A fast node-directed multipath algorithm: Dijkstra-Hyperstar

Jiangshan Ma[1], Daisuke Fukuda[2] and Jan-Dirk Schmöcker[2]

[1]Ph.D. Candidate, Dept. of Civil and Environmental Eng., Tokyo Institute of Technology
(Ookayama, Meguro-ku, Tokyo 152-8552, Japan)
E-mail:ma.js@plan.cv.titech.ac.jp
[2]Member of JSCE, Associate Professor, Dept. of Civil and Environmental Eng., Tokyo Institute of Technology
(Ookayama, Meguro-ku, Tokyo 152-8552, Japan)
fukuda@plan.cv.titech.ac.jp
[3]Member of JSCE, Associate Professor, Dept. of Civil Eng., Kyoto University
(Katsura Campus, C1-2-431, Kyoto 615-8540, Japan)
schmoecker@trans.kuciv.kyoto-u.ac.jp

Abstract: This paper studies the speed of three link-to-link multipath algorithms with the purpose of improving their applicability for route navigation. These previously established algorithms are the original Spiess & Florian coding of the strategy approach for transit passengers and two derivations of it, the so-called Hyperstar-Dijkstra (HPD) algorithm and the Hyperstar (HS) algorithm. The latter firstly applied the strategy approach to route navigation. A new speed-up algorithm, named Dijkstra-Hyperstar (DHS), is proposed by utilizing, firstly, Dijkstra shortest distances as heuristics and, secondly, the idea of a node directed search which reduces the link set to be checked. This is achieved by "dynamically" confining the link set by gradually extending it to include those from newly updated node. The second improvement can also be independently applied to the original Spiess & Florian (for transit network) or Adapted Spiess & Florian (ASF, for road network) algorithms without heuristics. The performances among these four algorithms are compared in synthetic networks and a significant improvement in running time can be attained with the newly proposed DHS algorithm. Applications to both transit assignment as well as route navigation are discussed.

Key Words : *Adapted Spiess & Florian (ASF), Hyperstar (HS), Hyperpath-Dijkstra (HPD), Dijkstra-Hyperstar (DHS), Node directed search, Route navigation*

1. INTRODUCTION

Shortest path algorithms, digital maps and positioning technologies have made vehicle navigation come true and the market has been spreading. The cost to produce navigation equipment is continuously decreasing so that there is a fast growing number of global positioning system (GPS) integrations with many IT products such as mobile phones, personal digital assistants (PDA) and, nowadays, digital cameras. The traditional in-vehicle route navigation equipment has been challenged by these GPS-integrated terminals. To keep their market position, specialized services are added to vehicle route navigation equipment. These are often based on real traffic data, in many cases with the aim to provide more alternative routes to meet with various driver preferences.

In present navigation systems, the A-star shortest path algorithm introduced by Hart et al. (1968) and Pearl (1984) has a pivotal role. As a heuristic version of Dijkstra's algorithm (Dijkstra, 1959) the speed-up of searching by A-star is remarkable but it may not guarantee the shortest path if the heuristic estimations surpass the real distances. Besides, since the heuristic of A-star is based on external information, the exact time complexity is entirely unforeseen. The estimation accuracy of the rest distance from the current node to the destination is a key issue for the efficiency of A-star: On the one hand, if the correlation of the distance estimation with the crow-fly distance is too weak, the time complexity of the algorithm is not significantly improved; on the other hand, if the correlation is too strong, the risk of non-optimal solutions may occur (Hofmann-Wellenhof, 1995).

An ideal heuristic will always return the actual minimum cost to reach the goal (Wichmann, 2004). A perfect knowledge of the remaining distance will reduce the nodes involved in the A-star search and guarantee the shortest path availability at the same time. In real applications, the Euclidean distance is widely adopted as the heuristic number for tile-based maps. A popular way for digital map based dynamic vehicle navigation is to carry out routing by a Dijkstra algorithm first from the reverse direction (from the destination to the origin) and then en-route rerouting by A-star algorithm with the shortest distance heuristics generated in the first step (e.g. Ben-Akiva et al. 1997).

Both, Dijkstra and A-star algorithms, only generate the shortest path according to the pre-determined cost index. In traffic networks, this index can be either road distance, travel time or a combination taken further factors into account (e.g. generalized cost). However, there are at least three issues that make it difficult to apply a single shortest path suggestion to route navigation:

Firstly, travel times and other aspects determining the attractiveness of a route are often uncertain, which means the recommended shortest path may also change during a journey. Secondly, travelers have different preferences, which means the suggested route might not be the one preferred by the traveller. Thirdly, from a network perspective, there is a potential risk that suggesting single routes causes congestion if the market penetration of navigation devices is high (e.g. Ben-Akiva et al. 1997). For all these reasons, it is suggested that multiple route navigation is preferred for the benefit of users and/or the traffic system. In some commercial applications today, hence multiple paths are provided by showing users the shortest path for different preferences (distance, money, travel time, etc.). Commercial applications that consider the potential delay of links to suggest a-priori a route set bundle, are, however, to our knowledge not yet on the market.

In research papers, the *K*-shortest path algorithm is well studied and often utilized to generate multiple route alternatives from the whole network (e.g. Eppstein, 1994, 1998; Jiménez, 1999, 2003). Popular algorithms, entailing path removing and path deviation were tested and compared by Santos (2006) who also proposed a new path deviation *K*-shortest path algorithm. A possible problem of *K*-shortest paths algorithms is that they are route-based and suggested paths may significantly overlap. Therefore, algorithms are often modified by some constraints before being applied to traffic networks. The typical way is to compute a sufficiently large path set first and then delete the disqualified paths by checking the constraints. For example, Zijpp et al. (2005) proposed a constrained *K*-path algorithm that finds the feasible routes and can be applied in combination with a wide class of constraints. To speed up the *K*-shortest path enumeration, Chen et al. (2007) integrated the A-star heuristic into the *K*-shortest path finding process. By setting different constraints such as max distance, travel time, historical probability to hit a failed link and some other factors, more reliable paths are attainable. Similarly aiming to provide reliable route navigation, Kaparias and Bell (2009) took the route reliability directly as a route constraint and compared the proposed guidance with those from existing commercial route navigation equipment. Considering the turn constraints and route similarity, Yongtaek and Hyunmyung (2005) proposed a link-based multipath algorithm and criteria of route dissimilarity were integrated into the algorithm to create heterogeneous routes. Though link-based, the algorithm of Yongtaek and Hyunmyung is still classified as a *K*-shortest path based algorithm, with the difference that the labeled units are links instead of nodes.

