International Journal of Management Science
Vol 12, No 2, November 2006

On Finding a Convenient Path in the Hierarchical Road Network*

Kiseok Sung
Dept. of Industrial and Systems Engineering,
Kangnung National University, Gangneug, Korea

Chan-Kyoo Park**
Dept. of Management, Dongguk University, Seoul, Korea

Sangwook Lee
Samsung SDS Co., Seoul, Korea

Seungyong Doh
IE & Software Group, Samsung Electronics Inc., Suwon, Korea

Soondal Park
Dept. of Industrial Engineering, Seoul National University, Seoul, Korea

(Received Apr. 2005; Revised Aug. 2006; Accepted Nov. 2006)

ABSTRACT

In a hierarchical road network, all roads can be classified according to their attributes such as speed limit, number of lanes, etc. By splitting the whole road network into the subnetworks of the high-level and low-level roads, we can reduce the size of the network to be calculated at once, and find a path in the way that drivers usually adopt when searching out a travel route. To exploit the hierarchical property of road networks, we define a convenient path and propose an algorithm for finding convenient paths. We introduce a parameter indicating the driver's tolerance to the difference between the length of a convenient path and that of a shortest convenient path. From this parameter, we can determine how far we have to search for the entering and exiting gateway. We also propose some techniques for reducing the number of pairs of entries and exits to be searched in a road network. A result of the computational experiment on a real road network is given to show the efficiency of the proposed algorithm.

Keywords: Route Planning, Convenient Path, Shortest Path, Hierarchical Road Network

---

* This work was financially supported by overseas scholarship of LG-YeonAm Foundation and Kangnung National University during the year 2004.
** Corresponding author, Email: parkck@dongguk.edu

1. INTRODUCTION

In route planning and traffic assignment, the iterative calculation of shortest paths is needed as a basic component. Finding shortest paths in a large-scale road network will require much computational time and memory space. The approach of using shortest paths in the route planning is based on the assumption that drivers prefer a shortest route. In many real situations, however, drivers tend to choose a route that is not only fairly short but also convenient to travel through, although the path might be not a shortest one.

In some previous researches, the properties of the hierarchical structure of road networks have been utilized. A hierarchical road network consists of a few grades of roads ranging from dense and low-speed local roads to sparse and high-speed expressways. When drivers find a route in the hierarchical road network, they first search the subnetwork of low-grade local roads in order to reach a gateway into the subnetwork of high-grade roads, then they go through high-grade roads to a gateway from which the destination can be reached through low-grade roads. This hierarchical approach has two major advantages. One is that it can reduce the computational effort for finding short routes because the whole road network is partitioned into small subnetworks in which routes can be found more easily. The other advantage is that the hierarchical property enables us to find more human-oriented routes. A shortest path of the whole network may cause a driver to drive in and out of local roads and high-speed expressways repeatedly [5].

Shapiro, Waxman, and Nir [7] first discussed about the practical situations when the additional information about the types of roads is available, and introduced a level graph that is based on the level of nodes and edges. They provided some algorithms for finding a path in the level graphs under the assumption that it is desirable to spend as little time as possible in low-grade roads. Similarly, Campbell [2] used continuous space models to analyze several urban travel decisions in which a driver must select between a shortest route on relatively low-speed roadways and a longer route which includes travel on high-speed roadways. Chou, Romeijn, and Smith [3] proposed a hierarchical algorithm for approximating a shortest path between all pairs of nodes in a large-scale network. As a rule for selecting entering and exiting gateways on the high-level subnetwork, they considered two strategies. One is to choose the gateway nearest to the origin and the gateway nearest to the destination as the entering gateway and the exiting gateway, respectively. The other is to choose the pair of gateways that approximate a shortest path by enumerating all pairs of entering and exiting gateways on the high-level subnetwork. Recently, Liu [5] proposed an approach of exploiting the knowledge of road types and partitioning the whole network into many small subnetworks to reduce the time and memory space required for approximating a shortest path. Also, Seong, Sung and Park [6] proposed an algorithm for finding a shortest path of a grid-type urban road network by making use of the hierarchical structure of road networks. Recently, Duckham and Kulik [4] proposed a "simplest" path, in terms of the instruction complexity, for automated route selection for navigation. The "simplest" paths are concerned with only the number of turns at the intersections, rather than the length of the paths.

In this paper, to exploit the hierarchical structure of road networks, we define convenient paths. Most existing researches focused only on finding one convenient path which goes through the entering and the exiting gateway nearest to the origin and the destination, respectively. This approach leads to a convenient path whose length is far greater than that of a shortest convenient one. On the other hand, the existing approach for finding a shortest convenient path considers all pairs of the entering and exiting gateways, and consequently requires much computational time. The main purpose of this paper is to propose a new algorithm for finding a nearly shortest convenient path with a little computation. For this, we introduce a parameter indicating the driver's tolerance of the difference between the length of a convenient path and that of a shortest convenient path. The parameter will limit the range within which we need to search nodes for entrances and exits to find a convenient path acceptable to the drivers. In addition, some techniques are proposed to reduce the number of pairs of entries and exits to be searched in a road network. The second purpose is to discuss whether convenient paths can approximate a shortest path in the whole network. Although the hierarchical structure of road networks has been exploited in previous studies, there have been no studies which deal with the gap between the length of convenient paths and that of a shortest path in the whole network. We derive the relationship between the length of convenient paths and that of a shortest convenient path, and between the length of a shortest convenient path and that of a shortest path. Experimental results show that a shortest convenient path can be a good approximation to a shortest path.

This paper is organized as follows: In section 2 a convenient path is defined, and in section 3 its properties are discussed. In section 4, we propose some techniques for enumerating convenient paths efficiently. In section 5, a new algorithm for finding a shortest convenient path in the hierarchical network is proposed, and the experimental results of the proposed algorithm using a real road network are presented in section 6. Finally, some conclusions of this work are given in section 7.

2. THE DEFINITION OF A CONVENIENT PATH

Consider a directed road network $G = (N, A)$ with N and A being the set of nodes and the set of arcs, respectively. The travel time or the length of an arc (i, j) is denoted by c_{ij}. The length and the travel time of arcs will be used interchangeably. We assume that $c_{ij} > 0$ for every arc $(i, j) \in A$. For each arc $(i, j) \in A$, the nodes i and j are called the *head node* and the *tail node* of arc (i, j). A sequence of distinct nodes, $p = (i = i_1, i_2, \ldots, j = i_r)$ with $r \geq 2$, is called a *path* from i to j or an *i-j path* if it satisfies the condition that $(i_k, i_{k+1}) \in A$ for each $1 \leq k \leq r-1$. The expression $(i, j) \in p$ means that arc (i, j) is included in path p. Let $d(p)$ denote the length of a path p, that is, the sum of the length of the arcs included in p. Let $\mathbf{P}_{ij}$ denote the set of all paths from i to j. Then, the *shortest path problem* is to find the path $p^* \in \mathbf{P}_{st}$ such that $d(p^*) \leq d(p)$, $\forall p \in \mathbf{P}_{st}$ where s and t are the origin and destination, respectively. Note that if c_{ij} denotes the travel time of arc (i, j), the path p^* represents the minimum traveling-time path. Throughout this paper, it is assumed that there is at least one path from s to t. Also, if there is no explicit description, the origin and the destination of any path are assumed to be s and t, respectively.

