J Supercomput (2009) 49: 234–247
DOI 10.1007/s11227-008-0236-7

Hybrid shortest path algorithm for vehicle navigation

Hsun-Jung Cho · Chien-Lun Lan

Published online: 12 December 2008

Abstract Vehicle navigation is one of the important applications of the single-source single-target shortest path algorithm. This application frequently involves large scale networks with limited computing power and memory space. In this study, several heuristic concepts, including hierarchical, bidirectional, and A*, are combined and used to develop hybrid algorithms that reduce searching space, improve searching speed, and provide the shortest path that closely resembles the behavior of most road users. The proposed algorithms are demonstrated on a real network consisting 374,520 nodes and 502,485 links. The network is preprocessed and separated into two connected subnetworks. The upper layer of network is constructed with high mobility links, while the lower layer comprises high accessibility links. The proposed hybrid algorithms are implemented on both PC and hand-held platforms. Experiments show a significant acceleration compared to the Dijkstra and A* algorithm. Memory consumption of the hybrid algorithm is also considerably less than traditional algorithms. Results of this study showed the hybrid algorithms have an advantage over the traditional algorithm for vehicle navigation systems.

Keywords Shortest path algorithm · Heuristics · Hierarchical network

1 Introduction

This study considers a single-source single-target shortest path problem (also known as the point-to-point shortest path problem, P2P) involving a large scale static network. Some network preprocessing is introduced to accelerate the query time. The problem is especially interesting owing to its important role in numerous real world

H.-J. Cho (✉) · C.-L. Lan
Department of Transportation, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu, Taiwan
e-mail: hjcho@cc.nctu.edu.tw

problems, i.e., transportation, communication networks, and some combinatorial models. A typical transportation application of the shortest path problem is the vehicle routing problem. With the maturation of Global Positioning System (GPS) and Geographic Information System (GIS) technology, the vehicle navigation system has gradually attracted attention. During operation of the in-vehicle navigation system, a recommended path should be identified within a short period (a few seconds); with limited computing power, little memory space, and a large network.

The Dijkstra algorithm with a Fibonacci heap data structure remains by far the fastest known algorithm for the general case of arbitrary nonnegative link cost networks. Although some researchers have developed algorithms with average linear time, but the limitation makes it unsuitable for road network [1]. However, the traditional optimal shortest path algorithms, while remaining computationally intensive, do not meet the needs of vehicle navigation.

Traffic networks remain relatively constant over time, and thus preprocessing can be done to improve the query time. The problem itself becomes a trade-off between storage space and computation time. Although the precomputed shortest path can provide near constant query time, but the $O(n^2)$ storage space prevents the real traffic networks with node numbers exceeding hundreds of thousands from doing the precomputation. Some researchers reduced the search space of the Dijkstra algorithm to $50 \sim 10\%$ of the origin search space by using precomputed information that can be stored in $O(n+m)$ space [2]. While facing the on-board unit of vehicle navigation equipment, the storage space and computing power is more limited; it becomes necessary to introduce an even faster algorithm.

While dealing with a traffic network, a person can rapidly identify a reasonably short path between two arbitrarily selected nodes. Although this path may not be the actual shortest path, it is a good alternative. Humans treat networks hierarchically, and the hierarchical concept can be introduced in an effort to solve the shortest path for navigation systems. The contribution of this paper lies in combining the hierarchical concept and other techniques to establish a suitable algorithm for the on-board vehicle navigation system. The algorithm developed with this combined approach has the potential to be several thousand times faster than the traditional Dijkstra algorithm. We begin, in Sect. 2, with the preliminary discussion of shortest path algorithms. The hybrid shortest path algorithm is proposed in Sect. 3, while its implementation results are discussed in Sect. 4. Conclusions are discussed in Sect. 5.

2 Preliminaries

2.1 Graphs

A weighted directed graph $G = (N, A, c)$ comprising a finite set of nodes N, arc set $A \subseteq N \times N$, and a real-valued nonnegative weight function $c : A \rightarrow \mathbf{R} \geq 0$ is considered in this study. The weight of arc $c(u, v) \in A$ with start node v and end node u is denoted as $c(u, v)$, if $(u, v) \notin A$, $c(u, v) = \infty$. Throughout this study, the number of nodes $|N|$ is denoted by n and the number of arcs $|A|$ is denoted by m. The arc cost represents the impedance of a vehicle traveling through that arc, and can generally be described as arc length (actual length) or travel time. A graph is considered sparse, if the number of arcs m is in $O(n^2)$; and is considered large if it is only possible to afford a memory that is linear in the size of graph $O(n+m)$. A path from an origin (s) to destination (t) is defined as a sequential list of links: $(s,i),\dots,(j,t)$ and the path cost is the total costs of the individual links.

2.2 Shortest path algorithms

Given a distinguished source node $s \in N$ and destination node $t \in N$, the single-source single-target shortest path problem is to derive the weight of the least weight path from s to t. Traditional algorithms, also known as optimal algorithms, have been studied for over 40 years in various categories, such as transportation and computer science. The majority of traditional shortest path algorithms are essentially applications of dynamic programming, and the optimal shortest path was identified via a recursive decision-making.

