Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 3324–3343
computers & operations research
www.elsevier.com/locate/cor

Heuristic shortest path algorithms for transportation applications: State of the art

L. Fu[a,*], D. Sun[b], L.R. Rilett[c]

[a]*Department of Civil Engineering, University of Waterloo, Waterloo, ON, Canada N2L 3G1*
[b]*College of Automation, Chongqing University, Chongqing, 400044, China*
[c]*Mid-America Transportation Center, University of Nebraska-Lincoln, W339 Nebraska Hall, P.O. Box 880531, Lincoln, NE 68588-0531, USA*

Available online 3 May 2005

Abstract

There are a number of transportation applications that require the use of a heuristic shortest path algorithm rather than one of the standard, optimal algorithms. This is primarily due to the requirements of some transportation applications where shortest paths need to be quickly identified either because an immediate response is required (e.g., in-vehicle route guidance systems) or because the shortest paths need to be recalculated repeatedly (e.g., vehicle routing and scheduling). For this reason a number of heuristic approaches have been advocated for decreasing the computation time of the shortest path algorithm. This paper presents a survey review of various heuristic shortest path algorithms that have been developed in the past. The goal is to identify the main features of different heuristic strategies, develop a unifying classification framework, and summarize relevant computational experience.

Keywords: Artificial intelligence; Traffic network; Shortest path algorithm; Heuristics; Heuristic shortest path algorithm; In-vehicle route guidance system (RGS); Intelligent transportation systems (ITS)

1. Introduction

In recent years there has been a resurgence of interest in the shortest path problem for use in various transportation engineering applications. This is directly attributed to the recent developments in Intelligent Transportation Systems (ITS), particularly in the field of in-vehicle Route Guidance System (RGS) and

* Corresponding author. Tel.: +1 519 888 45674; fax: +1 519 888 6197.
E-mail addresses: lfu@uwaterloo.ca (L. Fu), d3sun@hotmail.com (D. Sun), lrilett2@unlnotes.unl..edu (L.R. Rilett).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.03.027

real time Automated Vehicle Dispatching System (AVDS) where there is a definite need to find the shortest paths from an origin to a destination in a quick and accurate manner. In a distributed RGS, an in-vehicle computer is commonly used to calculate the optimal route in a large traffic network. Typically the recommended routes must be found within a very short time period (e.g., a few seconds). In a real-time AVDS new routes and schedules must be identified within a reasonable time after a customer requests a service. Because the travel times are the basic input to the real-time routing and scheduling process and are dynamic in most urban traffic environments, there is an implicit requirement to use a minimum path algorithm repeatedly during the optimization procedure.

In the above applications, the traditional optimal shortest path algorithms often cannot be used because they are too computationally intensive to be feasible for real-time operations. A number of heuristic search strategies have been developed for increasing the computational efficiency of shortest path search. Most of these heuristic search strategies originated in the artificial intelligence (AI) field [1–4], where the shortest path problem is often used as a testing mechanism to demonstrate the effectiveness of these heuristics.

The current RGS field tests in North America, Europe and Japan have generated renewed interest in using heuristic algorithms to find shortest paths in a traffic network for real-time vehicle routing operations. Guzolek and Koch [5] discussed how heuristic search methods could be used in a vehicle navigation system. Kuznetsov [6] discussed applications of an A* algorithm (called the force driven method in his paper), a bi-directional search method, and a hierarchical search method used for identifying paths in the TravTek project. Since then, many researchers have followed the same track and tried to develop a universal strategy for improving the efficiency of the shortest path search process. These efforts have resulted in a large body of literature including a wide spectrum of search strategies and mechanisms. But to date, there has not been a comprehensive review examining the implementation and performance of these heuristic algorithms.

This paper first provides a brief overview of the optimal shortest path algorithms, which are often considered as the starting point for the development of many heuristic shortest path algorithms. The paper then examines four heuristic search strategies: (i) limit the area searched, (ii) decompose the search problem, (iii) limit the links searched, and (iv) some combination of above. The review summarizes the distinguishing idea of each search strategy and its algorithmic variations.