Not only for route navigation but also for traffic assignment path generation methods such as *K*-shortest path algorithms are used. However, different to path-based route choice modeling, for example, Fosgerau et al. (2009) recently proposed a link-based route choice model that considers the route choice as a sequence of link choices. The idea of link-to-link route choice is in fact similar to strategy based route choice proposed for transit assignment proposed by Spiess and Florian (1989) and by Nguyen and Pallotino (1988) who also introduce the term “hyperpath” to transit assignment. A hyperpath refers to a set of paths any of which is potentially optimal due to travel time uncertainty on one or more links. In transit assignment this uncertainty might be created by the arrival time of vehicles. With the assumption that one takes the first-coming vehicle from the set of attractive paths, the optimal strategy that minimizes the total expected in-vehicle travel time and waiting time of transit passenger was formulated as a linear programming problem.

Cominetti and Correa (2001) proposed a method named Hyperpath-Dijkstra (HPD), which was adapted from Dijkstra’s shortest path algorithm, and declared a better theoretical time complexity than Spiess & Florian’s original algorithm to determine the same hyperpath. However, although they mentioned the importance of an empirical comparison, such a comparison between the two algorithms has not been done by others so far. Inspired by the Spiess & Florian Algorithm (SFA)

for frequency-based transit assignment, Bell (2009) adapted this algorithm to road networks with uncertain travel times by defining the link frequency as the reciprocal of maximum delay. Consequently, the Adapted Spiess & Florian algorithm (ASF) for road network was introduced. (Note that SFA and ASF are of no difference except their model interpretation. SFA deals with transit networks and ASF deals with road networks.) Bell (2009) also proposed an A-star-like heuristic version named Hyperstar (HS) algorithm and suggested its application to route navigation in road networks.

Without requiring any path enumeration, the HS algorithm provides us with a novel solution to multipath navigation and link-to-link multipath navigation, which could be more appropriate to represent drivers' actual behavior. However, as an algorithm of polynomial time complexity, its speed is mostly dependent on the number of links, which is different from Dijkstra algorithm's dependence on the number of nodes. In traffic networks, the number of links generally far exceeds the number of nodes and therefore it is necessary to conceive a better link searching strategy to accelerate the algorithm. The HPD algorithm considers the node-link topology which in general leads to an improved time complexity. This is even though, just as Dijkstra's algorithm, the HPD algorithm searches every node to obtain the hyperpath. The HS algorithm is a heuristic algorithm that is claimed to be faster than the Spiess & Florian algorithm, though again to our knowledge empirical comparisons have not been carried out so far. A main reason might be that the Spiess and Florian as well as the HPD algorithm are mainly intended for transit assignment where the efficiency requirement is not as stringent. However, since Bell (2009) started applying these algorithms in the navigation filed, their efficiency becomes a rigorous problem. By our testing results in large networks, it can be found that the HS algorithm is still far from satisfying the required speed for navigation applications, albeit it is better than the ASF algorithm. The HPD is much better than these two but we believe could be more intelligent: It searches all the nodes in the network regardless of the OD pair, which brought about our intention to find a better algorithm to attain a higher efficiency.

The purpose of this paper is hence to propose and verify the idea of improving the speed of the HS algorithm with the aim of applicability for multipath navigation. There are two modifications to the original HS algorithm: First, the heuristic distances required for the A-star algorithm are generated by running Dijkstra's algorithm from the origin so that the links involved in the search tend to be reduced. Second, a node-directed link selection is adopted by utilizing the pre-stored topology information so that the link selection process only involves those links that have been updated. This so-called node-directed link selection maintains the link list dynamically by taking advantage of the connecting information among nodes and links. In line with Bell (2009), assuming a relative low rate of navigation in use during a certain time interval, we will mainly focus on the speed improvement for navigation instead of the effect of navigation to the network that is involved in dynamic traffic assignment topics.

The rest of this paper is organized as follows. In Section 2, the previously proposed three algorithms to create hyperpaths are outlined. Because of the similarity between HS and ASF, these two algorithms are illustrated together. Section 3 presents the speed-up methods which lead to the proposed Dijkstra-Hyperstar (DHS) algorithm. The empirical performance comparisons among these four algorithms are illustrated in Section 4. Finally, in Section 5, some conclusions are drawn regarding applications for route guidance as well as transit assignment.

2. ASF, HS, AND HPD ALGORITHMS

In Spiess and Florian (1989), each link has a service frequency of which the inverse is assumed to describe the expected link waiting time. In Bell (2009), by analogy to Spiess and Florian's model for transit networks, a road link is considered to provide a service that may be subject to a delay. Bell interprets the inverse of service frequency as the maximum delay. Higher service frequency, therefore, indicates a lower maximum delay which leads to higher travel time reliability. Drivers are assumed to aim to minimize their maximum exposure to delay and are hence developing a strategy potentially including a large number of alternative paths. With this new interpretation of the optimal strategy Bell (2009) attempts to solve the multipath problem for road network applications. Firstly, Bell suggests an adapted version of ASF, then, aiming to improve the efficiency of algorithm, and inspired by the well-known A-star algorithm, he added a heuristic variable into the ASF algorithm to improve the algorithm's performance. This intelligent version was named Hyperstar (HS) and the algorithm requires neither iterative use of penalties, *K*-shortest path algorithm based path enumeration, nor Monte Carlo simulation, which means it a promising method for route navigation.

Unlike Bell's heuristic idea to improve the speed of the Spiess & Florian algorithm, Cominetti and

Correa (2001) focused on the maintenance of the node updating process and proposed the Hyperpath-Dijkstra (HPD) algorithm. By iteratively manipulating the travel time from nodes to the destination and the set of solved nodes, it has a preferable theoretical time complexity of $O(n * \log n + m * \log\delta)$ where n and m are the number of nodes and links respectively and δ represents the maximum number of outgoing links from a single node. The HPD algorithm does not maintain the links, thus some links may become no longer optimal when additional links from the currently updated node are checked. Consequently, compared to the ASF or HS algorithm, it needs an extra process to recheck the hyperpath links from every current node.

For formulation, consider a network graph *Q* which is represented with a set of nodes and links (*N*, *L*). Most of the following notation is consistent with Bell (2009).