The Dijkstra algorithm is one of the best known shortest path algorithms. This algorithm is a width-first search method proposed by Dijkstra in 1959, and is based on searching from a node and gradually expending the search space to neighboring links [3]. For each node, there is a super distance $D(v) \geq d(v)$; the super distance $D(v)$ the cost of current evaluated path from s to v, and the calculated minimum distance between s and v is denoted as $d(v)$. A set $S \subseteq N$ is introduced, such that $\forall v \in S : D(v) = d(v)$ and $\forall v \notin S : D(v) = \min_{u \in S}\{d(u) + c(u,v)\}$. The algorithm begins with $S = \{s\}$, $D(s) = d(s) = 0$, and $\forall v \neq s : D(v) = c(s,v)$. A link relaxation method, selecting neighbor nodes to the node v with minimum $D(v)$, is then repeated to obtain the shortest path from the origin node to each expended node. Consequently, $\forall (u,v) \in A$, if $D(u) + c(u,v) < D(v)$ then $D(v) = D(u) + c(u,v)$. The algorithm stops when $t \in S$. Several related studies focused on designing efficient node structures, e.g., Dijkstra's bucket and Dijkstra's heap [4–6]. The algorithm time complexity varies with data structure. As for the linear array, the complexity is $O(n^2 + m)$; moreover, with a binary and Fibonacci heap, the complexity is, respectively, $O((n+m)\log n)$ and $O(m + n\log n)$. These data structures represent an improvement over the original linear array [7, 8].

The Bellman–Ford algorithm has an advantage over the Dijkstra algorithm in dealing with links that have negative costs, but remains unable to deal with networks incorporating negative loops. The Bellman–Ford algorithm also utilizes a link relaxation method to determine the shortest path between nodes [9]. This algorithm begins from any node in the network, and after kth iterations, the algorithm is able to identify the shortest path between the origin node and every node located within k links. The complexity of the Bellman–Ford algorithm is $O(mn)$; it is worse than the Dijkstra algorithm and has even greater difficulty coping with large networks.

2.3 Heuristics

The traditional optimal shortest path algorithm is computationally too intensive for real-time applications such as the in-vehicle route guidance system. This inefficiency comes from the fact that it does not utilize any prior information contained in the network structure, e.g., the location of origin node and destination node. If the origin node is located in the center of the network and the destination node is located on the north side of the network, then intuitively the area between these nodes should be searched first. Heuristic algorithms can be classified into two categories: (i) limiting search space and (ii) search problem decomposition.

2.3.1 Limiting search space—A algorithm*

The traditional optimal search algorithms, e.g., Dijkstra, examine all possible nodes from the origin to the destination node without considering the possibility that they will be included in the shortest path. If the algorithm utilizes the node location information and limits the search to a certain area, the search space can be reduced and the search performance can be improved [10].

The A* algorithm, proposed by Hart, Nilsson, and Raphael, is one of the most effective methods of utilizing the limiting search space idea. A* algorithm is a best-first search algorithm with a heuristic evaluating function $f(v) = g(v) + h(v)$. The $f(v)$ denotes the label of node v, $g(v)$ represents the cost of current evaluated path from origin node to node v, and $h(v) : A \to \mathbf{R}$ is the estimated cost from node v to destination node t. The $h(v)$ is important in these heuristics, and has the following characteristics.

1. If the $h(v)$ equals 0, then the A* algorithm performs as Dijkstra.
2. If the $h(v)$ is less than the actual distance between node v and the destination node, then the A* algorithm is guaranteed of finding the optimal shortest path. The search space is increased with decreasing $h(v)$.
3. If the $h(v)$ is larger than the actual distance between node v and the destination node, then the A* is not guaranteed to find the optimal shortest path. Computational speed increases with $h(v)$.

Let $h(v)$ and $h'(v)$ denote two estimation functions that $h(t) = h'(t) = 0$ and $h'(v) \geq h(v), \forall v \in N$. The set of nodes scanned by A* algorithm using $h'(v)$ is contained in the set of nodes scanned by A* search using $h(v)$ [11]. A* algorithm utilizes the same search procedure as Dijkstra, except $D(v)$ is now replaced with $f(v)$.

In the worst case, the time complexity of A* algorithm is exponential. Regarding the application of vehicle navigation, the estimated cost $h(v)$ can be calculated based on the Euclidian distance between node v and the destination node. Based on this estimated cost, the search space of the A* algorithm tends to follow the same direction as the destination. The A* algorithm is the most efficient algorithm that guarantees the identification of an optimal solution. A study empirically found that the A* algorithm explores less than 10% of the nodes expanded by the Dijkstra algorithm [12]. Some researchers demonstrated that A* algorithm identifies the shortest path in many Euclidean graphs with an average polynomial computational complexity [13]. Owing to its performance and accuracy, the A* algorithm is the most popular algorithm currently implemented in the vehicle navigation system. Figure 1(a) compares the search space with the Dijkstra algorithm.