All roads of a road network can be classified into two groups according to their width or speed limit. One group consists of roads with higher speed limit and many lanes; for example, expressways and major roads. The other consists of roads with lower speed limit and a few lanes; for example, local roads and minor roads. The roads of the first group and the second group form the high-level subnetwork and low-level subnetwork, which will be denoted by G_H and G_L respectively. Then, G can be divided into two subnetworks $G_H = (N_H, A_H)$ and $G_L = (N_L, A_L)$ where, $N_H \cup N_L = N$, $A_H \cup A_L = A$ and $A_H \cap A_L = \varnothing$. Note that N_H and N_L have common nodes, but no arc (i, j) can be an element of both A_H and A_L. Without a great loss of generality, it is assumed that both G and G_H are connected. If G is not connected, finding a path in G is the same as finding a path in each connected component of G. The assumption of the connectedness of G_H may not hold in some real road networks. Even in that case, G_H can be made connected by moving some arcs of G_L into G_H if G satisfies connectedness.

Let p denote an $i-j$ path where $p=(i=i_1, i_2, \cdots, j=i_r)$. If $(i_{u-1}, i_u) \in G_L$ and $(i_u, i_{u+1}) \in G_H$ for some $2 \le u \le r-1$, node i_u is called an ***entry*** of p. Also, if the first arc (i_1, i_2) of p is on the high-level subnetwork $i(=i_1)$, is called an entry of p. Similarly, if $(i_{v-1}, i_v) \in G_H$ and $(i_v, i_{v+1}) \in G_L$ for some $2 \le v \le r-1$, node i_v is called an ***exit*** of p. If the last arc (i_{r-1}, i_r) of p is on the low-level subnetwork $j(=i_r)$, is also called an exit of p. A convenient path is defined as follows:

Definition 1 (Convenient path). *A path $p=(i=i_1, i_2, \cdots, j=i_r)$ is called a convenient path if p has at most one entry and one exit.*

For example, in Figure 1 paths (a), (b), (c), (d), and (e) are convenient paths, but path (f) and (g) are not a convenient path because it has two entries and two exits. Note that path (a) has neither an entry nor an exit. Path (a) goes through only the low-level subnetwork, while (b) goes through only the high-level subnetwork.

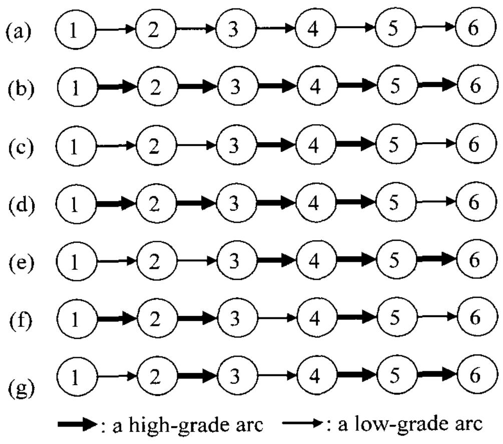

Figure 1. Examples for convenient and non-convenient paths

The definition of a convenient path is motivated by the manner in which drivers think when they make a route choice. When drivers plan a route in the hierarchical road network, they search an entering gateway on the high-level subnetwork and then go through high-grade roads to an exiting gateway from which the destination can be reached through low-grade roads. In general, drivers do not prefer to paths which enter into and get out of the high-level subnetwork repeatedly because the paths are not *convenient* for drivers to travel through.

If a convenient path is not longer than that of any other convenient path, it is called a ***shortest convenient path***. Note that a shortest convenient path might not be a shortest one. For a convenient path p that has one entry and one exit, the subpath of p between the entry and the exit, is called a ***major segment*** of p. A convenient path has at most one major segment. Similarly, the subpaths of p which are formed after removing all arcs in the major segment are called ***minor segments*** of p. A convenient path has at most two minor segments. Let $\alpha(p)$ denote the length of the major segment of p, and $\beta(p)$ denote the sum of the length of the minor segments of p. Note that $d(p) = \alpha(p) + \beta(p)$. Moreover, if a convenient path p has only minor segments, then $d(p) = \beta(p)$ and $\alpha(p) = 0$. Similarly, if p consists of only one major segment, then $d(p) = \alpha(p)$ and $\beta(p) = 0$. A minimal convenient path is defined as follows:

Definition 2 (Minimal convenient path). *A convenient path is called a minimal convenient path if its minor segments are shortest paths in the low-level subnetwork and its major segment is a shortest path in the high-level subnetwork.*

For example, consider an undirected network in Figure 2. A convenient path $1 \to 2 \to 5 \to 8 \to 9 \to 12$ is not a minimal convenient path because its minor segment $1 \to 2 \to 5$ is not a shortest path in the low-level subnetwork. Path $1 \to 3 \to 5 \to 8 \to 9 \to 12$ is a minimal convenient and shortest convenient path from 1 to 12. In fact, any shortest convenient path is a minimal convenient one. Another path $1 \to 3 \to 4 \to 6 \to 7 \to 11 \to 12$ is a minimal convenient path which consists of only one minor segment. Since any shortest convenient path is a minimal convenient one, we are concerned with only minimal convenient paths to find a shortest convenient path.

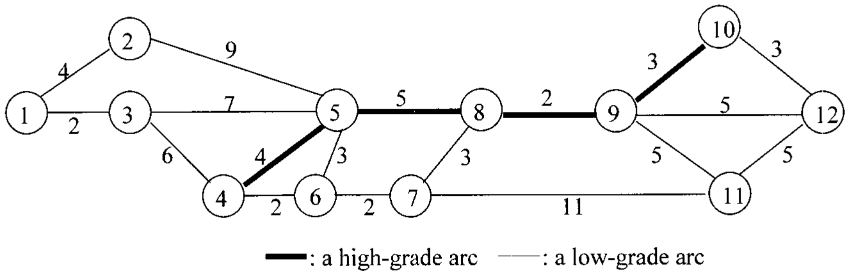

Figure 2. Minimal convenient paths in a road network

There have been two methods for finding convenient paths: One method is to find a convenient path under the assumption that it is desirable to spend as little time as possible in the low-level subnetwork [7, 3]. This approach for finding a convenient path will be called the *nearest entry-exit algorithm* because it finds the entry and exit of the convenient path which are nearest to the origin and the destination, respectively. The convenient path found by the nearest entry-exit algorithm might be much longer than a shortest convenient path. For example, consider convenient paths from 6 to 12 in Figure 2. The convenient path obtained by the nearest entry-exit algorithm is $6 \to 4 \to 5 \to 8 \to 9 \to 12$, and its length is 19. However, the shortest convenient path from 6 to 12 is $6 \to 7 \to 8 \to 9 \to 12$, whose length is 12.

Another method for finding a convenient path is to consider all pairs of entries and exits, which was used by Chou, Romeijn, and Smith [3]. This approach, which will be called the *all pairs algorithm*, can find a shortest convenient path, but requires much computational efforts.

3. SOME PROPERTIES OF CONVENIENT PATHS

To find a short convenient path with moderate computational efforts, we need to reduce the number of pairs of entries and exits to be searched. As we consider farther entries and exits from the origin and the destination, we can obtain shorter convenient paths. This property of convenient paths is formalized in Property 1. Throughout the paper, $\mathbf{P}_{st}$ denote the set of all minimal convenient paths from s to t and let $\mathbf{P}_{st}(T)$ denote $\{p \in \mathbf{P}_{st} \mid \beta(p) \le T\}$ for a positive constant T.