Define the following variables that are universally used in this paper:

r: Origin node,
s: Destination node,
L: Set of links,
TL: Set of temporary links,
N: Set of nodes,
H: Set of links containing the hyperpath between nodes *r* and s,
H_i: Subset of *H* containing the links outgoing from node *i*,
UN: Set of updated nodes
UL: Set of updated links,
ω: The most recently updated node,
S: Set of links which have been selected,
$L^+(i)$: Set of outgoing links from node *i*,
$L^-(i)$: Set of links leading into node *i*,
$i(a)$ Tail node of link *a*
$j(a)$ Head node of link *a*
d_a: Maximum delay on link *a*,
p_a: Probability that link a is used,
c_a: Uncongested travel time on link *a*,
u_i: Expected travel time from node *i* to node *s* by pessimists who minimize maximum exposure to delay,
h_i: Potential at node i with respect to node *r*,
sp_i: Shortest uncongested travel time from node *i* to node *r*,
M: A large number whose size depends on the precision of computation.

Table 1 Pseudo codes of ASF and HS algorithms

Step 0: Variables initialization
$u_i \leftarrow \infty;\ i \in N - \{s\};\ u_s \leftarrow 0;$
$if\ d_a > 0, f_a \leftarrow \frac{1}{d_a}, else\ f_a \leftarrow M;\ f_i \leftarrow 0, i \in N;$
$y_i \leftarrow 0, i \in N - \{r\};\ y_r \leftarrow 1, H \leftarrow \emptyset;\ TL \leftarrow L;$
$h_i \leftarrow$ *the number of grid gaps*
Step 1: Selecting step
for every link in TL,
find link a with the minimum $u_{j(a)} + c_a + h_{i(a)}$, $TL - \{a\}$
Step 2: Updating step
$if\ u_i \geq u_j + c_a\ then$
$if\ u_i = \infty\ and\ f_i = 0\ then$
$\beta \leftarrow 1\ else\ \beta \leftarrow f_i u_i$
$u_i \leftarrow \frac{\left(\beta + f_a(u_j + c_a)\right)}{f_i + f_a}, f_i \leftarrow f_i + f_a, H \leftarrow H + \{a\}$
if $TL = \emptyset$ *or* $u_j + c_a + h_i > u_r$, *go to step 3 else go to step 1*
Step 3: Loading step
for every link $a \in L$ *in decreasing order of* $u_j + c_a + h_i$,
$if\ a \in H\ then\ \ p_a \leftarrow (f_a/f_i)y_i\ and\ y_j \leftarrow y_j + p_a\ else\ p_a = 0.$

In **Table 1**, the ASF and HS algorithms are illustrated together for simplicity and their similarities and differences can be easily observed. If one omits the shadowed and underlined parts, the HS algorithm reduces to the ASF algorithm. In the HS the heuristic information hi of a node is set to be the number of grid gaps from the origin and the link-to-link path search is from the destination to the origin so that this heuristic information will reduce the search by selecting the search direction. In many shortest path problems, it is usual to search the Dijkstra shortest path at first and then use the A-star algorithm for possible subsequent searches to the same destination under slightly changed network conditions. Following this idea, the shortest distance will be used as the heuristic information in this paper.

From the pseudo code illustrated in **Table 1**, the theoretical time complexity of the ASF algorithm can be analyzed as $O(\sum_{i=1}^{m} i^2)$, thus $O(m^3)$ whereas the time complexity of the HS algorithm is dependent on the heuristic information and unknown beforehand. Incorporating an improvement to the Dijkstra algorithm by utilizing prior queue data structures such as Fibonacci heap, the actual time complexity of the ASF algorithm can be reduced to $O(\sum_{i=1}^{m} i * \log i)$, thus $O(m * (\log m)^2)$ (Cormen, 1991). Apparently, as mentioned in Cominetti and Correa (2001), the ASF algorithm should be much slower than HPD algorithm by comparing the theoretical time complexity. For a better understanding of the HPD algorithm the pseudo code is illustrated in **Table 2**.

Table 2 Pseudo codes of HPD algorithm

Step 0: Variables initialization
$u_i \leftarrow \infty, H_i \leftarrow \emptyset,\ \forall\ i \in N - \{s\}, u_s \leftarrow 0;$
$f_a \leftarrow 1/d_a\ if\ d_a > 0, else\ f_a \leftarrow M;$
$f_i \leftarrow 0, i \in N;$
$y_i \leftarrow 0, i \in N - \{r\}, y_r \leftarrow 1, H \leftarrow \emptyset, UN \leftarrow \emptyset$
Step 1: Updating step

1.1 Generating hyperpath set for each node

While $UN \neq N$,

node updating

for every node in $N - UN$, *find node* j *with min* u_j

for each link $a \in L^-(j)$

if $i(a) \in N - \{s\}$, *then* $u_i \leftarrow u_j + c_a$

if $u_{i(a)} > u_{j(a)} + c_a$

if $u_i = \infty$ *and* $f_i = 0$ *then* $\beta \leftarrow 1$ *else* $\beta \leftarrow f_i u_i$

$u_i \leftarrow \frac{\left(\beta + f_a(u_j + c_a)\right)}{f_i + f_a}$, $f_i \leftarrow f_i + f_a$, $H_i \leftarrow H_i + \{a\}$

link re-check

for each link $b \in H_{i(a)}$

if $u_i \leq u_{j(b)} + c_b$

$u_i \leftarrow \frac{\left(\beta - f_b(u_{j(b)} + c_b)\right)}{f_i - f_b}$, $f_i \leftarrow f_i - f_b$, $H_i \leftarrow H_i - \{b\}$

$UN \leftarrow UN + \{j\}$

1.2 Combining hyperpath set

for each node $i \in N$, $H \leftarrow H + H_i$

Step 2: Loading step

for each link $a \in L$ *in decreasing order of* $u_j + c_a$,

if $a \in H$ *then* $p_a \leftarrow (f_a/f_i)y_i$ *and* $y_j \leftarrow y_j + p_a$ *else* $p_a = 0$.

Bell (2009) tested the ASF and HS algorithms in a synthetic 8 by 8 grid network with 64 nodes and 224 links (both directions). In the network, i and j are node IDs and $c_{(i,j)}$ is the undelayed travel time which is the same for both directions of a link. R represents the maximum delay that is generated from a random number between 0 and 1. (See **Fig.1**)

i	j	$c_{(i,j)}$	R
1	2	1.5000	0.5313
2	3	1.3610	0.7027
3	4	1.7729	0.1717
4	5	1.8187	0.6617
5	6	1.8722	0.9257
6	7	1.5820	0.9747
7	8	1.5706	0.3410
9	10	1.5597	0.7933
10	11	1.4173	0.8680
11	12	1.2747	0.8280
12	13	1.1977	0.3443
...	...	...	...