2. The shortest path problem and optimal algorithms

A road traffic network is represented by a digraph $G(N, A)$ that consists of a set of nodes N and a set of arcs A (or links used in this paper). Denote the number of nodes $|N| = n$ and the number of links $|A| = m$. A link $a = (i, j) \in A$ is directed from node i to node j and has an associated generalized cost c_{ij}. The generalized cost represents the impedance of an individual vehicle going through that link and is usually described by link travel time, link length, tolls, etc. Without losing generality, the term link travel time is used mostly in this paper. A path from an origin(o) to destination(d) may be defined as a sequential list of links: $(o, j), \ldots, (i, d)$ and the travel time of the path is the sum of travel times on the individual links. The problem is to find the path that has the minimum total travel time from the origin node to the destination node.

2.1. Optimal algorithms

This shortest path problem (SPP) has been studied for over 40 years in diverse fields such as computer science and transportation [7]. Due to their computational tractability, most of the research in this area has focused on developing increasingly efficient optimal algorithms to solve the problem. The majority of the optimal shortest path algorithms are essentially applications of dynamic programming theory to the search for the shortest path in a graph. The shortest path is found through a recursive decision making procedure from the origin node (or destination node) to the destination node (or origin node).

There is a standard procedure followed by most shortest path algorithms. To describe this procedure, some notations are introduced. The route cost from the origin node to a particular node i is defined as $L_{(i)}$ and this route cost is commonly referred as the "label" of the node. P is denoted as the list which stores the preceding links on the shortest path tree to each node, and $P_{(i)}$ represents the preceding link on the shortest path to node i. Q is denoted as the scan eligible node set which manages the nodes to be examined during the search procedure. The standard steps of the shortest path algorithms are as follows (assuming that the algorithm starts from the origin node):

Step 1: Initialization: Set $i = o$; $L_{(i)} = 0$; $L_{(j)} = \infty\ \forall j \neq i$; $P_{(i)} = \text{NULL}$.
Define the scan eligible node set $Q = \{i\}$;
Step 2: Node Selection: Select and remove a node (i) from Q.
Step 3: Node Expansion: Scan each link emanating from node i. For each link $a = (i, j)$
If
$L_{(i)} + c_a < L_{(j)}$
then
$L_{(j)} = L_{(i)} + c_a$; $P_{(j)} = a$
Insert node j into Q
Step 4: Stopping Rule: If $Q = \emptyset$ then STOP.
otherwise: goto step 2.

The major variations between different algorithms pertain to the data structure used to form the *scan eligible node set* and the manner in which the nodes are identified and selected for examination [8]. Based on the behavior of an algorithm, the optimal shortest path algorithms are usually classified into two categories: label-correcting or label-setting algorithms.

2.1.1. Label-setting algorithm (LS)

For the label-setting (LS) algorithm [9–11], the scan eligible node set is ordered based on the current path costs from the starting node to the nodes in the node set. During the shortest path search the node with smallest label is selected for examination while concurrently identifying the shortest path to the node. The major difference among the LS algorithms is the data structure used to maintain the ordered scan eligible node set. Examples include LS with sorted list [9], LS with buckets [12], and LS with binary heap [13].

The major feature of an LS algorithm is that if only the route from an origin node to a single destination node is required, the algorithm can be terminated when the label of that destination node is set. This type of operation is usually referred as *one-to-one* search mode. As a result, the LS algorithms are particularly appropriate for applications like RGS where the objective is to find shortest paths between two specific locations.

2.1.2. Label-correcting algorithm (LC)

The label-correcting (LC) algorithm [14–17] uses a list structure to manage the scan eligible node set that needs to be examined during the shortest path tree building process. It is the variations of the list operation policy that are used to differentiate the LC algorithms such as LC with queue [15], LC with double ended queue [18], and LC with threshold lists [19].