Property 1. *Let $p^* \in \mathbf{P}_{st}$ be a shortest convenient path. Let $q^* \in \mathbf{P}_{st}(T)$ be a convenient path such that $d(q^*) \le d(q)$, $\forall q \in \mathbf{P}_{st}(T)$. For a constant $0 < T \le d(p^*)$, the following inequalities hold:*

$$\frac{d(q^*) - d(p^*)}{d(p^*)} \le \frac{d(q^*)}{T} - 1 \le \frac{\alpha(q^*)}{T}. \tag{1}$$

Proof.

$$\frac{d(q^*) - d(p^*)}{d(p^*)} \le \frac{d(q^*)}{T} - 1 = \frac{\alpha(q^*) + \beta(q^*) - T}{T} \le \frac{\alpha(q^*)}{T}. \quad \square \tag{2}$$

Although Property 1 holds for any convenient path $q \in \mathbf{P}_{st}(T)$, the first part of inequality (1) becomes the tightest when q^* is applied to inequality (1). The high value of T in Property 1 means that far entries and exits are considered, and then the relative difference between the length of q^* and the length of a shortest convenient path will be reduced. The Property 1 seems to be rather trivial, but experimental results in section 6 will show that it is useful to reduce the number of pairs of entries and exits to be considered for finding a short convenient path.

As mentioned in the previous sections, one of the typical paths considered in the hierarchical road networks is the convenient path which goes through the nearest entry and exit from the origin and the destination. The following property shows that the path might be as long as twice the length of a shortest convenient path, even when the sum of the length of its minor segments is not less than the length of its major segment:

Property 2. *Let $p^* \in \mathbf{P}_{st}$ be a shortest convenient path. Let $\bar{p} \in \mathbf{P}_{st}$ be a minimal convenient path such that*

$$\beta(\bar{p}) = \min_{p \in \mathbf{P}_{st}} \beta(p)$$

Then,

$$\frac{d(\bar{p}) - d(p^*)}{d(p^*)} \le \frac{\alpha(\bar{p})}{\beta(\bar{p})}. \tag{3}$$

In addition, if $\beta(\bar{p}) \ge \alpha(\bar{p})$, then

$$d(\bar{p}) \le 2d(p^*). \tag{4}$$

Proof. Setting $T = \beta(\bar{p})$ in Property 1, we obtain that

$$\frac{d(\bar{p}) - d(p^*)}{d(p^*)} \le \frac{\alpha(\bar{p})}{\beta(\bar{p})}.$$

In addition, if $\beta(\bar{p}) \ge \alpha(\bar{p})$ then the right-hand side of inequality (3) is less than or equal to 1. Therefore, after some arithmetic calculations we find that $d(\bar{p}) \le 2d(p^*)$. □

Note that if $\beta(\bar{p}) < \alpha(\bar{p})$ the right-hand side of inequality (3) is greater than 1. In that case, the convenient path produced by the nearest entry-exit algorithm might be longer than two times the length of a shortest convenient path.

Next, we discuss to what extent a shortest convenient path can approximate a shortest path in the whole road network. The following lemma gives an upper bound to the relative difference between the length of a shortest convenient path and that of a shortest path.

Lemma 1. *For any two nodes* $u \in N_H$ *and* $v \in N_H (u \neq v)$, *let* g^*_{uv} *and* r^*_{uv} *denote a shortest path of G and a shortest path of* G_H *from u to v, respectively. Also, let* g^*_{st} *and* p^*_{st} *denote a shortest path and a shortest convenient path of G from s to t, respectively. If the following inequality holds for a nonnegative constant* θ,

$$\frac{d(r^*_{uv}) - d(g^*_{uv})}{d(g^*_{uv})} \leq \theta, \qquad \forall u \in N_H, \forall v \in N_H (u \neq v), \tag{5}$$

then

$$\frac{d(p^*_{st}) - d(g^*_{st})}{d(g^*_{st})} \leq \theta. \tag{6}$$

Proof. If g^*_{st} is a convenient path, then $d(g^*_{st}) = d(p^*_{st})$, which implies that the lemma is trivially established. If g^*_{st} is not a convenient path, then it has at least two major segments. Let (i, j) be the last arc of the first major segment of g^*_{st} and (k, l) be the first arc of the last major segment of g^*_{st}. Let r^*_{jk} be a *j*-*k* shortest path of G_H. Then, we obtain a convenient path q_{st} of *G* by replacing the *j*-*k* subpath g^*_{jk} of g^*_{st} with r^*_{jk}. Hence, we get

$$\begin{aligned} d(p^*_{st}) \leq d(q_{st}) &= d(g^*_{st}) + d(r^*_{jk}) - d(g^*_{jk}) \\ &\leq d(g^*_{st}) + \theta d(g^*_{jk}) \\ &\leq (1+\theta) d(g^*_{st}). \quad \square \end{aligned}$$

In inequality (5), if the origin is rather far away from the destination, the paths going through the high-level subnetwork are usually shorter than the paths going through the low-level subnetwork. Therefore, we can expect that the numerator, $d(r^*_{uv}) - d(g^*_{uv})$, is not large in well-designed real road networks, and a shortest convenient path will have almost the same length with a shortest path of the whole network. In such a sense, θ can be interpreted as a constant indicating how well a real road network conforms to the hierarchical property. As an extreme case, suppose that for any $u \in N_H$ and $u \in N_H$, there exists a shortest *u*-*v* path which consists of only high-grade arcs. In this case, $d(r^*_{uv}) - d(g^*_{uv})$ will be zero for any $u \in N_H$ and $v \in N_H$, and a shortest convenient s-t path becomes a shortest one in G.

Finally, by integrating Property 1 and Lemma 1, we come to a conclusion that convenient paths will be a good approximation of shortest path if the road network is well-designed and entries and exits within a moderate distance are searched. The relationship between the length of a convenient path and that of a shortest path can be established as follows:

Theorem 1. *Let g^* denote a shortest path of G from s to t, and $p^* \in \mathbf{P}_{st}$ be a shortest convenient path. For a constant $0 < T \leq d(p^*)$, let $q^* \in \mathbf{P}_{st}(T)$ be a convenient path such that $d(q^*) \leq d(q)$, $\forall q \in \mathbf{P}_{st}(T)$. If inequality (5) holds, then*

$$\frac{d(q^*) - d(g^*)}{d(g^*)} \leq (1+\theta)\frac{\alpha(q^*)}{T} + \theta.$$

Proof. By Property 1 and Lemma 1, we easily obtain that

$$\begin{aligned}\frac{d(q^*) - d(g^*)}{d(g^*)} &= \frac{d(p^*)}{d(g^*)}\frac{d(q^*) - d(p^*)}{d(p^*)} + \frac{d(p^*) - d(g^*)}{d(g^*)} \\ &\leq (1+\theta)\frac{\alpha(q^*)}{T} + \theta. \quad \square\end{aligned}$$

4. ENUMERATING CONVENIENT PATHS EFFICIENTLY

In this section, we are concerned with how to terminate early searching for entries and exits without aggravating the quality of the final convenient path. The first way to reduce searching efforts is to introduce a parameter δ which means the driver's tolerance to the relative difference between the length of a shortest convenient path and that of the path given by an algorithm. That is, the relative difference between the length of the convenient path provided by an algorithm and the length of a shortest convenient path is allowed to be at most δ-times longer than a shortest convenient path. That constraint can be expressed into

$$\frac{d(q^*) - d(p^*)}{d(p^*)} \leq \delta, \tag{7}$$

where $p^* \in \mathbf{P}_{st}$ denotes a shortest convenient path and q^* denotes a shortest one among all paths of $\mathbf{P}_{st}(T)$. The range of T satisfying inequality (7) is derived by Property 1 as follows:

$$\frac{d(q^*)-d(p^*)}{d(p^*)} \le \frac{d(q^*)}{T} - 1 \le \delta \quad \Rightarrow \quad T \ge \frac{d(q^*)}{1+\delta}. \tag{8}$$

In other words, if we only consider the entries and exits within the distance of $d(q^*)/(1+\delta)$, we can find a convenient path which is at most $(1+\delta)$-times longer than a shortest convenient path. The parameter δ prevents an algorithm from searching entries and exits which are too far way from the origin and destination.