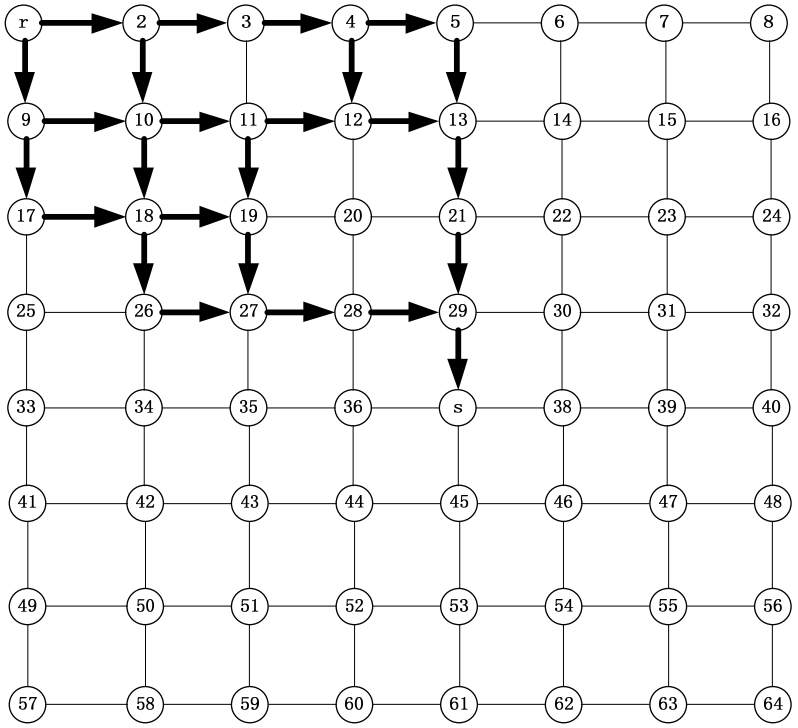

Fig.1 Hyperpath generated in Bell's synthetic road network (depicted from Bell, 2009)

The performances of ASF and HS algorithms are compared by the number of selected links (See **Table 3**). The table shows the links selected in Step 1 of the HS which accounts for most of the running time under 3 different maximum delay levels. The result of scenario 3 corresponds to **Fig.1**. Scenario 2 generates less links because of its lower maximum delay than Scenario 3. Scenario 1 ignores the potential delay and only generates the shortest path. One may refer to Bell (2009) for more detailed information.

Table 3 Comparative selected links of the HS algorithm (depicted from Bell, 2009)

Scenario ID	Maximum delay (d)	Number of selected links	
		ASF	HS
1	0	219	79
2	0.3R	222	111
3	R	223	148

3. SPEED-UP ALGORITHM: DIJKSTRA-HYPERSTAR (DHS)

By inspecting the ASF algorithm, one can observe that the efficiency mainly depends on two

indices: the number of selected links and the number of links that are searched to find a selected link. Since the topological relations among nodes and links are available, the actually involved links in the searching process can be well managed dynamically or "on the fly" by utilizing the pre-stored information of the in- and outgoing links. In the following it will be shown that with a slightly more complicated management of the link set to be checked, the speed of the algorithm dramatically improves if the link updating step is directed so that a significant number of unnecessary link checks can be excluded from the searched area. Accordingly, two measures, improved heuristic distance estimation and the node-directed search, are taken to speed up the algorithm.

As mentioned before, in Bell (2009), the grid gap number was adopted as the heuristic variable in his experimental grid network, i.e. the heuristic distance between two adjacent nodes in the network is estimated as 1. However, it is worth mentioning that all of the links of the experimental network are of great homogeneity and their link travel times are confined between 1 and 2 so that the heuristic estimation will not surpass the real distance. Without this condition, the heuristic grid gap may generate a non-optimal hyper-path. (A similar problem might occur when implementing the A-Star algorithm and estimating the heuristic estimates in another dimension than the route guidance advice, e.g. heuristic estimate is taking distance, but the shortest path in travel time is sought.)

The usage of shortest distance as heuristic variable generally makes the algorithm faster but such efficiency improvement is far less important than confining the search set when selecting the link with minimum $(u_i + h_j + c_a)$. To confine the search set, a similar idea to the one used in the HPD algorithm is utilized. That is, the HPD uses a Dijkstra-like node label setting algorithm. However, unlike the HPD algorithm, we stick to link label setting as in the HS with the help of node label correcting at the same time. That is, after a link label is permanently set, the label of its tail node is corrected. By recording the most recently updated node the link updating will be limited to those outgoing from this node, which leads to a "node-directed" search. The concept of this node-directed search is to keep the link selection search always confined within the updated link set. The updated node of the last updating step is the node whose incoming links will be updated and added to the updated links set. The elements in the links list will be dynamically added and removed within the running process so that the searched link set will be kept low. Compared to the HPD algorithm, the DHS is generally superior because it maintains the link set directly thus the link-checking step within Step 1.1 of the HPD is unnecessary to ensure optimality of the resulted hyperpath.

By utilizing the ideas mentioned above in connection with the HS algorithm, the number of selected links will keep unchanged but an obvious speed-up can be seen when testing the algorithm. In addition, this node-directed search can also be adopted in the ASF algorithm. In that case, the theoretical time complexity will decrease to the level of the HPD algorithm resulting in a much faster run time. **Table 4** shows the pseudo code of DHS algorithm. To get a blind version of the DHS algorithm without heuristic information, i.e. a node-directed ASF algorithm, one may just cross out the with shadow underlined lines of DHS algorithm.