2.3.2 *Limiting search space—hierarchical concept*

The hierarchical concept is known as an abstraction problem solving strategy. The basic idea is to first focus on the essential features of a complex problem with the details being completed later. Uchida et al. demonstrated that drivers prefer to select a path based on the connectivity, familiarity, and road types rather than selecting the optimal shortest path [14]. Huang and Jin proposed a hierarchical search method combined with a pre-calculated shortest path [15]. The study of Huang and Jin first divided a large network into several separated subnetworks and stored the all-to-all shortest path within the subnetworks. During the operation period, the shortest path can be derived using a look-up table. Though the speed is very fast, the in-vehicle navigation is not able to afford such memory requirements. Car and Frank studied a series of small networks and found that the hierarchical algorithm is twice as fast as a non-hierarchical algorithm. However, they also found that the paths generated by the hierarchical algorithm are an average of 50% longer than the optimal paths [16]. Jagadeesh et al. proposed a pre-check to ensure that the origin and destination nodes are not located in the same subgrid or two adjacent grids; otherwise, a nonhierarchical shortest path algorithm is being introduced [17]. The computational acceleration results from a hierarchical algorithm and are a function of network topology, search rules incorporated, and trip length [18].

2.3.3 *Search problem decomposition—bi-directional search method*

Most traditional search methods are uni-directional, meaning these search methods expend the search space from the origin node to the destination nodes. Dantzig proposed a search procedure that divided the problem into two separate procedures [19]. One search proceeds forward from the initial stage, while the other proceeds backward from the goal stage. The bidirectional algorithm is demonstrated to be slower than the A* algorithm [20]. Another bidirectional search method proposed by researchers, the search terminates when two A* search trees intersect at a given number of frontier nodes [21]. The performance and accuracy is controlled by the parameter of intersected number of nodes. Figure 1(b) compares the search space comparison with the Dijkstra algorithm.

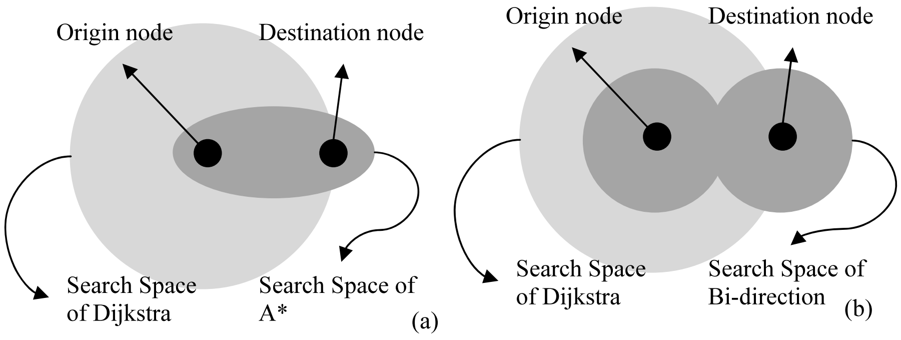


Fig. 1 The search space comparison among Dijkstra, (**a**) A* and (**b**) bidirection

3 Hybrid shortest path algorithm

This section combines several algorithms mentioned above, namely the Dijkstra, A*, hierarchical and bidirectional concepts.

3.1 Implementation of hierarchical and bi-directional concepts

In this research, the network is divided into two connected layers. The upper layer comprises high mobility links, for example, freeway and expressway; moreover, the lower layer comprises high accessibility links, for example, urban and rural networks. Pseudo nodes are created to connect these separated networks, and these nodes enable entering or exiting of the separated networks, and thus these nodes are labeled entry/exiting nodes. Not all entry nodes are equivalent to exiting nodes. For example, some freeway junctions have only exit or entrance. The characteristic of entry/exiting nodes should depend on the real network.

This study combines the bidirectional concept and the hierarchical concept, and crates a bidirectional hierarchical concept demonstrated in Fig. 2. This research combines the above concepts and suggests the following solution procedure.

1. Compute the shortest path between origin node and nearest entry node on the lower layer.
2. Compute the shortest path between destination node and nearest exiting node on the lower layer.
3. Compute the shortest path between the entry node and exiting node on the upper layer.

3.2 Integration of existing shortest path algorithm

The solution procedure proposed above involves several shortest path computations. These computations use traditional algorithms, including the Dijkstra algorithm and the A* algorithm. The Dijkstra algorithm, being recognized as the best well-known