The introduction of δ has two advantages. Using δ provides a simple way in which drivers can control the quality of the final path by compromising with the computational efficiency. The value of δ doesn't have to be affected by the location of the origin and the destination. However, the value of T need to be adjusted according to the distance from the origin (destination) to entrances (exits), which will be not easy for every driver. In the proposed algorithm, T will be automatically derived from δ using inequality (8). Another advantage is that the introduction of δ makes a significant contribution to bounding unnecessary entrances and exits by setting T to a tighter value, which will be shown empirically by experimental results in section 6.

Another way for reducing the number of pairs of entries and exits to be searched is to make use of the information which is stored in intelligent transportation systems (ITS) or geographic information systems (GIS). Usually ITS or GIS includes the data about the longitude/latitude or the coordinates of each node, from which the Euclidean distance between any two nodes is calculated easily. If the weight c_{ij} associated with each arc (i, j) represents the distance of arc (i, j), the Euclidean distance between the origin and the destination can be directly used as a lower bound to the length of a shortest convenient path. However, if c_{ij} denotes the traveling time of arc (i, j), some transformations are required to obtain a lower bound from the Euclidean distance as follows:

Property 3. *Given $T > 0$, let q^* denote a convenient path such that $d(q^*) \le d(q)$, $\forall q \in \mathbf{P}_{st}(T)$. Let p^* be a shortest convenient path and let l be the Euclidean distance between the origin and the destination. Let v_H and v_L denote the speed limit of high-grade arcs and low-grade arcs, respectively. Suppose that $v_H \ge v_L$. If ρ is set to*

$$\rho = \frac{l}{v_H} + \beta(q^*)(1 - \frac{v_L}{v_H}), \tag{9}$$

then ρ is a lower bound on $d(p^*)$.

Proof. Note that $\alpha(p^*) + \beta(p^*) \le \alpha(q^*) + \beta(q^*)$.

(i) In case that $\beta(p^*) \le \beta(q^*)$, we know that $d(p^*) = d(q^*)$ by definition. Also, we know that $l \le \alpha(q^*)v_H + \beta(q^*)v_L$. Hence,

$$\rho = \frac{l}{v_H} + \beta(q^*)(1 - \frac{v_L}{v_H}) = \frac{l - \beta(q^*)v_L}{v_H} + \beta(q^*) \le \alpha(q^*) + \beta(q^*).$$

Therefore, ρ is a lower bound to $d(p^*)$.

(ii) In case that $\beta(p^*) \ge \beta(q^*)$, we know that $l \le \alpha(p^*)v_H + \beta(p^*)v_L$. Therefore,

$$\rho = \frac{l}{v_H} + \beta(q^*)(1 - \frac{v_L}{v_H}) \le \frac{l}{v_H} + \beta(q^*)(1 - \frac{v_L}{v_H}) \le \alpha(p^*) + \beta(p^*). \quad \square$$

Note that ρ depends on q^*, which is the shortest convenient path among the paths in $\mathbf{P}_{st}(T)$. This means that if T changes during the search of entries/exits and a shorter path is found, then a lower bound ρ can be improved to have a greater value. The proposed algorithm will update ρ whenever it finds a new convenient path which is shorter than the paths ever found.

Using such a lower bound to the length of a shortest convenient path, another termination condition can be derived, which will help us terminate the searching algorithm early. Consider inequality (7) again. Replacing $d(p^*)$ with its lower bound ρ, we get the following inequality:

$$\frac{d(q^*) - d(p^*)}{d(p^*)} \le \frac{d(q^*)}{\rho} - 1. \tag{10}$$

Given δ, if the right-hand side of inequality (10) is less than or equal to δ, then inequality (7) is obviously satisfied. Hence, if q^* satisfies the following inequality, we have already found a convenient path whose length is not greater than (1+δ)-times the length of a shortest convenient path:

$$d(q^*) \le \rho(1 + \delta). \tag{11}$$

Finally, we present another property of convenient paths, which will be used as a termination criterion of the proposed algorithm. For a convenient path $q \in \mathbf{P}_{st}$ which has an entry and exit, let $\beta_s(q)$ denote the length of the subpath of q from s to the entry, and $\beta_t(q)$ denote the length of the subpath of q from the exit to t. Note that $\beta(q) = \beta_s(q) + \beta_t(q)$.

Property 4. *Let $p^* \in \mathbf{P}_{st}$ be a shortest convenient path. Suppose that there exists a minimal convenient path $\hat{p} \in \mathbf{P}_{st}$ consisting of only low-grade arcs. Let x be one of the nodes passed by $\hat{p}$. Let T_s and T_t denote the length of the subpaths of $\hat{p}$ from s to x and from x to t, respectively. Then,*

$$d(p^*) = \min\{d(\hat{p}), \min_{q \in \mathbf{P}_{st}}\{d(q) \mid \beta_s(q) \le T_s \text{ or } \beta_t(q) \le T_t\}\}.$$

Proof. Since any convenient path with no major segment is not shorter than p, we are only concerned with convenient paths which have a major segment. Let $p \in \mathbf{P}_{st}$ be a convenient path which has a major segment. Suppose that $\beta_s(q) > T_s$ and $\beta_t(p) > T_t$. Then, $d(p) > T_s + T_t = d(\hat{p}) \ge d(p^*)$. □

Property 4 can be explained as follows: To find a shortest convenient path, nodes in the low-level subnetwork are searched one by one in order of closeness to the origin and the destination. After searching farther nodes from the origin and the destination, we finally come to find a node where the path from the origin and the path from the destination are connected into a s-t convenient path which consists of only low-grade arcs. Then, we can stop searching any more because the shortest one of the minimal convenient paths found until that time is a shortest convenient path.

5. AN ALGORITHM FOR FINDING A CONVENIENT PATH

In this section, we propose an algorithm for finding a convenient path as presented in Figure 3. For the sake of simplicity, the proposed algorithm CONVPATH assumes that both the origin and the destination belong to the low-level subnetwork, not to the high-level subnetwork. In case that the origin or the destination is an element of the high-level subnetwork, some simple modifications are required so that the origin (the destination) itself is considered as an entry(an exit). Given a constant $\delta > 0$, the proposed algorithm CONVPATH finds a convenient path which is at most (1+δ)-times longer than a shortest convenient path. If δ is set to 0, CONVPATH produces a shortest convenient path. Otherwise, it may approximate a shortest convenient path.

```
CONVPATH: Finding a convenient path
1   $R^s \leftarrow \emptyset$ and $R^t \leftarrow \emptyset$.
2   $E \leftarrow \emptyset$ and $X \leftarrow \emptyset$
3   Set $\beta_s$ to the length of a shortest path from the origin to the nearest entry.
4   Set $\beta_t$ to the length of a shortest path from the nearest exit to the destination.
5   While
6      Find the next nearest node $u$ from the origin in the low-level subnetwork.
7      Set $T_s \leftarrow d(s \rightarrow u) + \beta_t$ and $R^s \leftarrow R^s \cup \{u\}$.
8      If $u$ can be an entry, then
9         find the shortest convenient path $q$ such that its entry is $u$ and its exit is in $X$.
10        If $d(q) < d(q^*)$,, set $q^*$ to $q$ and update a lower bound $\rho$.
11        $E \leftarrow E \cup \{u\}$.
12     End If
13     Call CHK_STOP_COND.
14     Find the next nearest node $v$ to the destination in the low-level subnetwork
15     Set $T_t \leftarrow d(v \rightarrow t) + \beta_s$ and $R^t \leftarrow R^t \cup \{v\}$.
16     If $v$ can be an exit, then
17        find the shortest convenient path $q$ such that its entry is in $E$ and its exit is $v$.
18        If $d(q) < d(q^*)$, set $q^*$ to $q$ and update a lower bound $\rho$.
19        $X \leftarrow X \cup \{v\}$..
20     End If
21     Call CHK_STOP_COND.
22  End While

CHK_STOP_COND: Checking if one of the termination criteria is satisfied
1   If $T = \min\{T_s, T_t\} > d(q^*)/(1+\delta)$, stop
2   If $d(q^*) \le \rho(1+\delta)$, stop
3   If $R^s \cap R^t \neq \emptyset$, stop
```

