appropriate number p of subgraphs should be determined and self-adjusted.
- Nodes that belong to the same subgraph should be geographically located near each other.
- The number of the nodes contained in G_i is approximately equal, i.e., $|V_i| \approx n/p$;
- Total weights of cutting edges should be minimized, i.e. minimize $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{v_1 \in V_i, v_2 \in V_j} w(\{v_i, v_j\})$.

However, choosing optimal graph partitioning is NP-hard (Geary and Johnson, 1979). This problem, also known as *Minimum K-Cut Problem*, can be reduced to set cover problem and to obtain the exact solutions is computationally intractable. To achieve suboptimal solutions, various heuristics have been proposed. Among these methods, we will employ METIS library (MET,) to achieve our goal. METIS is developed in the Karypis Lab, which is a widely used software package for partitioning graphs based on a multilevel paradigm.

5.2.2 Trafforithm

In this subsection, we develop our parallel traffic-aware shortest path algorithm, *Trafforithm*. It is performed in three phases.

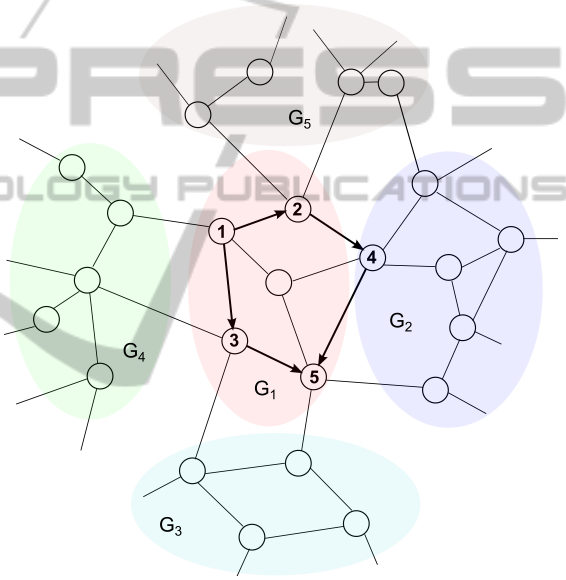

Figure 2: Example of subgraphs.

In the first phase, the graph partitioning described in the previous subsection of the whole given network will be done initially. Right after the partitioning, a temporary graph $G' = (V', E', w)$ is constructed where $V' = \cup_{j=1}^{p} V_j^o$ and $E' = \cup_{j=1}^{p} E_j^r$. In the above formula, E_j^r represents the union of the cutting edges E_j^o and the set of edges connecting any two of the nodes in V_j^o directly. In other words, the temporary graph G' consists of reduced variants of all subgraphs except for the one where the start and end nodes belong, which are in the form of non-reduced subgraphs containing in G_j. Followed by this, for each subgraph G_j, the algorithm performs localized routing process, namely to build a table keeping track of shortest distance between any pair of accessing points (x, y) where $x, y \in V_j^o$. This table is extremely helpful when we apply our algorithm on the whole graph: we

only need to look up in the table to find the cost in constant time when computing the total path cost. In other words, it is like building up "virtual" edges between any pair of the accessing points. And once the this is finished, we do not need to run this partition as well as these pre-computations again for the next request. In other words, the pre-computation is quite affordable since it only consumes time the first time the network is generated or when some significant changes happened in this network. In addition, this step could be distributed into different threads since the computations are all closed within subgraphs.

Next, when given the start point s and end point e, the algorithm determines the indexes i_s and i_e of the subgraphs that s and e belongs to, respectively. In G_{i_s}, the algorithm performs a local routing search to find the shortest path from s to all the accessing points of G_{i_s}. Similar case would happen in G_{i_e}. Together with the temporary graph $G^{'}$, we now are able to build our reduced graph $G^r = (V^r, E^r, w)$ where $V^r = V^{'} \cup V^i_{i_s} \cup V^i_{i_e}$ and $E^r = E^{'} \cup E_{i_s} \cup E_{i_e}$. At this point, the algorithm performs one iteration of modified Dijkstra's algorithm with path cost. Then a temporary shortest path P^r based on the reduced graph G^r is generated, from which we could get a list L^r that consists of indexes of all subgraphs containing edges of the path P^r. This is done as the second phase.