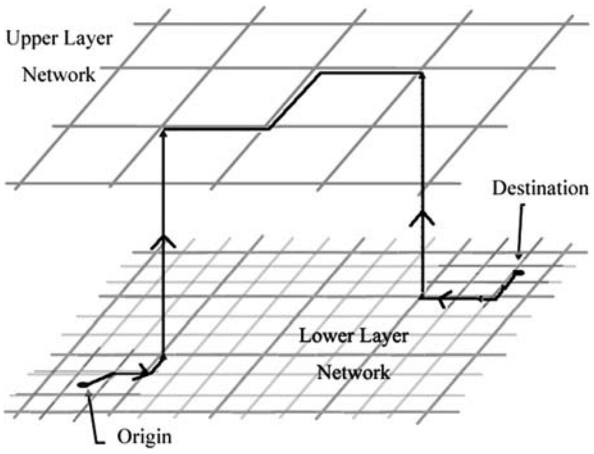


Fig. 2 Bidirectional hierarchical concept

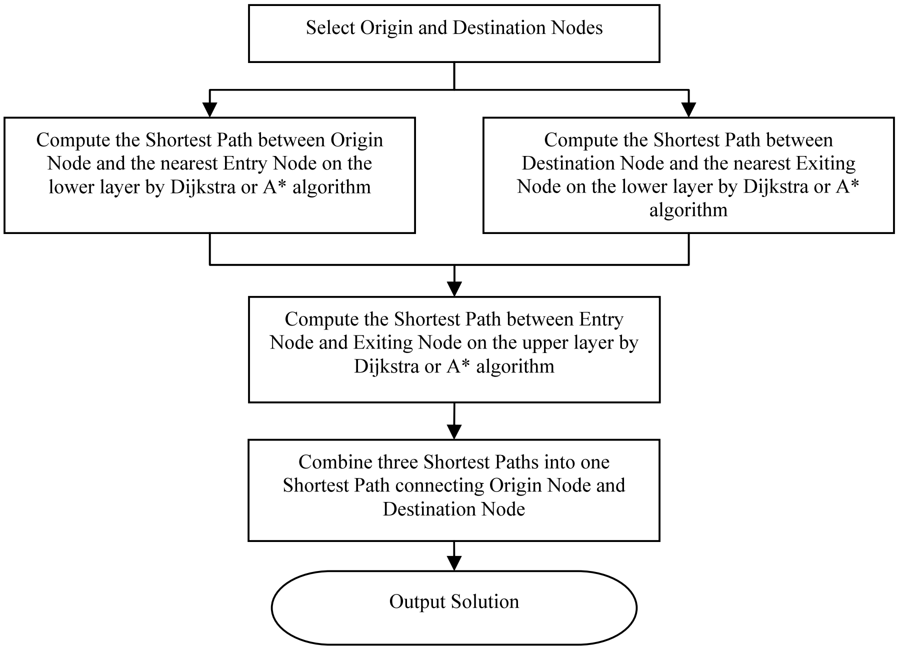

Select Origin and Destination Nodes
Compute the Shortest Path between Origin Node and the nearest Entry Node on the lower layer by Dijkstra or A* algorithm
Compute the Shortest Path between Destination Node and the nearest Exiting Node on the lower layer by Dijkstra or A* algorithm
Compute the Shortest Path between Entry Node and Exiting Node on the upper layer by Dijkstra or A* algorithm
Combine three Shortest Paths into one Shortest Path connecting Origin Node and Destination Node
Output Solution

Fig. 3 Solution framework of hybrid shortest path algorithm

shortest path algorithm that provides actual shortest path, is selected to evaluate the speed-up and bias caused by bidirectional hierarchical concept suggested in this research. The A* algorithm is then introduced in this integration to improve performance and reduce memory consumption.

Three hybrid algorithms are thus suggested. First, the D-D-D algorithm suggests that the Dijkstra algorithm is used for all shortest path computations including the lower layer and upper layers. Second, the A-A-A algorithm is based on the same idea as the D-D-D algorithm, and uses the A* algorithm in both the lower and upper layers. Third, the A-D-A algorithm is introduced because the upper layer includes less links and nodes, and thus a more precision algorithm (the Dijkstra algorithm) might be used without causing too much computational time. Figure 3 shows the solution framework of the hybrid shortest path algorithm.

The hybrid shortest may not always represent the optimal solution, but it should be close to optimal, and moreover should be closer in terms of time than in terms of distance. The result is owing to the hybrid algorithm utilizing the hierarchical concept that deducts the searching space well. When calculating the time shortest path, although this algorithm might obtain a longer actual distance on high layer, the speed on that layer is considerably faster than that on low layer, and the traveling time is less. Consequently, the error between the solution obtained using this hybrid algorithm and the optimal solution in terms of time may be less than for the shortest path in terms of distance

3.3 Direct access data structure

The method used to access stored network data can be either sequential access or direct access. The direct access method implies that data is stored at a known address, and has better performance than the sequential access method where a search is necessary to find where the data is stored. Searching algorithms are time consuming. Binary search, a fast and popular search algorithm, takes $O(2\log_2 n)$, while direct access takes only $O(1)$.

Extensive node search is performed during the node selection stage of the shortest path algorithm. If the relationship between links and nodes is stored in matrix form, where the row subscript represents the node id, the column subscript represents the link id, and the cost is the matrix element. The node can be directly accessed by the subscripts without searching, improving the speed of node selecting stage. However, since the traffic network is sparse, the matrix storage method is extremely wasteful. A crucial problem is how to simultaneously maintain the speed of direct access and minimal storage space. To achieve this, the links were first sorted and assigned a continuous id. These id numbers were then be stored immediately following each node structure according to the network structure; the id of the neighboring node was also stored. During the computation stage, when a node is selected, the connected link id was also being obtained; link information could then be directly accessed via the link id; and neighbor nodes that connected to the selected node can also be derived.