Figure 3. The proposed algorithm

Some notations in CONVPATH are explained. For any node pair <u, v> with $u \in N_L$ and $v \in N_L$, a shortest path from u to v which goes through only the arcs of A_L will be denoted by $u \rightarrow v$. CONVPATH has two parts. In lines 1-4, variables are initialized. Constant β_s and β_t stores the length of a shortest path from s to the nearest entry and the length of a shortest path from the nearest exit to t, respectively. In lines 4-22, entries and exits for convenient paths are searched repeatedly until one of the termination criteria is satisfied. In each iteration of while-loop, two new nodes are found in lines 6 and 14. If the new node found in line 6 is an element of N_H, it can be an entry of convenient paths. Similarly, if the new node found in line 14 is an element of N_H, it can be an exit of convenient paths. The convenient paths formed by new pairs of entries and exits are considered in lines 8-12 and lines 16-20. In addition, the length of each new convenient path is compared with that of the shortest path ever found in previous iterations. If a new convenient path is shorter than the convenient paths ever found, updating ρ is performed in lines 10 and 18 as described in Property 3. These updates enable us to obtain a tighter lower bound and eventually terminate the algorithm earlier. *E* and *X*, which represents the set of entries and exits ever found, are also updated in lines 11 and 19 at each iteration.

After the next nearest node is considered, CONVPATH calls CHK_STOP_COND to check if one of the termination criteria is already satisfied. The first line of CHK_STOP_COND tests whether $T = \min\{T_s, T_t\}$ satisfies inequality (8). At a certain iteration, CONVPATH have searched all convenient paths whose minor segments' total length is equal to or less than *T*. The second line checks if inequality (11) holds. At the third line, if a minimal convenient path having no major segments is constructed, the algorithm can stop by Property 4.

CONV is based on the bidirectional Dijkstra's algorithm. (See [1] for the bidirectional Dijkstra's algorithm.) In fact, lines 6 and 14 performs the forward and the backward Dijkstra's algorithm, respectively. Line 6 finds one node which has been most recently permanently labelled by the forward Dijkstra's algorithm, and the forward Dijkstra's algorithm from *s* to every other node of N_L can be performed only one time through all iterations of while-loop. The same arguments are applied to line 14 except that the backward Dijkstra's algorithm is performed instead of the forward Dijkstra's algorithm. Therefore, the total amount of computations required for lines 6 and 14 is bounded by $S(|N_L|, |A_L|)$ where $S(|N_L|, |A_L|)$ denotes the worst case complexity of the shortest path problem in the directed network $G_L(N_L, A_L)$.

The complexity of CONVPATH also depends on the amount of computations in lines 8-12 and lines 16-20. Since shortest paths between all pairs of entries and exits in G_H need to be computed in the worst case, the total amount of computations in lines 8-12 and 16-20 is bounded by $O(|N_H \cap N_L|^2 S(|N_H|, |A_H|))$.

Therefore, the computational complexity of CONVPATH is $S(|N_L|,|A_L|)$ + $O(|N_H \cap N_L|^2 S(|N_H|,|A_H|))$. However, since the number of the nodes of G_H is relatively small in a real road network, it is not so impractical to store the length of a shortest path between every pair of an entry and an exit in G_H before CONVPATH starts. In that case, the worst case complexity can be reduced to $S(|N_L|,|A_L|)+O(|N_H \cap N_L|^2)$.

As δ in CONVPATH decreases, a convenient path q^* produced by CONVPATH becomes shorter, and its length approaches to the length of a shortest convenient path p^*. By Property 1 and the termination criteria of CONVPATH, the ratio of $d(q^*)-d(p^*)$ to $d(p^*)$ is bounded by δ. On the other hand, as δ becomes smaller, the number of nodes searched by CONVPATH will increase, and consequently the amount of computations will increase. To illustrate a trade-off between computational efforts and the length of q^*, suppose that the number of nodes to be searched by the forward and backward Dijkstra's algorithm in CONVPATH is inversely proportional to $(1+\delta)$. Then, the total amount of computations of CONVPATH can be approximated to be $S(|N_L|/(1+\delta),|A_L|/(1+\delta))+O(|N_H \cap N_L|^2 / (1+\delta)^2)$. Figure 4 shows the relationship between computational efforts and the ratio of the length difference.

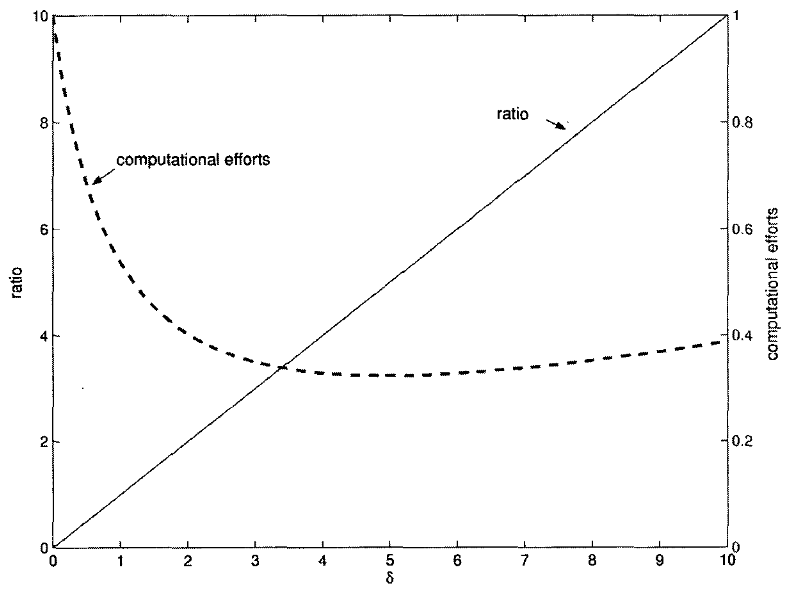

Figure 4. Trade-off between computational efforts and the length of a convenient path

In Figure 4, the curve, labeled by 'ratio' and graduated in the left y-axis, represents the change of the ratio of $d(q^*) - d(p^*)$ to $d(p^*)$ as the parameter δ increases. The other curve, labeled by 'computational efforts' and graduated in the right y-axis, represents the change in the computational effort as δ increases. The graph for computational efforts represents the normalized computational effort which is obtained by dividing an approximated computational complexity of CONVPATH by TC where TC means the amount of computations when $\delta = 0$. Since the best computational complexity of the shortest path problem is known to be $O(|A_L| + |N_L| \log |N_L|)$, the computational complexity of CONVPATH can be approximated to be $O((|A_L| + |N_L| \log |N_L| /(1+\delta)))/(1+\delta) + |N_H \cap N_L|^2 /(1+\delta)^2)$. As δ increases from 0 to 3, the ratio of the length difference to the shortest convenient path's length decreases very sharp, but the increase in computational efforts is relatively slow. In addition, when δ changes from 7 to 10, the reduction of the computational efforts is relatively slight at the sacrifice of computational efficiency. Although these observations seem rather over-simplified and theoretical, they will be validated by experimental results in the next section.