Table 4 Pseudo codes of DHS algorithm

Step 0: Variables initialization
$u_i \leftarrow \infty, i \in N - \{s\}, u_s \leftarrow 0;$
$f_a \leftarrow 1/d_a \; if \; d_a > 0, else \; f_a \leftarrow M;$
$f_i \leftarrow 0, i \in N;$
$y_i \leftarrow 0, i \in N - \{r\}, y_r \leftarrow 1, H \leftarrow \emptyset$
$\omega \leftarrow s, UL \leftarrow \emptyset$
Run Dijkstra algorithm, $h_i \leftarrow sp_i$*;*
Step 1: Select link a
for every link $a \in L^-(\omega)$,
$if \; a \notin S \; and \; a \notin H \; then \; UL \leftarrow UL + \{a\}$
find link a in UL with the minimum $u_{j(a)} + c_a \; +h_{i(a)}$,
$UL \leftarrow UL - \{a\}, S \leftarrow S + \{a\}$
Step 2: Update node i
$if \; u_i \geq u_j + c_a \; then$
$if \; u_i = \infty \; and \; f_i = 0 \; then \; \beta \leftarrow 1 \; else \; \beta \leftarrow f_i u_i$
$u_i \leftarrow \frac{(\beta + f_a(u_j + c_a))}{f_i + f_a}, f_i \leftarrow f_i + f_a, H \leftarrow H + \{a\}, \omega \leftarrow i$
$if \; u_j + c_a \; +h_i > u_r$ *, go to step 3 else go to step 1*
Step 3: Load
for every link $a \in L$ *in decreasing order of* $u_j + c_a \; +h_i$,
$if \; a \in H \; then \; p_a \leftarrow (f_a/f_i) y_i \; and \; y_j \leftarrow y_j + p_a \; else \; p_a = 0.$

In summary, there are two changes that contribute to improve algorithmic efficiency. One is the Dijkstra heuristic and the other is the node-directed search. Both of these improvements will not influence the resulting hyperpath . To illustrate this, the following three propositions are set.

Proposition 1. Dijkstra shortest path heuristic ensures the optimal solution.

Proof 1. The heuristic potentials have to satisfy the following inequality principle (Wagner and Willhalm, 2006):

$$h_j \leq h_i + c_a, \; \forall a = (i,j) \in L \qquad (1)$$

There are two possible situations: Either link a belongs to the shortest path sp_j or not. If link a belongs to sp_j, we will have $sp_j = sp_i + c_a$. Since that shortest path is taken when initializing the

network, we have $h_j = sp_j$ and $h_i = sp_i$ such that $h_j = h_i + c_a$. Similarly, if link a does not belong to sp_j, it turns out that $h_j < h_i + c_a$ because the path *r-i-j* is another path rather than the shortest path *r-j*. Consequently, both situations satisfy the inequality principle, which ensures the optimal solution. □

Proposition 2. The node-directed search avoids the link re-check in the HPD algorithm.

Proof 2. In the HS algorithm, the initial link set to be searched is the whole link set and the selected link will be removed in every step until termination of the algorithm. The DHS instead initializes an empty link set to be searched and allows a gradual node-directed link addition. Only the added links will be checked in every iteration. As a consequence of the reduction in u_i, some links which have been added to H_i in previous iterations may not be advantageous anymore and are removed through the link re-check in the HPD algorithm. By contrast, the DHS algorithm updates the link set directly and the labels of updated nodes are not affected by the link updating process so that u_i will not reduce in later iterations and at any time $\forall a \in H_i$ the hyperpath addition condition: $u_i \geq u_j + c_a$ is always satisfied, so that there is no need for link re-check.

Proposition 3. The DHS and HPD result in the same hyperpath.

Proof 3. The hyperpath is determined by the summation of the selected links in each step. The other processes including the node-updating step and the loading step are the same except for the UN management in the node updating step of the DHS. Consequently, if the same link will be selected by the HS and DHS algorithms in every single link selection, the resulting hyperpath will be the same.

Step 1 finds the link with minimum $(u_i + h_j + c_a)$ among the links except those that have already been selected before. From proposition 1 it is apparent that the hyperpath result will not change if shortest distances are used as heuristic information. On condition that the HS adopts the shortest distance heuristic information, the only difference between the DHS algorithm and the HS algorithm is the gradual node-directed link addition. In the DHS, the *UL* set, which excludes the links that have never been updated and are hence are also never selected in the HS due to initialized infinite u_i, is a subset of the *TL* set of HS. The *UL* is identical to the *TL* except for excluding links a that have a head node with $u_j(a) = \infty$ and therefore the same link will be selected in each iteration. Consequently, DHS and HS generate the same hyperpath. Furthermore, since both the HS and HPD algorithms generate the same hyperpath with ASF, all four algorithms generate the same hyperpath.□

4. ALGORITHM PERFORMACE TESTS

In this section, first, the same network with Bell (2009) will be tested and then the performances of ASF, HS, HPD and the proposed DHS are compared in larger synthetic networks with different topologies. All of the algorithms are coded in Microsoft C# with Visual Studio 2010 and the testing environment is Intel T6400 2.13GHz/4G RAM/Windows Vista. Only the result of tests on grid networks will be shown here for convenience because a similar result can be seen on other network topologies according to our performed tests. On the 8 by 8 grid network as used in Bell (2009), we firstly compare the number of selected links (**Fig.2**). Due to the different structure of HPD from the other three algorithms, it will not be included in this comparison.

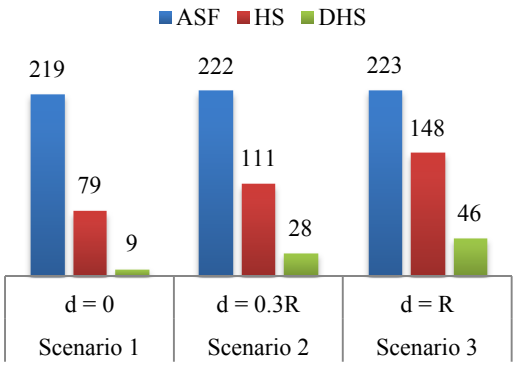

Fig.2 Comparison of the number of selected links in Bell (2009) network

In **Fig.2**, it is clear that the proposed DHS algorithm selects significantly less links during the link selection step so that we can expect a speed-up in real running time. One may argue about the extra time required to run the Dijkstra shortest path algorithm in Step 0 of the DHS algorithm and doubt whether the run time of DHS is hence in fact faster. Therefore the CPU run time is tested in the following where the time required to run the Dijkstra algorithm is included in the DHS.

To make run time differences large enough to be observable, tests are performed on a larger grid network. This larger grid network has a size of 50 by 50, leading to 2500 nodes and 9800 links. The data table format is shown in **Table 5**. The uncongested travel time was randomly generated by uniform distribution within [30, 50] and then the congested travel time was generated by adding a new random variable within [15, 25] to the uncongested travel time. The difference of congested and uncongested travel times corresponds to the maximum delay in Bell (2009).