Most nodes in real traffic networks were connected to three or four links; it was extremely rare for there to be more than six connections. The increase in storage space compared to the forward star structure was very limited, but the speed improvement was significant.

4 Implementation and discussion

The proposed hybrid algorithms are implementing in C++ programming language to illustrate the execution performance. Both PC and hand-held platform are used in this implementation. In the PC platform, the hybrid algorithm is demonstrated with MS C++ compiler and optimizing option “-o3” on an Intel Pentium 4 2.4 GHz processor with 2 GB memory running MS Windows XP. In the hand-held platform, the hybrid algorithm is demonstrated with MS eVC compiler on an Intel PXA 255 processor with 64 MB memory running MS Windows CE 4.2. The test network is the highway network of the Taiwan area, shown in Fig. 4(a), derived from the Institute of Transportation. This test network consists of 374,520 nodes and 502,485 links and lengths of the links are stored as double precision floating points.

Two network separation schemes are involved in this implementation. First, separate the freeway and expressway to the upper layer while keeping other links in the lower layer, as illustrated in Fig 4(b). Second, separate the freeway, expressway, and arterial highway to the upper layer while keeping other links in the lower layer, as illustrated in Fig. 4(c). In the first situation, 3,511 links are included in the upper layer; as for the second situation, 24,330 links are included in the upper layer.

In this numerical implementation, origin-destination pairs for shortest path computation are randomly chosen. The pairs are distributed into two distance ranges:

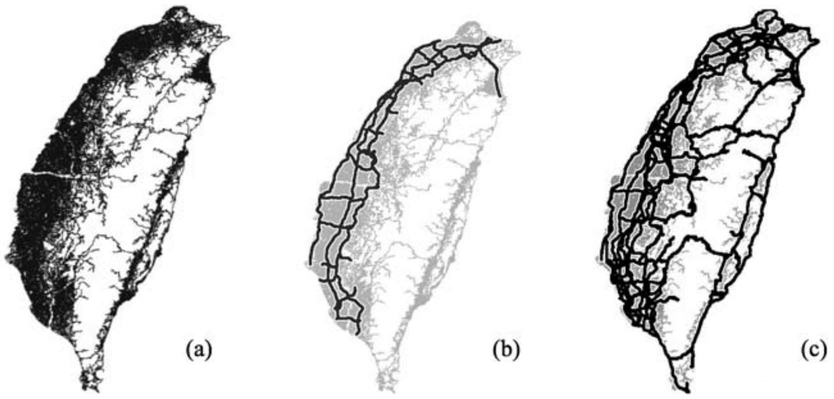


Fig. 4 (**a**) Taiwan area highway network, (**b**) freeway and expressway highlighted, and (**c**) freeway, expressway, and arterial highway highlighted

170 km (Taipei–Taichung) and 400 km (Taipei–Kaohsiung). The path costs in this implementation include: (i) actual link length and (ii) link travel time (estimated by dividing the link length by the link speed limit). Hybrid shortest path algorithms proposed in this research, including D-D-D, A-D-A, and A-A-A, are compared with the Dijkstra and A* algorithm. The comparison involving (i) the computing time, (2) the cost of computed shortest path, and (3) the size of shortest path search space explored (the main factor of memory usage).

4.1 Implementation results on PC platform

The implementations on PC platform mainly focused on the network of separating freeway and expressway to upper layer, as demonstrated in Fig. 4(b). The other separation scheme is also performed, but it shows a significant draw back on speed with nearly unchanged solution correctness. Hence, the test network addressed here mainly focused on including freeway and expressway in the upper layer.

4.1.1 Implementation results—treat link distance as link cost

Link costs in this subsection corresponds to the link distance. In this test, 60 origin-destination pairs are randomly generated in two distance ranges: 30 pairs for 170 km range and 30 pairs for 400 km range. Dijkstra algorithm denotes performance based through the test. In 170 km range, the average speed-up of each algorithm compared to Dijkstra algorithm is as follows. The A* algorithm is twice as fast, the D-D-D algorithm is about 7,600 times faster, the A-D-A algorithm is 10,800 times faster, and the A-A-A algorithm is 14,600 times faster. Furthermore, as for the 400 km range, the A* algorithms are approximately 1.75 times faster, the D-D-D algorithm is 24,000 times faster, the A-D-A algorithm is 27,700 times faster, and the A-A-A algorithm is 39,400 times faster. Comparison of 30 random generated origin-destination pairs are shown in Fig. 5; the computing time is in logarithmic scale.