Finally, since CONVPATH is based on the bidirectional Dijkstra's algorithm, some additional computations may be required to find a minimal convenient path which consists of only low-grade arcs. That is, if CONVPATH is stopped by the termination criterion, $R^s \cap R^t \neq \emptyset$, in line 3 of CHK_STOP_COND, we need to apply a simple post-processing procedure to find a minimal convenient path consisting of only low-grade arcs (See [1] for the details).

6. EXPERIMENTAL RESULTS IN A REAL ROAD NETWORK

We implement the proposed algorithm and make a comparison between its performance and the existing algorithms by using a road network data obtained from geographic information systems. The road network covers all expressways and local main roads of South Korea, and has 285,119 nodes and 664,176 arcs. In GIS, each arc has an attribute that gives information about whether it is an expressway or a local main road. By classifying each road of the road network into two groups according to its attribute, we obtain a hierarchical road network. Note that the hierarchical road network is undirected and can be easily transformed to the directed network. The high-level subnetwork has 6,239 nodes and 12,618 arcs, and the low-level subnetwork has 281,525 nodes and 651,558 arcs. The number of nodes which can be entries or exits is only 2,645 because some nodes on the high-level subnetwork have no arcs connected with nodes on the low-level subnetwork. The travel time of each road is derived by dividing its distance by its speed limit where the speed limit of high-grade roads is set to 100km/h and the speed limit of low-grade roads is set to 60km/h. The speed limits are also applied to Property 3, i.e., $v_H = 100\text{km/h}$ and $v_L = 60\text{km/h}$, to obtain a lower bound to the length of a shortest convenient path.

We consider the three algorithms for finding a convenient path as follows:

- **The nearest entry-exit algorithm** (NEAREST): Only the entry nearest to the origin is considered as an entering gateway. Also, only the exit nearest to the destination is considered as an exiting gateway.
- **All pairs algorithm** (ALL): All pairs of entries and exits are considered for finding a shortest convenient path.
- **The proposed algorithm** (CONVPATH) in Figure 3.

Table 1 shows the experimental results of the three algorithms(NEAREST, ALL, CONVPATH) applied to 50 different pairs of origins and destinations. The pairs of origins and destinations are chosen randomly under the constraint that a shortest convenient path between the origin and destination includes a major segment. The first, second and third column of Table 1 represent the data number, the length of a shortest path and the length of a shortest convenient path between the origin and the destination, respectively. The fourth column represents the ratio of the third column to the second column. The fifth column represents a lower bound to a shortest convenient path, and is computed by Property 4. In Table 1, CP means the length of the convenient path found by the corresponding algorithm, and β means the sum of the length of the minor segments of a convenient path. Therefore, the columns β/CP (6-th, 9-th, 13-th, ...) mean the ratio of the minor segments' total length to the length of the convenient path found by the corresponding algorithm. The columns CP/SCP (7-th, 10-th, 14-th, ...) represent the ratio of the length of the convenient path to the length of a shortest convenient path. In addition, the columns PAIRS (8-th, 12-th, 16-th, ...) represent the number of pairs of entry and exit searched by CONVPATH. The columns C (11-th, 15-th, 19-th, ...) represent the termination criterion that makes CONVPATH stop where '1', '2', and '3' mean the line number of CHK_STOP_CON.

Although CONVPATH algorithm is experimented for $\delta = 0, 1, \ldots, 10$, but only the results for $\delta = 0, 0.5, 1, 2,9$ are presented in Table 1. For example, the eighth to eleventh columns show the experimental result of CONVPATH algorithm with

Table 1. The experimental results

NO	SP	SCP	SCP/SP	ρ	NEAREST		CONVPATH($\delta \cong 9$)				CONVPATH($\delta \cong 2$)			
					β /CP	CP/SCP	pairs	β /CP	CP/SCP	C	pairs	β /CP	CP/SCP	C
1	32325	32325	1.000	2200	0.155	1.094	45	0.161	1.083	1	37800	0.198	1.000	1
2	28846	28846	1.000	1840	0.059	1.262	289	0.064	1.251	1	83304	0.366	1.000	1
3	28287	28295	1.000	1600	0.231	1.643	58	0.231	1.643	1	3596	0.542	1.016	1
4	25847	25847	1.000	1800	0.136	1.013	13	0.136	1.013	1	27984	0.147	1.000	1
5	30428	30428	1.000	1560	0.208	1.001	45	0.211	1.000	1	4480	0.211	1.000	1
6	37616	37616	1.000	2300	0.222	1.081	87	0.240	1.054	1	27590	0.296	1.000	1
7	20719	20719	1.000	1620	0.193	1.000	3	0.193	1.000	2	3577	0.193	1.000	1
8	30780	30780	1.000	2110	0.151	1.091	1	0.151	1.091	2	12400	0.198	1.000	1
9	44979	45005	1.000	2250	0.251	1.034	24	0.264	1.007	1	481	0.264	1.007	1
10	37165	37165	1.000	2380	0.088	1.058	117	0.088	1.058	1	16080	0.223	1.000	1
11	25078	25078	1.000	1450	0.404	1.056	22	0.404	1.056	1	22	0.404	1.056	1
12	42255	42255	1.000	2370	0.211	1.083	3	0.211	1.083	1	6076	0.343	1.000	1
13	36295	36295	1.000	2690	0.069	1.169	176	0.091	1.002	2	165240	0.092	1.000	1
14	47888	47888	1.000	2750	0.183	1.029	46	0.183	1.029	1	97403	0.200	1.000	1
15	20819	20819	1.000	1200	0.293	1.322	2	0.293	1.322	1	42	0.296	1.322	1
16	27112	27112	1.000	1470	0.262	1.616	14	0.262	1.616	1	240	0.442	1.286	1
17	28117	28117	1.000	1930	0.151	1.037	32	0.194	1.034	1	33196	0.293	1.000	1
18	45750	45847	1.002	2750	0.177	1.098	65	0.182	1.093	1	93015	0.529	1.024	1
19	42956	42956	1.000	2890	0.206	1.000	137	0.206	1.000	1	53215	0.207	1.000	1
20	38927	39624	1.017	2270	0.205	1.023	12	0.205	1.023	1	3905	0.218	1.000	1
21	16583	16583	1.000	940	0.248	1.015	29	0.253	1.010	1	874	0.266	1.000	1
22	48874	49362	1.010	2730	0.231	1.278	110	0.240	1.273	2	3036	0.307	1.113	1
23	40517	41837	1.032	2930	0.257	1.057	83	0.288	1.040	1	4255	0.316	1.000	1
24	17359	17359	1.000	1450	0.182	1.885	116	0.483	1.000	1	116	0.483	1.000	1
25	33113	33940	1.025	1940	0.093	1.155	42	0.098	1.146	1	47376	0.098	1.146	1
26	46833	48153	1.028	3350	0.223	1.113	59	0.249	1.035	1	4698	0.273	1.000	1
27	46151	46151	1.000	2810	0.185	1.052	54	0.207	1.030	1	61093	0.226	1.000	1
28	26846	27532	1.025	1200	0.152	2.340	92	0.152	2.340	1	6672	0.346	1.148	1
29	48660	48660	1.000	2640	0.197	1.070	55	0.197	1.070	2	11990	0.253	1.000	1
30	43761	43761	1.000	2820	0.311	1.021	173	0.417	1.012	1	173	0.417	1.012	1
31	17426	17426	1.000	930	0.341	1.324	9	0.363	1.254	1	168	0.386	1.230	1
32	24830	24830	1.000	1800	0.148	1.307	11	0.148	1.307	2	4450	0.303	1.034	1
33	31939	32056	1.003	1420	0.319	1.118	93	0.336	1.116	1	93	0.336	1.116	1
34	33503	33774	1.008	1970	0.196	1.050	10	0.196	1.050	1	5995	0.324	1.030	1
35	37968	37976	1.000	1660	0.203	1.189	6	0.208	1.184	1	858	0.325	1.058	1
36	34662	34670	1.000	1730	0.190	1.079	1	0.190	1.079	2	4950	0.350	1.000	1
37	38677	38677	1.000	2550	0.061	1.034	693	0.075	1.000	1	288015	0.075	1.000	1
38	22733	23430	1.030	1490	0.081	1.000	32	0.086	1.000	1	3850	0.086	1.000	1
39	5276	5276	1.000	40	0.355	1.052	34	0.419	1.000	1	34	0.419	1.000	1
40	15409	15409	1.000	840	0.366	1.075	6	0.383	1.053	1	6	0.383	1.053	1
41	44838	45094	1.005	3140	0.057	1.044	2376	0.124	1.002	2	298592	0.128	1.000	1
42	22005	22005	1.000	1670	0.189	1.015	16	0.191	1.012	1	5427	0.239	1.000	1
43	43134	43506	1.008	2420	0.118	1.000	11	0.118	1.000	2	6561	0.118	1.000	1
44	31515	31515	1.000	2270	0.164	1.066	25	0.164	1.066	1	14522	0.178	1.000	1
45	33739	33741	1.000	1500	0.183	1.708	50	0.218	1.639	1	19926	0.450	1.045	1
46	41321	41634	1.007	2010	0.206	1.094	33	0.206	1.094	1	1264	0.275	1.000	1
47	41779	41779	1.000	2350	0.384	1.034	65	0.384	1.034	2	65	0.384	1.034	1
48	36063	36071	1.000	1930	0.209	1.188	13	0.209	1.188	1	600	0.270	1.000	1
49	37162	37859	1.018	2150	0.210	1.077	5	0.210	1.077	1	5335	0.325	1.000	1
50	44437	44437	1.000	3300	0.015	1.024	20200	0.027	1.000	1	256802	0.027	1.000	1