We randomly tested the performance for different OD pairs where the tested node pairs are all randomly selected by a uniform random generator

that is available online (http://www.random.org/integers/). In **Table 6** the test scenarios are ordered by the shortest distances from the origin to the destination because it is reasonable to assume that the algorithm running time increases when the OD pairs are farther away considering that more link searches may be included. **Fig.3** illustrates the relative positions of these OD scenarios in a rectangular coordinate system, for example, node 1 as (1, 1) and node 2500 as (50, 50). Note that the codes of all algorithms are implemented with the same data handling structure and that they share the same codes as much as possible so that there will be no specific code bias for any algorithm. For convenience, the data structure we used is the adjacency list and the practical speed can be much faster by using other data structures such as heap structure. In any case, these factors have little effect on the performance comparison and the comparisons are fair. The following HS algorithm test results are all with the shortest distance information and the one with the grid gap information will not be discussed since it is slower and may cause deviated hyperpath.

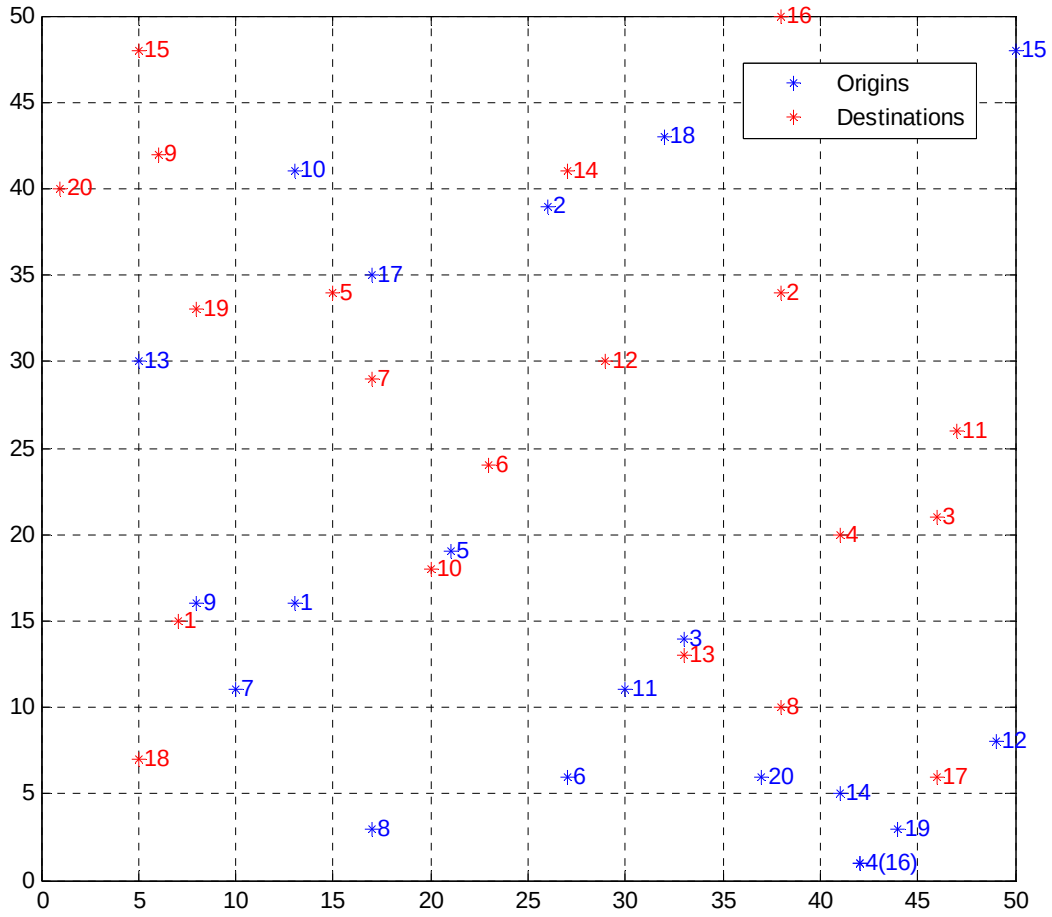

Fig.3 Relative positions of the tested OD scenarios
(The numbers are scenario ID, 4(16) means the same node for scenario 4 and 16)

The result of our test is shown in **Table 6**. In our tests, the average run time of these 20 scenarios for the ASF algorithm was 7068.1ms, 6814.9ms for the HS algorithm, 298.1ms for the HPD algorithm and only 51.3ms for the proposed DHS algorithm. It turns out that the Dijkstra running time is almost negligible compared to the total running time of DHS in the scenarios that OD pairs are far away from each other and that the total run time of DHS is similar to the speed of running the Dijkstra algorithm if the destination is, in terms of grid distance, close to the origin.

In most of the scenarios, the HS algorithm is faster than the ASF algorithm except for the last four scenarios where OD pairs are near the opposite corners of the network. This is because the heuristic information helps to direct the search to the origin. However, such a merit can improve the speed very limitedly when the OD pair scenario are in opposite corners. (Note that the link search is inversely from the destination to the origin.) Consequently, the scenario where HS algorithm is even slower means that the speed-up by heuristic information cannot compensate for the intrinsic additional run time consumption for adding the heuristics and the difference is not trivial. Considering that such

search scenarios are rare, the HS in general outperforms the ASF algorithm.

Table 6 Test scenarios and results

Scenario ID	*r* (coordinate)	*s* (coordinate)	Shortest Distance	Algorithm running time (ms)			
				ASF	HS	HPD	DHS
1	763 (13,16)	707 (7,15)	260	751	137	311	7
2	1926 (26,39)	1688 (38,34)	614	4227	1277	323	11
3	683 (33,14)	1046 (46,21)	711	4008	1727	299	12
4	42 (42,1)	991 (41,20)	731	5429	1563	287	10
5	921 (21,19)	1665 (15,34)	768	5907	1982	302	13
6	277 (27,6)	1173 (23,24)	810	7057	2172	311	14
7	510 (10,11)	1417 (17,29)	860	7540	2642	310	16
8	117 (17,3)	488 (38,10)	1002	7109	3649	285	24
9	758 (8,16)	2056 (6,42)	1019	6297	3268	297	20
10	2013 (13,41)	870 (20,18)	1058	9421	4174	313	26
11	530 (30,11)	1297 (47,26)	1104	7542	4658	281	32
12	399 (49,8)	1479 (29,30)	1482	10655	7897	310	61
13	1455 (5,30)	633 (33,13)	1580	10449	8509	312	69
14	241 (41,5)	2027 (27,41)	1735	11044	10238	289	65
15	2400 (50,48)	2355 (5,48)	1753	9912	5656	297	26
16	42 (42,1)	2488 (38,50)	1907	10804	9182	293	54
17	1717 (17,35)	296 (46,6)	1991	10432	12040	284	102
18	2132 (32,43)	305 (5,7)	2159	10532	13456	290	130
19	144 (44,3)	1608 (8,33)	2293	10826	14662	280	178
20	287 (37,6)	1951 (1,40)	2416	11340	14546	287	156
	Average			7068.1	6814.9	298.05	51.3