The error mentioned here is the length increase related to the actual shortest distance path. For 170 km range, A* algorithm has the same solution as Dijkstra, D-D-D

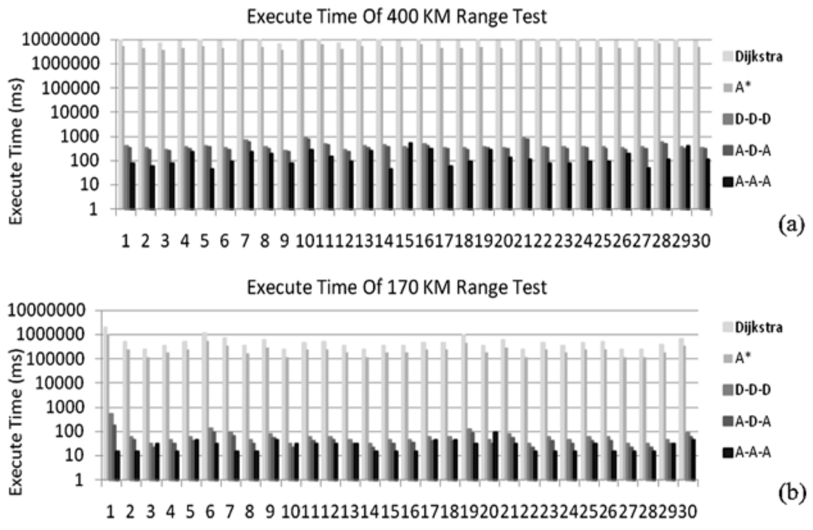


Fig. 5 Execute time comparison of (**a**) 400 km range and (**b**) 170 km range

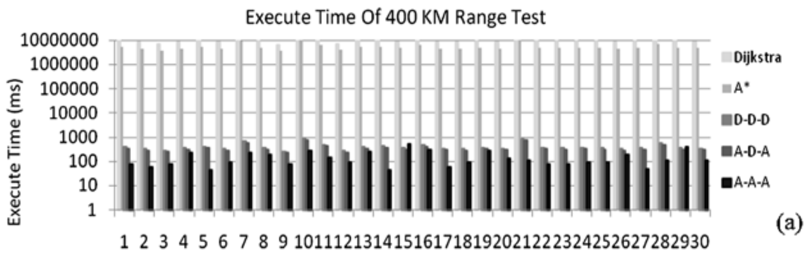
has 18% of bias, A-D-A and A-A-A have the same result of 9% bias. As for 400 km range, A* has no bias while D-D-D has 4.5%, A-D-A and A-A-A have 3.4%.

The reason for larger bias with D-D-D lies on search directions. The search direction of Dijkstra radiates from the origin with no preference, while that of A* radiates from origin and toward destination. With the characteristic of different search direction, D-D-D tends to find the "nearest entry node" and probably finds an entry node located in the opposite direction to the destination node; while A-D-A and A-A-A tend to find an entry node located in the same direction to the destination. That makes A-D-A and A-A-A to have less bias compared to D-D-D.

4.1.2 Implementation results—link travel time as link cost

This subsection conducts tests via the same procedure as the previous subsection, but the link costs is altered from link actual distance to link travel time. Link travel time is estimated by dividing link distance to link speed limit. In this test, 60 origin-destination pairs are randomly generated in two distance ranges: 30 pairs for 170 km range and 30 pairs for 400 km range. Dijkstra algorithm denotes performance based through the test. In 170 km range, the average speed-up of each algorithm compared to Dijkstra algorithm is as follows. The A* algorithm is 1.75 times as fast, the D-D-D algorithm is about 21,900 times faster, the A-D-A algorithm is 25,500 times faster, and the A-A-A algorithm is 33,400 times faster. Furthermore, as for the 400 km range, the A* algorithms approximately 1.86 times faster, the D-D-D algorithm is 23,000 times faster, the A-D-A algorithm is 37,000 times faster, and the A-A-A algorithm is 78,700 times faster. Comparison of the random generated origin-destination pairs are shown in Fig. 6; the computing time is in logarithmic scale.

The error mentioned here is the length increase related to the actual time shortest path. For 170 km range, A* algorithm is 2% longer than Dijkstra, D-D-D has 1.7%



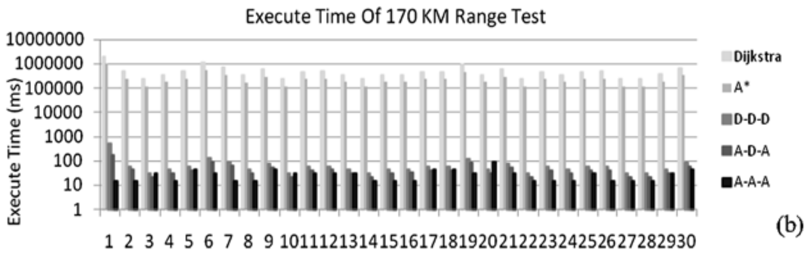


Fig. 6 Execute time comparison of (**a**) 400 km range and (**b**) 170 km range

of bias, A-D-A and A-A-A have the same result of 5.2% bias. As for 400 km range, A* has 1.2% bias while D-D-D has 0.72%, A-D-A and A-A-A have 2.73%.

Compared to the previous subsection where we treat the actual distance as link cost, the bias of the hybrid algorithm is much smaller. The main reason of this result lies on the path generated by the hybrid algorithm prefers the high mobility link (higher speed limit compare to local roads). Although the local roads might have shorter distance, but when the speed is taken into account, major roads would have shorter travel time. D-D-D performs best in this test due to the characteristic of Dijkstra. In this test, link costs are treated as link travel time, and Dijkstra will seek for the nearest entry node (in the sense of time) and keep the path on the freeway/expressway until it reaches the nearest junction to the destination (also in the sense of time).