The major feature of an LC algorithm is that it cannot provide the shortest path between two nodes before the shortest path to every node in the network is identified. The necessity of this type of operation (referred as *one-to-all* search mode) makes the LC algorithms more suitable in situations when many shortest paths from a root node need to be found. Based on empirical evidence the LC algorithm is often used in transportation planning applications where multiple routes have to be identified.

2.2. Computational performance

The computational performances of various LC and LS algorithms have been studied widely from both a theoretical and empirical perspective in many different research fields. According to the literature focused on transportation networks with sizes ranging from hundred links to several millions of links [20–25], a number of conclusions pertaining to algorithm characteristics have been identified. Among the LC algorithms, double ended queue and threshold list data structures are found dominant with respect to computational efficiency. The difference in computation time between those two LC algorithms is relatively minor for most transportation road network problems and consequently the former has commonly been used because it is easier to implement. On the other hand, among LS algorithms Dial's bucket implementation and binary heap data structure are the most efficient.

3. Heuristic shortest path algorithms

The optimal shortest path algorithms discussed in Section 2 tend to be too computationally intensive for real-time one-to-one applications in realistic traffic networks. This "inefficiency" stems from the fact that the algorithms employ "uninformative" outward search techniques without making use of a priori knowledge on the location of the origin and destination nodes, path composition and network structure. For example, if the origin node was located in the center of the city and the destination node was located in the far south, the optimal search techniques would be just as likely to search for the minimum path route north of the origin node as they would search south of the origin node. Intuitively, the efficiency of the algorithms could be improved if more information was used in the search process. This latter point was recognized very early by researchers in the AI field and a number of heuristics were proposed that attempted to use various sources of additional knowledge to reduce the search efforts.

The heuristic search strategies identified in the literature can be generally classified into four strategies: (i) limit the area searched, (ii) decompose the search problem, (iii) limit the links searched, and (iv) some combination of above. In the following sections, these heuristic search strategies and their applications in shortest path search are investigated.

3.1. Limit the search area

The non-informative search algorithms, such as optimal LS and LC algorithms, have a fundamental disadvantage in that they examine all the intermediate nodes from origin node to destination node without

considering how likely it is for these nodes to be on the shortest path. The idea behind the "limit search area" strategy is to make use of some knowledge about the attributes of the shortest path(s) from the origin node to the destination node to constrain the shortest path search within a certain area. The theory is that the resulting search area would be much smaller than that by a non-informative optimal algorithm.

There could be many different ways to limit the area searched in algorithms and the two popular methods, namely, the branch pruning method and A* algorithm are described in the following section.

3.1.1. Branch pruning method

The branch pruning method was developed from work in dynamic and stochastic route selection research [26,27]. This fundamental idea is similar to the popular AI technique called IDA* [28], which attempts to limit the area searched by "pruning" the intermediate nodes that have little possibility of being on the shortest path to the destination node.

In a typical urban traffic network, each link (or road segment) is typically connected only to the neighboring nodes (e.g., intersections) and the travel time on a link is generally correlated with its length. This attribute allows the search area to be constrained within a specified area surrounding the origin node and destination node. The nodes outside this area are assumed to have little probability of being on the shortest path and therefore can be dismissed without further examination during the search procedure. The problem is to define the search area such that the computation time can be effectively reduced while at same time obtaining a good solution. In Fu's [26] implementation, the following inequality was proposed to bound the search area when travel time is used as a routing criterion or general cost:

$$L_{(i)} + e_{(i,d)} \leqslant E_{(o,d)}, \tag{1}$$

where $L_{(i)}$ is the current minimum travel time from origin node o to node i; $e_{(i,d)}$ the estimated travel time from node i to destination node d; $E_{(o,d)}$ an estimated upper bound of the minimum travel time from the origin node to the destination node.