In **Fig.4** the performance comparison among these 4 algorithms is shown. **Fig.4** implies that as the shortest distance increases, there is a trend that also the run time increases although the relationship is not a strict one because there is no strict linear relationship between the shortest distance and the algorithm run time. In contrast to the HS and DHS algorithms, the HPD algorithm almost keeps the same run time. By analogy to the Dijkstra shortest path algorithm and the A-star algorithm, the reason is clear: Like Dijkstra shortest path algorithm, no matter where the OD pairs are, the HPD algorithm will always cover all of the nodes before termination. Furthermore, **Fig.5** shows the run time of the HS algorithm and DHS separately and one may find that the increasing trends in run time the further apart the OD pairs are in terms of shortest distance, are quite similar for both algorithms. It can be explained with the facts that both adopt the shortest distance as the node heuristic information and that the DHS algorithm is a constrained link search version of the HS algorithm.

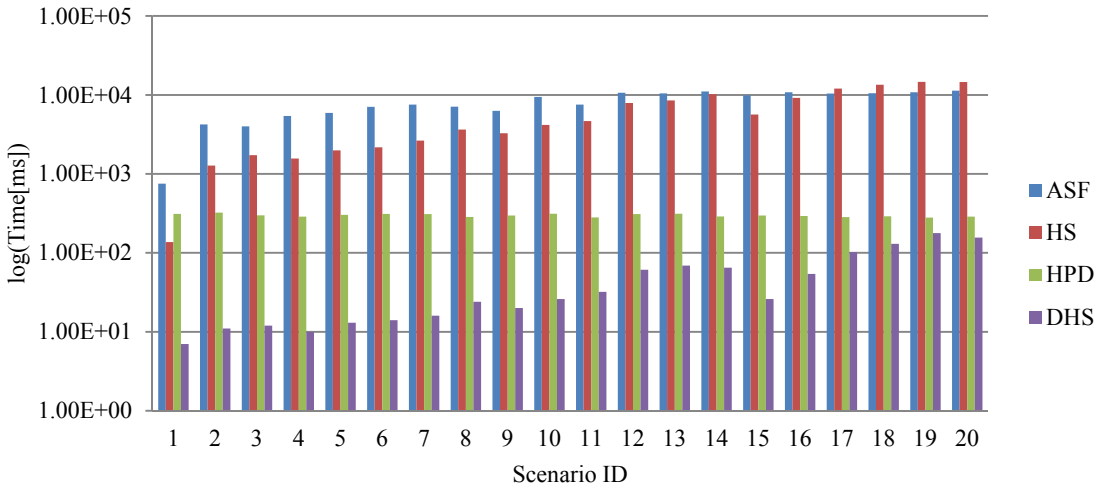

Fig.4 Performance comparison

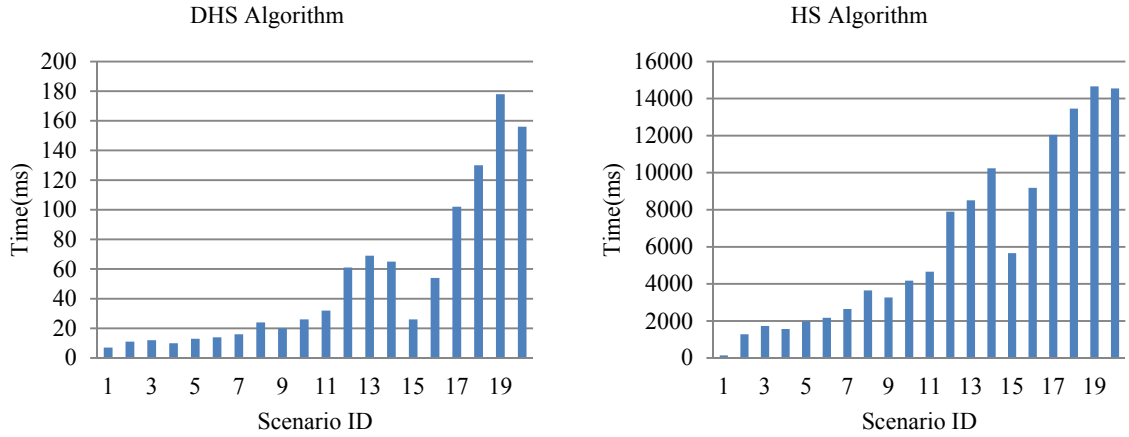

Fig.5 The trend comparison between HS and DHS algorithms

Finally, one may question whether the observed differences in run time also hold for networks with different topologies. We carried out another test on a synthetic radial network which contains one central node, 50 radial links and 50 connecting cycles (2501 nodes and 10000 links). Test results on this network also show that the proposed DHS is faster than the other algorithms, especially compared with ASF or HS algorithm. The same random OD pairs as the former tests are tested. The average run time for the ASF algorithm on the radial network is 11232 ms, 9249 ms for HS algorithm and 305 ms for the HPD algorithm. The average run time of the proposed DHS algorithm is only 21ms. The similarly increasing trend between HS and DHS also holds. The general speeds on the radial network for ASF and HS algorithms are a little slower; it may be explained mainly by the fact that the radial network increases the number of links to 10,000. The run time of the HPD algorithm is almost identical compared with that in the grid network. This is because the run time of the HPD algorithm is mainly constrained by the number of nodes and there is only one node more in the grid network. However, we may find that the run time of DHS algorithm is much faster than that on the grid network. In the radial network, the shortest distance is less than that of grid networks because the cycles shorten the distance, thus the node-directed strategy will lead to much less links to be searched. That is, it may be concluded that heuristic information and node-directed strategy contribute more to the speed-up on radial networks than on grid networks. Because the performance comparisons are similar to that of the grid network, for brevity, the detailed test results will not be shown in this paper but are available if requested.