Note that if $i_e == i_s$, which means that the start and end point belong to the same subgraph, we cannot simply populate L^r with this only index. The reason is that this local shortest path may not be guaranteed to entirely become the globally shortest paths considering the other subgraphs. An example can be found in Figure 2, in which the shortest path from node 1 to node 5 is to follow the path $< 1,2,4,5 >$ rather than $< 1,3,5 >$ if $\pi(< 1,2,4,5 >)$ is less than $\pi(< 1,3,5 >)$. Therefore, only considering the subgraph G_1 is not enough to find the global minimum. Whenever this situation happens, the algorithm would do a k-hop breadth first search (BFS) starting from s. In other words, the indexes of subgraphs found within k hops from s will all be added into the list L^r, building up the reduced graph based on these subgraphs together as shown in Figure 2. This k can either from user input or be self-adjusted according to the size of subgraphs.

In the third phase, a pruned graph G^p is constructed from original subgraphs that appear correspondingly in the list L^r. Graph $G^p = (V^p, E^p, w)$ is then a non-reduced graph, and from phase two we have determined through these subgraphs an optimal shortest path can be achieved.

The whole algorithm is presented in Algorithm 1. Graph partitioning is done in line 3. In line 4, the algorithm performs the routing pre-computation on subgraphs. Line 5 and 6 give the indexes of subgraphs that s and e belongs to respectively. If they are identical, line 7 to 11 generate the index list L^r. Otherwise, a reduced graph G^r is constructed from line 13 to 28. $Neighbor(i)$ is a method that returns a list of indexes of subgraphs that are adjacent to the given subgraph of index i and $index(g)$ is a method that

Algorithm 1: Trafforithm.

```
   Input: G = (V,E,w), s, e, (k)
   Output: Shortest path P
 1 L^r ← NULL;
 2 V^r, V^p ← NULL; E^r, E^p ← NULL ;
 3 Graph partitioning using METIS library ;
 4 Parallel localized subgraph routing
   pre-computation ;
 5 Identify the subgraph indexes i_s, i_e for s, e ;
 6 if i_s == i_e then
 7     L^r.push(i_s) ;
 8     for index i found in k − hop BFS do
 9         L^r.push(i) ;
10     end
11 end
12 else
13     for each subgraph g ∈ G do
14         if index(g) == i_s or index(g) == i_e
           then
15             V^r ← V^r ∪ V_index(g) ;
16             E^r ← E^r ∪ E_index(g) ;
17         end
18         else
19             V^r ← V^r ∪ ACC(index(g)) ;
20             for each pair {u,v} where u,v ∈
               ACC(index(g)) do
21                 if {u,v} ∈ E_index(g) then
22                     V^r ← V^r ∪ {u,v} ;
23                 end
24             end
25         end
26     end
27     L^r = runsp(G(V^r, E^r, w), s, e) ;
28 end
29 for each index i ∈ L^r do
30     for each subgraph g ∈ G do
31         if index(g) == i then
32             V^p ← V^p ∪ V_index(g) ;
33             E^p ← E^p ∪ E_index(g) ;
34         end
35     end
36 end
37 P ← ParallelPathComp(G(V^p, E^p, w), s, e) ;
38 return P ;
```

retrieves the index of a given subgraph g. Note that $ACC(index(g))$ gives a set of accessing points of subgraph g, i.e. $V^o_{index(g)}$. Followed by that, the first run is performed in line 27, and $runsp(G,s,e)$ is a function that returns a list of indexes of all subgraphs containing edges of the shortest path from s to e in graph G. The pruned graph based on this list L^r is generated from line 29 to 36 either from case that s and e belongs to the same subgraph or the other way around. $G(V,E,w)$ is a graph constructor given a vertice set V, an edge set E and a weight function w. And finally, a parallel path computing is performed on this pruned graph G^p and the real shortest path P is returned in line 37 and 38.

Algorithm 2: Parallel Path Computation.