Eq. (1) can be added to the optimal LS and LC search procedure to form the following two heuristic branch pruning shortest path algorithms: branch pruning LS algorithm and branch pruning LC algorithm. The branch pruning LC algorithms have the same form as the optimal algorithms with the only difference being that during the search procedure condition (1) will be tested before a node is selected for examination. For example, the branch pruning algorithm has the same procedure as the ordinary labeling algorithm shown in Section 2.1, except the second step is modified as follows.

Step 2: Node Selection: Select and remove the node with the lowest label (travel time) from Q,
This is node i;
If $L_{(i)} + e_{(i,d)} > E_{(o,d)}$, then goto step 4.

Basically, the modified step 2 states that if node i is not in the search area it is removed from further consideration. The efficiency of the branch pruning algorithms is schematically illustrated in Fig. 1. It can be seen that the new heuristic strategy reduces the search area into an ellipse from a circle expanded by an optimal LS algorithm. In an idealized Euclidean grid network, these heuristic algorithms are shown to examine a search area that could as small as 20% of the search area used by the LS algorithm [26].

It should be noted that the application of the branch pruning method in the LC algorithm effectively allows the LC algorithm to be used in a one-to-one search mode. The resulting LC algorithm should therefore be more efficient than the corresponding LS algorithm because, while they have the same

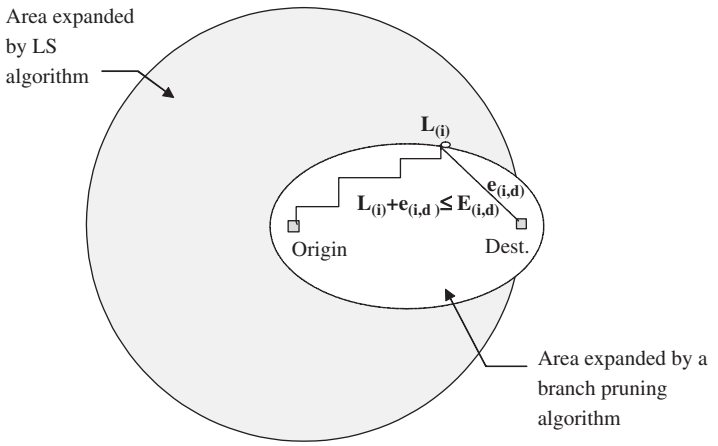

Fig. 1. Pruning power of the branch pruning algorithms.

search area, the LC algorithm uses the queue data structure which does not need to maintain an ordered list as in the LS algorithm [20].

The efficiency and accuracy of the branch pruning shortest path algorithms depend on the quality of the estimation functions $e_{(i,d)}$ and $E_{(o,d)}$. It can be seen that optimality of the branch pruning algorithms will be preserved only if the estimation $e_{(i,d)}$ is always lower than the minimum travel time from node i to destination node d and at the same time the estimated value of $E_{(o,d)}$ is greater than the minimum travel time from origin node o to destination node d. In addition, it can be noticed that the branch pruning algorithms will become optimal shortest path algorithms when $e_{(i,d)}$ approaches to zero and $E_{(o,d)}$ approaches to infinite. That is, when the constraint in Eq. (1) is no longer in effect.

On other hand, if the $e_{(i,d)}$ value is over-estimated and/or the $E_{(o,d)}$ value is underestimated, the minimum path to the destination node may be discarded before the destination node is examined. As a result, the $e_{(i,d)}$ and $E_{(o,d)}$ functions are critical for the branch pruning algorithms. While there are many methods by which $e_{(i,d)}$ and $E_{(o,d)}$ may be defined, Fu [26] suggested using the travel distance $D_{(i,d)}$ and average travel speed V to estimate $e_{(i,d)}$:

$$e_{(i,d)} = D_{(i,d)}/V \tag{2}$$

and define $E_{(o,d)}$ as a function of $e_{(o,d)}$:

$$E_{(o,d)} = Ke_{(o,d)}, \tag{3}$$

where K is referred to as the bound factor which is used to set the difference between the estimated lower bound on travel time ($e_{(o,d)}$) and the upper bound on travel time ($E_{(o,d)}$).