5. DISCUSSION AND CONCLUSIONS

The multipath-routing strategy to "minmax exposure to delay" seems a reasonable assumption not only for transit passengers but also for road users. Whereas for transit passengers it is fairly obvious to assume that they fear delayed service arrivals, the precise assumption on what exactly road drivers fear is more of a topic for discussion.

The focus of this paper is on the performance of multi-path route guidance algorithms, though, and we chose to follow Bell's assumptions about the hyperpath chosen by drivers in road networks. The DHS algorithm is proposed as an approach to obtain this hyperpath faster.

The HPD algorithm has a very similar name to our DHS algorithm, the idea is different: Hyperpath-Dijkstra has a structure resembling Dijkstra's algorithm; the dominant loop is controlled by the number of solved nodes instead of solved links. Although it is impossible to compare the efficiency between HPD and DHS algorithm in theoretical time complexity because of the heuristic characteristic of the DHS algorithm, we showed the lower run time of the DHS algorithm through tests on artificial grid and radial networks. In most cases, the DHS outperformed the HPD algorithm significantly and even in the worst case it is still almost twice as fast as the HPD algorithm.

The proposed DHS algorithm is an efficiency improved version of the original Spiess & Florian algorithm inspired from the on-the-fly concept: The link set to be searched is extended on the fly by gradually finding the lastly updated node; the pre-stored node-link topology is hence well used for maintaining the updated link set dynamically. Note that this requires some more memory space resources, in terms of pre-stored node look-up tables. Since run-time improvement is believed to be crucial for route guidance applications though, our significant run time improvements mean that it is worth doing so, especially because hard disk drives are much cheaper than processors. The algorithm is still potential to be improved by adopting more complicated, but also more efficient data structures, such as Fibonacci heap if necessary.

It is worth mentioning that the speed-up methods proposed in this paper are not limited to road networks but also can be adopted for transit networks if maximum delay is interpreted as service frequency. We are not declaring a comparatively superior multipath algorithm to the K-path algorithm or other types of multipath algorithm here but only trying to improve the efficiency of the known frequency (or maximum delay) based link-to-link multipath algorithm so that it will be much more potentially eligible to be adopted in navigation fields. Besides being useful for route guidance, the proposed algorithm may also be applied within traffic assignment. In this paper, the effect of navigation to network equilibrium solutions was not studied but is a topic of further research. Further, our plan is to test the algorithm on a real network and compare the generated paths with the collected route choice data so that the applicability of this algorithm and the underlying assumptions on user behaviour to observed route choice data can be explored.

REFERENCES

1) Bell, M.G.H. (2009). Hyperstar: A multipath A-star algorithm for risk averse vehicle navigation. *Transportation Research Part B* 43(1): 97-107.
2) Ben-Akiva, M., Bierlaire, M., Bottom, J., Koutsopoulos, H. and Mishalani, R. (1997). Development of a route guidance generation system for real-time application. In Papageorgiou, M. and Pouliezos, A. (eds.), *The 8th IFAC Symposium on Transportation Systems*, Crete, Greece, June 1997.
3) Chen, Y.Y., Bell, M.G.H. and Wang, D. (2006). Risk-averse time dependent route navigation by a constrained dynamic A* search in decenterized structure. *Proceedings of the Transportation Research Board Annual Meeting 2006*. Washington D.C., USA.
4) Chen, Y.Y, Bell, M.G.H. and Bogenberger, K. (2007). Reliable pre-trip multipath planning and dynamic adaptation for a centralized road navigation system. *IEEE Transactions on Intelligent Transportation Systems* 8(1): 14-20.
5) Cominetti, R., Correa, J. (2001). Common bus lines and passenger assignment in congested transit networks. *Transportation Science* 9: 115-121.
6) Cormen, T., Ch. Leiserson, R. Rivest. (1991). *Introduction to Algorithms*. McGraw-Hill, MIT Press, Boston, MA.
7) Dijkstra, E. (1959). A note on two problems in connection with graphs. *Numerical Mathematics* 1: 395-412.
8) Eppstein, D. (1994). Finding the k shortest paths. *The 35th IEEE Symposium Foundations of Computer Science*: 154-165.
9) Eppstein, D. (1998). Finding the k shortest paths. *Technical Report CA92717*, Department of Information and Computer Science, University of California, Irvine.
10) Fosgerau, M., Frejinger, E. and Karlström, A. (2009). Route choice modeling without route choice. *Proceedings of European Transport Conference*.
11) Hart, P.E.; Nilsson, N. J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4* 4 (2): 100-107.
12) Hofmann-Wellenhof, B., Legat K. and Wieser M. (2003). *Navigation: principles of positioning and guidance*. Springer Verlag.
13) Jiménez, V.M. and Marzal, A. (2003). A lazy version of Eppstein's k shortest paths algorithm, *Proceedings of the 2nd international conference on Experimental and efficient algorithms*: 179-191, Ascona, Switzerland.
14) Kaparias, I. and Bell, M.G.H. (2009). Testing a reliable in-vehicle navigation algorithm in the field. *Intelligent Transport Systems* 3(3): 314-324.
15) Nguyen, S. and Pallottino, S. (1988). Hyperpaths and shortest hyperpaths, *Lectures given at the third session of the Centro Internazionale Matematico Estivo (C.I.M.E.) on Combinatorial optimization*: 258-271, Como, Italy
16) Pearl, J. (1984). Heuristics: *intelligent search strategies for computer problem solving*, Addison-Wesley Longman Publishing Co., Inc.
17) Santos, J.L.K. (2006) Shortest Path Algorithms [EB/OL]. *http://www.dis.uniroma1.it/~challenge9/papers/santos.pdf.*
18) Spiess, H. and Florian, M. (1989). Optimal strategies: A new assignment model for transit networks. *Transportation*

Research Part B 23(2): 83-102.
19) Wagner, D., Willhalm, T. (2006). Speed-up techniques for shortest path computations. *ARRIVAL-TR-0024* (<http://arrival.cti.gr/>).
20) Wichmann, D.R. and Wuensche, B.C. (2004). Automated route finding on digital terrains. *Proceedings of IVCNZ '04: 5-9*, Akaroa, New Zealand.
21) Yongtaek, L. and Hyunmyung, K. (2005) A shortest path algorithm for real road network based on path overlap. *Journal of the Eastern Asia Society for Transportation Studies* 6: 1426- 1438.
22) Zijpp, N.J. and Catalano, S.F. (2005). Path enumeration by finding the constrained *K*-shortest paths. *Transportation Research Part B* 39(6): 545-563.