4.2 Implementation results on hand-held platform

Commercial navigation systems usually have less computing power and memory than PCs. To illustrate the feasibility of the proposed algorithm, implementations are done on a hand-held platform. In the test platform, a hand-held device with Windows CE 4.2 running on Intel PXA 255 processor and 64 MB memory is used. The execute time for each platform is shown in Fig. 7, using a logarithm scale.

Figure 7 shows the comparison of a PC and hand-held platform with 30 random generated origin destination pairs distributed on 400 km and 170 km distance ranges. In the above comparison, the PC platform is about 100 times faster than the hand-held platform. The average computing time on a hand-held platform is 11.5 seconds for A-A-A on 400 km range, 14.6 seconds for D-D-D on 400 km range, 4.1 seconds for A-A-A on 170 km range, and 6.4 seconds for D-D-D on 170 km range, respectively.

Besides the computing power, memory consumption is another important issue for implementation on a hand-held platform. Most commercial hand-held platforms

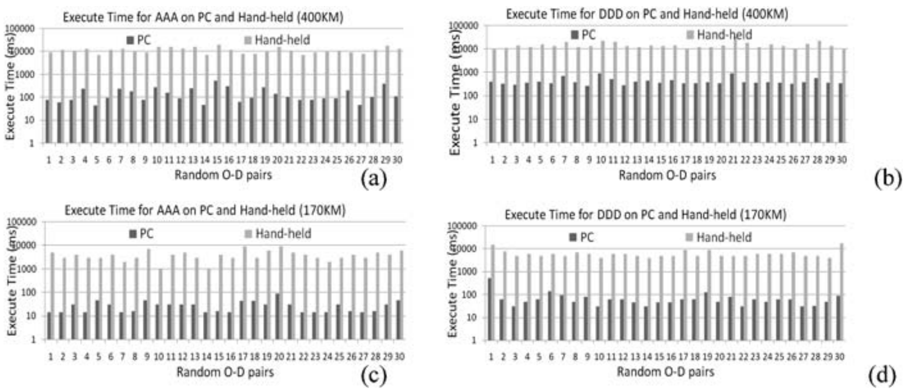


Fig. 7 Execute time comparison of the PC and hand-held platform on (**a**) 400 km range with A-A-A, (**b**) 400 km range with D-D-D, (**c**) 170 km range with A-A-A, and (**d**) 170 km range with D-D-D

have memory between 32 to 128 MB. If the algorithm increases its speed by too much sacrifice to memory usage, it would not be suitable for real-world implementation. Memory consumed in this algorithm mainly depends on the lists of "calculated nodes" and "candidate nodes." In the above random generated test, the average memory consumption of the A-A-A on 400 km range is 491.8 kB, the D-D-D on 400 km range is 688 kB, the A-A-A on 170 km range is 261.5 kB, and the D-D-D on 170 km range is 407.6 kB, respectively. The maximum memory consumed among all the tests is 1,250 kB, and it is still far below the memory limitation of most devices.

The implementation result indicates the system resources (including computing time and memory consumption) required for the proposed algorithm is affordable to most of the hand-held devices.

4.3 Implementation results—summary

With the above implementation, it is clear to see the efficiency of hybrid algorithm over Djikstra and A* algorithms. Speed-up is especially significant in long distance situation. The Dijkstra algorithm denotes the comparison base; results obtained using different algorithms are demonstrated in Table 1.

The error in Table 1 represents bias from the actual shortest path. Actual shortest path is the nearest way to the destination, but it might not be a sensible solution. This shortest path would contain some minor alleys or tiny shortcuts, which is shorter but troublesome for drivers. Paths generated by hybrid algorithms have the characteristic of getting to major roads (freeway/expressway) as soon as possible, and keep going until the nearest exit to destination. This kind of solution is similar to actual driving behavior. Although using the nearest junction might have longer travel distance on the freeway, the speed of the freeway is much faster than the local road. Hence, when travel time is treated as link cost, there is smaller bias.

The noticeable speed-up is mainly related to the reduction of search space. The hybrid algorithm utilizes a hierarchical concept that is similar to what most drivers do; it separates the network into different layers with the criteria of mobility and accessibility. The search process starts from the origin node to the nearest entry node

Table 1 Test result for shortest path algorithms

Link cost	Actual distance				Travel time			
	400 km range		170 km range		400 km range		170 km range	
	Seed-up	Error	Speed-up	Error	Speed-up	Error	Seed-up	Error
Dijkstra	1	0%	1	0%	1	0%	1	0%
A*	1.75	0%	2	0%	1.86	1.2%	1.75	2%
D-D-D	24,000	4.5%	7,600	18.1%	23,000	0.7%	21,900	1.7%
A-D-A	27,700	3.4%	10,800	9.2%	37,000	2.7%	25,500	5.2%
A-A-A	39,400	3.4%	14,600	9.2%	78,700	2.7%	33,400	5.2%