Given the above definitions, the performance of the branch pruning algorithms will be controlled by the estimation of travel distance $D_{(i,d)}$, the average speed V, and the bound factor K. The travel distance $D_{(i,d)}$ is set to the Euclidean distance from node i to node d and can be directly calculated based on their coordinates. The bound factor K therefore becomes the only controllable parameter in a branch pruning algorithm. The larger the bound factor, the larger the search area becomes, and the more likely the optimal solution will be found. The negative side to a larger bound factor is the associated increase in computation

time. A suitable value of K can be determined by conducting an experimental computation analysis on a given traffic network [26].

One problem pertaining to the branch pruning shortest path algorithm is that the algorithm may stop without providing any solution if the bound factor used is too small so that all of the branches of the shortest path tree from the origin node are pruned before reaching the destination node. This problem can be resolved by adding a self-adjustment loop in the algorithm, i.e., the algorithm will automatically increase the bound factor K and restart the search from the origin node if a solution has not been found when the scan eligible list is exhausted. Instead of restarting the whole search, an alternative method would be to record all the pruned nodes during the search procedure and place them back into the scan eligible node list once the list is exhausted. The latter approach has been shown much faster than the former method [26].

The K value at each iteration can be determined either by simply increasing a pre-specified proportion or by using the information from the previous iteration. In the latter method, the K value can be determined by using the ratio of maximum value of the estimated route travel times going through the pruned nodes to the estimated lower bound on the route travel time from the origin node to the destination node:

$$K_{n+1} = \frac{\max_{i \in P_n} \{L_{(i)} + e_{(i,d)}\}}{e_{(o,d)}}, \tag{4}$$

where P_n is the pruned node set at iteration n.

It should be noted that a good initial K value can avoid both the self-adjustment problem and searching an unnecessarily large area. A computational analysis done by Fu [26] shows that with appropriate parameter values, a computational saving of 40–60% over the ordinary LS or LC algorithm is achievable.

Karimi [27] proposed limit the path finding space within a rectangular area (called window) defined on the basis of the origin and destination locations of the path. During the search process, the nodes that are located out of the window are pruned for further examination. The problem of premature termination was however not addressed, i.e., it may not find a solution at all. Also, when the origin and destination nodes are far apart (e.g., located at the edge of the network in opposite direction), the benefit of using a window to constraint the search area will diminish.

Lysgaard [29] introduced a pruning technique through a two-phase process. In the first phase, an A* algorithm used to find an upper bound of the travel time from the origin node to the destination (the A* algorithm is discussed in the following section). This upper bound is subsequently used in Eq. (1) to prune nodes with a label greater than the upper bound. The algorithm is guaranteed to find the optimal solution, but its performance is tied to the heuristic function used in the A* algorithm. A computational analysis based on a real road network of 1064 nodes indicated a 40–60% savings in computational time as compared to the Dijkstra algorithm.

3.1.2. A* *algorithm*

Unlike the branch pruning method in which the nodes with a low probability of being on the shortest path are pruned, the algorithm A* keeps these nodes on the scan eligible node set but assigns a low priority to them. The A* algorithm was first proposed by Hart et al. [1] and further discussed and popularized by Nilsson [2], Pohl [30] and Pearl [4].