Table 1. Continued

	CONVPATH($\delta \cong 1$)				CONVPATH($\delta \cong 0.5$)				CONVPATH($\delta = 0$)			
	pairs	β /CP	CP/SCP	C	pairs	β /CP	CP/SCP	C	pairs	β /CP	CP/SCP	C
1	349236	0.198	1.000	1	637599	0.198	1.000	3	637599	0.198	1.000	3
2	351107	0.366	1.000	1	457828	0.366	1.000	3	457828	0.366	1.000	3
3	12403	0.571	1.000	1	162590	0.571	1.000	1	243760	0.571	1.000	3
4	272626	0.147	1.000	1	552680	0.147	1.000	3	552680	0.147	1.000	3
5	68992	0.211	1.000	1	97980	0.211	1.000	3	97980	0.211	1.000	3
6	493190	0.296	1.000	1	669130	0.296	1.000	3	669130	0.296	1.000	3
7	28892	0.193	1.000	1	212058	0.193	1.000	1	275552	0.193	1.000	3
8	119085	0.198	1.000	1	438075	0.198	1.000	3	438075	0.198	1.000	3
9	125800	0.381	1.000	1	134568	0.381	1.000	3	134568	0.381	1.000	3
10	145080	0.223	1.000	1	406026	0.223	1.000	3	406026	0.223	1.000	3
11	8249	0.434	1.000	1	81190	0.434	1.000	1	193887	0.434	1.000	3
12	100864	0.343	1.000	1	445936	0.343	1.000	1	592662	0.343	1.000	3
13	648810	0.092	1.000	3	648810	0.092	1.000	3	648810	0.092	1.000	3
14	701250	0.200	1.000	1	959418	0.200	1.000	3	959418	0.200	1.000	3
15	4158	0.613	1.000	1	25676	0.613	1.000	1	40128	0.613	1.000	3
16	25454	0.678	1.012	1	159528	0.692	1.000	1	215201	0.692	1.000	3
17	230910	0.293	1.000	1	376128	0.293	1.000	3	376128	0.293	1.000	3
18	461686	0.647	1.000	1	893418	0.647	1.000	1	919240	0.647	1.000	3
19	449228	0.207	1.000	1	746327	0.207	1.000	3	746327	0.207	1.000	3
20	74520	0.218	1.000	1	220459	0.218	1.000	3	220459	0.218	1.000	3
21	8722	0.266	1.000	1	30336	0.266	1.000	1	35700	0.266	1.000	3
22	51205	0.356	1.000	1	201804	0.356	1.000	1	323736	0.356	1.000	3
23	59220	0.316	1.000	1	413284	0.316	1.000	3	413284	0.316	1.000	3
24	2358	0.484	1.000	1	46781	0.484	1.000	1	96383	0.484	1.000	3
25	424473	0.654	1.013	1	733392	0.679	1.000	3	733392	0.679	1.000	3
26	67925	0.273	1.000	1	264355	0.273	1.000	1	433068	0.273	1.000	3
27	631400	0.226	1.000	1	911976	0.226	1.000	3	911976	0.226	1.000	3
28	80969	0.907	1.000	1	86829	0.909	1.000	3	86829	0.909	1.000	3
29	148680	0.253	1.000	1	483538	0.253	1.000	3	483538	0.253	1.000	3
30	251563	0.492	1.000	1	497424	0.492	1.000	1	680525	0.492	1.000	3
31	4635	0.567	1.000	1	18624	0.567	1.000	1	22848	0.567	1.000	3
32	98523	0.370	1.000	1	347310	0.370	1.000	3	347310	0.370	1.000	3
33	91869	0.700	1.007	1	201310	0.722	1.000	3	201310	0.722	1.000	3
34	77688	0.507	1.000	1	202288	0.507	1.000	3	202288	0.507	1.000	3
35	24009	0.445	1.000	1	151998	0.445	1.000	3	151998	0.445	1.000	3
36	47432	0.350	1.000	1	70200	0.350	1.000	3	70200	0.350	1.000	3
37	620740	0.075	1.000	1	916426	0.075	1.000	3	916426	0.075	1.000	3
38	26112	0.086	1.000	1	77035	0.086	1.000	3	77035	0.086	1.000	3
39	2736	0.421	1.000	1	10324	0.421	1.000	1	39552	0.421	1.000	3
40	1230	0.514	1.000	1	13950	0.514	1.000	1	22050	0.514	1.000	3
41	652188	0.128	1.000	1	765094	0.128	1.000	3	765094	0.128	1.000	3
42	48790	0.239	1.000	1	124764	0.239	1.000	1	159317	0.239	1.000	3
43	189980	0.118	1.000	1	885920	0.118	1.000	3	885920	0.118	1.000	3
44	196847	0.178	1.000	1	394082	0.178	1.000	3	394082	0.178	1.000	3
45	109056	0.736	1.000	1	223020	0.736	1.000	3	223020	0.736	1.000	3
46	125028	0.275	1.000	1	233835	0.275	1.000	3	233835	0.275	1.000	3
47	45588	0.493	1.000	1	362250	0.493	1.000	1	490500	0.493	1.000	3
48	36556	0.270	1.000	1	300120	0.270	1.000	3	300120	0.270	1.000	3
49	48530	0.325	1.000	1	228285	0.325	1.000	3	228285	0.325	1.000	3
50	578925	0.027	1.000	1	795854	0.027	1.000	3	795854	0.027	1.000	3

$\delta = 9$. The number of the searched pairs of entries and exits is 45, and the length of the minor segments of the found convenient path is 0.161-times its total length. The length of the found convenient path is 1.083-times longer than that of a shortest convenient path, and the algorithm is terminated by the condition. $\min\{T_s, T_t\} > d(q^*)/(1+\delta)$. Note that the number of pairs of entries and exits searched by NEAREST algorithm is always 1, and the number of pairs of entries and exits searched by ALL algorithm is always the same as by CONVPATH algorithm with $\delta = 0$. Also, note that the convenient path found by CONVPATH with $\delta = 0$ is a shortest convenient path.