Table 2 Average search space and memory consumption of different algorithm

Link cost	Actual distance				Travel time			
	400 km range		170 km range		400 km range		170 km range	
	Searched nodes	Memory (kB)	Searched nodes	Memory (kB)	Searched nodes	Memory (kB)	Searched nodes	Memory (kB)
Dijkstra	340,388	21,054	155,143	9,704	314,022	19,644	150,998	9,457
A*	250,144	15,652	99,374	6,237	171,766	10,759	86,263	5,983
D-D-D	8,556	688	4,929	408	8,012	666	4,673	387
A-D-A	7,145	596	4,366	368	6,911	578	4,203	357
A-A-A	6,010	492	3,278	262	5,963	502	3,201	270

of the upper layer. The process continues by searching the nearest exiting node to the destination node. After the identification of entry and exiting nodes, the search process will be limited to the upper layer of network. The upper layer being used in this implementation contains only 3,511 links; compared to the total network, it is significantly reduced.

Memory consumed in shortest path algorithms mainly depend on the lists of "calculated nodes" and "candidate nodes." With the reduction of search space, memory consumption is also reduced. Average search space and memory consumption of different algorithms are compared in Table 2.

5 Conclusion

This study proposed hybrid shortest path algorithms including D-D-D, A-D-A, and A-A-A. The hybrid algorithms integrate Dijkstra, A*, hierarchical, and bidirectional concepts. It reduces the search space and accelerates the searching process. The proposed hybrid algorithms are implemented on real road network with more than 370,000 nodes and 500,000 links. The network is separated into two layers with the criteria of mobility and accessibility. The implementation results show a significant performance improvement compared to traditional algorithms. In the PC platform, recommended paths from end to end of the network can be calculated within a second; while in a hand-held platform, it can be generated in around 10 seconds. Memory consumption for the proposed algorithm is also lowered. Compared to nearly 20 MB for the Dijkstra algorithm, the proposed algorithm uses no more than 1.5 MB among all tests. The test results demonstrate a significant advantage for implementing the proposed algorithms in vehicle navigation systems.

Acknowledgements The authors would like to thank the National Science Council of the Republic of China, Taiwan, for financially supporting this research under Contract No. NSC 95-2221-E-009-346.

References

1. Meyer U (2001) Single-source shortest-paths on arbitrary directed graphs in linear average-case time. In: Proceeding of 12th annual ACM-SIAM symposium on discrete algorithms, 2001, pp 797–806
2. Wagner D, Willhalm T (2003) Geometric speed-up techniques for finding shortest paths in large sparse graphs. Lect Notes Comput Sci 2832:776–787
3. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271. doi:10.1007/BF01386390
4. Whiting PD, Hillier JA (1960) A method for finding the shortest route through a road network. Oper Res Q 11:37–40
5. Dail BR (1969) Algorithm 360:shortest path forest with topological ordering. Commun Assoc Comput Mach 12:632–633
6. Bellman R (1958) On a routing problem. Q Appl Math 16:87–90
7. Fredman ML, Tarjan RE (1987) Fibonacci heaps and their uses in improved network optimization algorithms. J ACM 34:596–615. doi:10.1145/28869.28874
8. Karimi HA (1996) Real-time optimal-route computation: a heuristic approach. Intel Transp Syst J 3:111–127. doi:10.1080/10248079608903712
9. Zhan FB (1997) Three fastest shortest path algorithms on real road networks: data structures and procedures. J Geogr Inf Decis Anal 1:69–82
10. Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 2:100–107
11. Goldberg AV (2007) Point-to-point shortest path algorithms with preprocessing. Lect Notes Comput Sci 4362:88–102. doi:10.1007/978-3-540-69507-3_6
12. Golden LB, Ball M (1978) Shortest paths with Euclidean distances: An explanatory model. Networks 8:297–314. doi:10.1002/net.3230080404
13. Sedgewick R, Vitter JS (1986) Shortest paths in Euclidean graphs. Algorithm 1:31–48. doi:0.1007/BF01840435
14. Uchida T, Iida Y, Nakahara M (1994) Panel survey on drivers' route choice behavior under travel time information. In: Proceeding of vehicle navigation and information systems, 1994, pp 383–388
15. Huang Y, Jing N (1996) Evaluation of hierarchical path finding techniques for ITS route guidance. In: Proceeding of annual meeting of ITS America, 1996, pp 340–350
16. Car A, Frank AU (1994) General principles of hierarchical spatial reasoning—the case of way-finding. In: Proceedings of the 6th international symposium on spatial data handling, 1994, pp 646–664
17. Jagadeesh GR, Srikanthan T, Quek KH (2002) Heuristic techniques for accelerating hierarchical routing on road networks. IEEE Trans Intell Transp Syst 3:301–309. doi:10.1109/TITS.2002.806806
18. Fu L, Sun D, Rilett LR (2006) Heuristic shortest path algorithms for transportation applications: state of the art. Comput Oper Res 33:3324–3343. doi:10.1016/j.cor.2005.03.027
19. Dantzig GB (1960) On the shortest route through a network. Manag Sci 6:87–90
20. Fu L (1996) Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks. Ph.D. Thesis, University of Alberta
21. Pohl I (1971) Bi-directional search. Mach Intel 6:127–140