The A* algorithm makes use of a heuristic evaluation function $F_{(i)} = L_{(i)} + e_{(i,d)}$ as a label for node i, where $L_{(i)}$ is the travel time of the current evaluated path from origin node to node i and $e_{(i,d)}$ is an estimated travel time from node i to destination node d. The sum of these two functions, $F_{(i)}$, is the "merit"

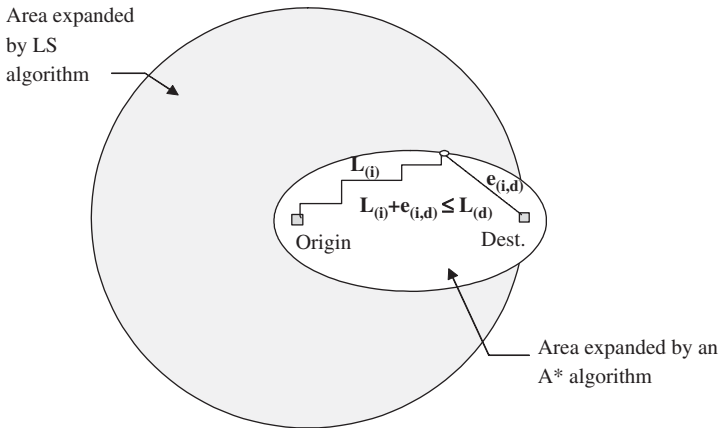

Fig. 2. Pruning power of the A* algorithms.

of node i, and reflects how likely it is for node i to be on the shortest path. The lower the merit of a node, the more likely the shortest path will go through this node. Based on this idea, the algorithm does a best first search, (i.e., it maintains an ordered list of nodes to be scanned according to their F values and selects a node whose F value is the lowest among all the nodes for expansion). The selected node is expanded by scanning its outgoing (or incoming) links, evaluating the link destination nodes according to their F values, and inserting them into the ordered scan eligible node set. This continues until the destination node is chosen for expansion. Therefore, the A* algorithm has a similar procedure as the LS algorithm except that the evaluation function $F_{(i)}$would be used as the label, instead of $L_{(i)}$, for determining the node examination order. The main difference lies in step 3 which is modified as shown below:

Step 3: Node Expansion: Scan the outgoing links of node i. For each link (i, j)
If
$L_{(i)} + c_{ij} + e_{(j,d)} < F_{(j)}$,
then
$L_{(j)} = L_{(i)} + c_{ij}$; $F_{(j)} = L_{(i)} + c_{ij} + e_{(i,d)}$; $P_{(j)} = a$,
Insert node j into Q;

Because the A* algorithm proceeds as a best first search, the nodes which satisfy the following inequality would be examined before the algorithm terminates (or the destination node is examined):

$$L_{(i)} + e_{(i,d)} \leqslant L_{(d)}. \tag{5}$$

It is this attribute that gives the A* algorithm an important solution property—it is guaranteed to find the optimal solution as long as the heuristic function is *admissible* or never overestimates the actual travel time [2].

Eq. (5) also defines the area which would be examined during the shortest path search. The resulting search area is essentially elliptical in shape and may be contrasted to the circular shape of the search area associated with the LS algorithm. The relative search area of the A* algorithm is schematically illustrated in Fig. 2. It should be noted that all the nodes on the edge of the search area (ellipse) will need to be examined during the search procedure. This contrasts with the situation of the branch pruning