Input: Pruned graph $G^p = (V^p, E^p, w), s, e$
Output: Shortest path P

```
1  S ← NULL, P ← NULL ;
2  Q1, Q2 ← NULL, P1, P2 ← NULL ;
3  mark all nodes (except for s) unvisited ;
4  S ← S ∪ {s} ;
5  select node n with argmin_n{π(< s,n >)} ;
6  Q1.insert(n) ;
7  >> For each of the two threads:
8  while true do
9  |  {u,v}({v,u}) ← Q1.removeMin() ;
10 |  if v(u) has been visited then
11 |  |  Break ;
12 |  end
13 |  P1.add({u,v}) ;
14 |  mark {u,v} as visited ;
15 |  if v ∉ S then
16 |  |  S ← S ∪ {v} ;
17 |  |  select node n with argmin_n{π(P1 ≻ n)}
   |  |  Q1.insert(n) ;
18 |  end
19 |  select n with argmin_n{π(P1 ≺ v ≻ n)} ;
20 |  Q1.insert(n) ;
21 end
```

The parallel bidirectional path computing process is presented in Algorithm 2. It fist marks all nodes unvisited except for s and select the node n with the least path cost when forming a new path $< s,n >$ in line 3 to 6. In the first thread, each time we extract the node leading to the minimum path cost from the queue Q_1 and mark it as visited and examine all its neighbors' path cost during line 8 to 19. At the end we also insert the minimum path cost leaving from the starting node of the newly inserted edge in line 20 and 21. Symbol $\succ$ is used to extend the current path (on the left) with the path right to this symbol while $\prec$ means to remove the sub-path on the right from the path (on the left). Hence $P \prec v \succ n$ means to remove v from the end of P and add n to the end instead in line 20. In the second thread, we merely take similar moves, and we use variables in round brackets to distinguish from the first thread in the algorithm like in line 9 and 10. Note that the graph is directed, each time we examine from the end to the start, before adding to the queue, we have to switch to the normal moving direction. The algorithm will stop if one direction touches an edge that has already been visited by the other direction. In other words, we break the loop like in line 11 when we touch a certain node that has been marked as visited previously. In general, this algorithm adds nodes to S in order of increasing distance from start node s and therefore we could incrementally get our shortest path P from s to e eventually.

6 EVALUATION

In this section, we will present some ideas and different phases on how to evaluate our proposed algorithm. Firstly, our method relies on the choice of proper values for the parameters $\alpha(T_s^p)$, $\beta(T_s^p)$, and γ. In a first set of experiments, we would analyze how to learn appropriate values for these coefficients. With respect to these experiments we report about results of our algorithm related to the final outcome of its last iteration. To achieve this, the real road network data could be fed to our path cost concept in order to learn the approximate sub-optimal expression of path cost. This learning process is unsupervised, with convergence on $\alpha(T_s^p)$, $\beta(T_s^p)$, and γ.

After that, in the second phase, by considering traffic influence factors, we will compare the shortest path result from our algorithm with some of the prevailing algorithms including ALT (Goldberg and Harrelson, 2005), Arc Flags (Hilger et al., 2006), and Transit Node Routing (Bast et al., 2007), (Bast et al., 2006) as well as the classical Dijkstra's algorithm (Dijkstra, 1959) and A* search (Hart et al., 1968). The testing environment is Windows 7 with Intel i7-4770 3.4 GHz CPU and 16 GB RAM with stable working state. The quad cores will fully utilize the parallelism design of our algorithm.

The efficiency and effectiveness of a shortest path algorithm should be evaluated based on (but not limited to) the following criteria:

- Pre-processing time. Quite a number of prevailing algorithms perform a pre-processing phase. It is reasonable to include the pre-processing time into comparison.
- Average response time. The average response time to shortest path queries will be the most im-

portant factor that judges the effectiveness and quality of the algorithms.

- Recovery interval. Considering the continuous changes of the network, like traffic jams and road constructions, where certain paths are blocked, the ability (and the time it takes) of recovering from such changes and returns to the state of query-ready is another important measure of the merits of the algorithms.