We make some remarks about the experimental results. First of all, we find that there is little difference between the length of a shortest convenient path and that of a shortest path. In 50 pairs of origins and destinations, s shortest convenient path is at most 1.05-times longer than a shortest path on the average. This implies that the hierarchical approach and the concept of convenient paths are useful in real road networks.

Next, the experimental results show that the difference between the length of a shortest convenient path and the length of the convenient path found by NEAREST algorithm is not negligible. In the worst case, the latter is about 2.34-times longer than the former in Table 1. It means that NEAREST algorithm, which has been most popular so far, may fail to produce a convenient path acceptable to drivers.

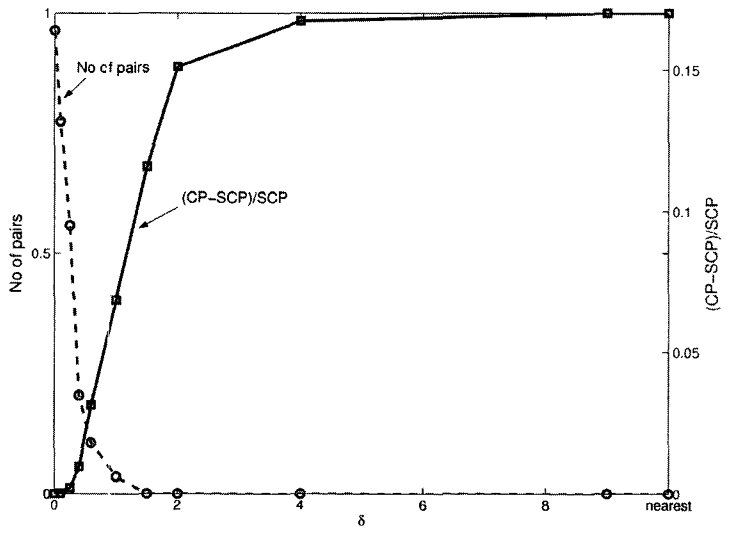

Figure 5. The length of convenient paths vs the number of pairs of entries and exits

Thirdly, CONVPATH algorithm can be a good alternative for finding a convenient and nearly shortest path. In Figure 5, the one trend line, labelled by '(CP-SCP)/SCP' and graduated in the right y-axis, represents the change of the length difference between the final path and a shortest convenient path, as the parameter δ changes from $\delta = 0$ to $\delta = 9$. After dividing the length difference between the final convenient path and a shortest one by the length of the shortest convenient path, the ratios are averaged for the 50 data in Table 1. Similar calculations are applied to NEAREST algorithm for comparison with CONVPATH. The other trend line, labeled by 'No of pairs' and graduated in the left y-axis, represents the relative number of pairs of entries and exits searched by NEAREST and CONVPATH with $\delta = 0,0.1,\ldots,9$. Each point on the trend line implies the ratio of the average number of pairs of entries and exits searched by NEAREST or CONVPATH to the average number of pairs searched by ALL algorithm. As shown in Figure 5, the convenient paths found by NEAREST algorithm are on the average about 1.16-times longer than those found by ALL algorithm. However, the convenient paths found by CONVPATH algorithm with $\delta = 4$ are on the average about 1.09-times longer than those found by ALL algorithm, while the average number of pairs of entries and exits increases to about 1 percent of the average number of pairs of entries and exits searched by ALL algorithm. When $\delta = 2$ and $\delta = 1.5$, the convenient paths are about 1.03-times and 1.01-times longer than those found by ALL algorithm, while the average number of pairs of entries and exits increases to about 5 percent and 18 percent of the average number of pairs of entries and exits searched by ALL algorithm, respectively. When $\delta = 1$, CONVPATH algorithm finds shortest convenient paths in 45 out of 50 data, but the average number of pairs of entries and exits searched becomes rather large. When δ is less than 0.6, the improvement in the length of convenient paths does not seem to compensate for the increase in the number of pairs of entries and exits. To sum up, CONVPATH algorithm with $\delta = 4$ or $\delta = 2$ appears to be a good alternative to find a convenient path that is short enough to be accepted by drivers. For finding a convenient path with the almost shortest length, CONVPATH algorithm with $\delta = 1.5$ or $\delta = 1$ will be a good alternative whose computational efforts are moderate. Comparing Figure 5 with Figure 4, the experimental results in Figure 5 are consistent with the theoretical observation represented in Figure 4.

Finally, the termination criterion derived from inequality 8, which is represented as '1' in column 'C' of Table 1, appears to be effective for CONVPATH algorithm with δ being less than about 0.6. On the other hand, when δ is greater than 0.6, the termination criterion derived from Property 4, which is represented as '3', made CONVPATH algorithm terminate in most problems. Lower bounds to the length of convenient paths, which are calculated by Property 3, did not work very well in our experiment. However, this result will not preclude us from using lower bounds to the length of convenient paths as an effective termination criterion. If tight lower bounds are available from previous calculations or experience, the termination criterion derived from inequality (11) is expected to be effective.

7. CONCLUSIONS

In this paper, we defined a convenient path that exploits the hierarchical structure of road networks, and proposed an algorithm for finding a convenient path that is short enough to be accepted by drivers. By our experimental results, the introduction of a convenient path turns out to be an effective method for finding a human-oriented and nearly shortest path with a little computation.

We were concerned with the hierarchical network with two levels, but the generalization of our results in the hierarchical network with more than two levels will be worth further research. Also, the application of the proposed algorithm to real road networks and empirical analysis of the results is an interesting topic for future research.

REFERENCES

[1] Ahuja, R. K., T. L. Magnanti, and J. B. Orlin, *Networks flows: Theory, algorithms and applications*, Prentice-Hall, 1993.

[2] Campbell, J. F., "Selecting routes to minimize urban travel time," *Transportation Research-Part*, 26B (1992), 261-274.

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

[4] Duckham, M. and L. Kulik, "Simpliest paths: Automated route selection for navigation," In W. Kuhn, M. F. Worboys, and S. Timpf(Eds), *Spatial Information Theory: Foundations of Geographic Information Science, Lecture Notes in Computer Science*, Vol.2825 (2003), 182-199, Springer, Berlin.

[5] Liu, B., "Route finding by using knowledge about the road network," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Hu-*

mans, 27 (1997), 436-448.

[6] Seong, M., K. Sung, and S. Park, "Finding a route in a hierarchical urban transportation network," *Proceedings of 5th World Congress on Intelligent Transport Systems(10.12-16, 1998)*, Korea.

[7] Shapiro, J., J. Waxman, and D. Nir, "Level graphs and approximate shortest path algorithms," *Networks*, 22 (1992), 691-717.