7 CONCLUSIONS AND FUTURE WORK

In this article, we point out the importance of traffic influence factors in route planning in a road network environment. Based on our proposed generic two-layered model, an efficient multi-thread shortest path algorithm with the consideration of traffic influence factors is presented. With this framework, we could demonstrate the value of our algorithm and how it leads to fast, accurate and practical search results.

In addition, we have presented some ideas on how to evaluate our algorithm performance. Future work would include the implementation of our algorithm and empirical experiments.

REFERENCES

Metis, www.cs.umn.edu/metis.

Awasthi, A., Lechevallier, Y., Parent, M., and Proth, J. M. (2005). Rule based prediction of fastest paths on urban networks. In *8th Intl. IEEE Conference on Intelligent Transportation Systems*.

Barrat, A., Barthélemy, M., Pastor-Satorras, R., and Vespignani, A. (2004). The architecture of complex weighted networks. In *Proc Natl Acad Sci*, volume 101, pages 3747–3752.

Bast, H., Funke, S., and Matijevic, D. (2006). Transit: ultrafast shortest-path queries with linear-time preprocessing. In *Proc. of the 9th DIMACS Implementation Challenge*, pages 175–192.

Bast, H., Funke, S., Sanders, P., and Schultes, D. (2007). Fast routing in road networks with transit nodes. *Science*, 316(5824):566.

Dantzig, G. B. (1962). *Linear Programming and Extensions*. Princeton University Press.

Demiryurek, U., Banaei-Kashani, F., and Shahabi, C. (2010). A case for time-dependent shortest path computation in spatial networks. In *Proceedings of the 18th SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, pages 474–477.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Freeman, L. (1978). Centrality in social networks: Conceptual clarification. In *Social Networks*, volume 1, pages 215–239.

Geary, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of Incompleteness*. W.H. Freeman and Company.

Goldberg, A. V. and Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory. In *SODA*, pages 156–165.

Google. Google maps, www.maps.google.com.

Hart, P. E., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, volume 4, pages 100–107.

Hilger, M., E. Köhler, R. H. M., and Schilling, H. (2006). Fast point-to-point shortest path computations with arc-flags. In *Proc. of the 9th DIMACS Implementation Challenge*, pages 73–92.

Jiang, B. (2007). A topological pattern of urban street networks: universality and peculiarity. *Physica A: Stat. Mechanics and its Applications*, 384(2):647–655.

Jiang, B. and Liu, X. (2011). Computing the fewest-turn map directions based on the connectivity of natural roads. *International Journal of Geographical Information Science*, 25(7):1069–1082.

Kanjilal, V. and Schneider, M. (2011). Modeling and querying spatial networks in databases. *JMPT*.

Kanoulas, E., Y. Du, T. X., and Zhang, D. (2006). Finding fastest paths on a road network with speed patterns. In *22nd International Conference on Data Engineering*.

Lu, E. H.-C., Huang, C. W., and Tseng, V. S. Continuous fastest path planning in road networks by mining real-time traffic event information.

Lu, E. H.-C., Lin, C.-C., and Tseng, V. S. (2008). Mining the shortest path within a travel time constraint in road network environments. In *11th Intl. IEEE Conference on Intelligent Transportation Systems*.

Navteq. Navteq, www.maps.navteq.com.

Qi, L. and Schneider, M. (2012). Monet: Modeling and querying moving objects in spatial networks. In *3rd ACM SIGSPATIAL Int. Workshop on GeoStreaming*.

Qi, L. and Schneider, M. (2014). Realtime response of shortest path computation. In *7th ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWCTS)*, volume 4.

Šimek, F. and Šimeček, I. (2011). Improvement of shortest path algorithms through graph partitioning. In *Proceedings of Intl. Conf. Presentation of Mathematics'11*, pages 131–137, Liberec. Technical University.

Tang, Y. and Y. Zhang, H. C. (2008). A parallel shortest path algorithm based on graph-partitioning and iterative correcting. In *10th IEEE International Conference on High Performance Computing and Communications*, pages 155–161.

TOMTOM. Tomtom iq-routes, www.tomtom.com/page/iq-routes.