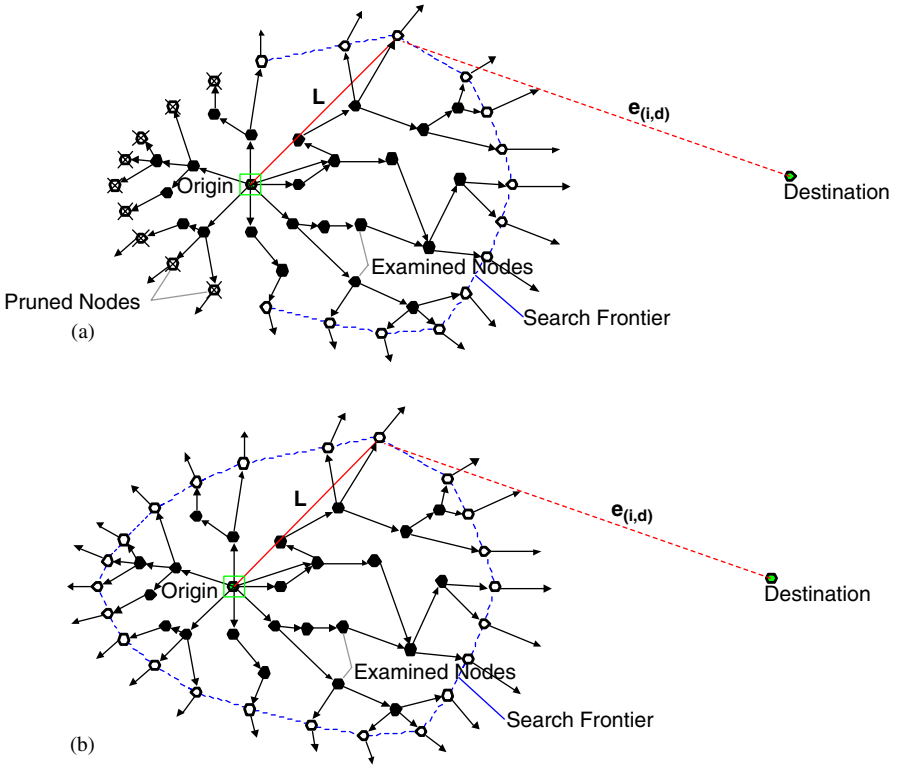

Fig. 3. Shortest path search: branch pruning vs. A*.

algorithm where some nodes on the edge are discarded from the scan eligible node list after proving to be located out of the assumed feasible solution area, as shown in Fig. 3. Consequently, the branch pruning algorithm maintains a smaller scan eligible node set as compared to the A* algorithm and is expected to be computationally faster.

The performance of the A* algorithm in Euclidean networks has been well documented. Golden et al. [31] empirically found that the A* algorithm expands less than about 10% of the nodes that would be expanded by a LS algorithm. Sedgewick et al. [32] proved that the A* finds the shortest path in many Euclidean graphs with an average computation effort $O(n)$, compared to $O(n \log n)$ required by the LS algorithm. More recently, Nicosia et al. [33] proved a general approximation property of A* which allows a solution being found with a given approximation ratio: if the heuristic function $e_{(i)}$ is not itself a lower bound, but there exists $k > 1$ such that e/k is a lower bound, then the solution returned by A* is less than or equal to k-times the value of an optimal solution.

The actual performance of the A* algorithm in a transportation network depends primarily on the quality of the heuristic function $F_{(i)}$ or $e_{(i,d)}$ used. Similar to the branch pruning method, $e_{(i,d)}$ can also be estimated by Eq. (2). When the travel distance is estimated by using Euclidean distance, the average speed (V) becomes the only controllable parameter in this algorithm. In order to ensure that $e_{(i,d)}$ is admissible or a lower bound estimate, an upper bound average speed should be used. However, it should be noted that the higher the average speed used, the larger the search area will be, and thus the more

computational effort required. As the value of V approaches infinity, the estimated travel time approaches zero and the A* algorithm becomes the same as the LS algorithm. On the other hand, the use of a smaller value of V (or a non-admissible heuristic function) can limit the area searched to a greater degree and the algorithm would be faster—however the probability of finding the optimal solution is reduced. Fu [26] showed that, with an appropriate value of the parameter V, the A* algorithm can guarantee optimal solutions while still reducing computation time by 10–30% compared to the LS algorithm.

Bander and White [34] extended the A* algorithm to include a simple form of machine learning by utilizing travel times of previously developed paths as the heuristic function value. A small test network with 131 nodes (a model of the major roads in Cleveland) was used in their experimental study. Their tests show that on average A* and their improved A* expanded 58% and 23% of the nodes expanded by a LS algorithm. The small number of nodes examined by their proposed heuristic was a result of some initial "learning" of the network and storage of some good paths.

Adler [35] attempted to determine if extra computational effort in determining a more exact estimation of travel time to the destination node ($e_{(i,d)}$) is justified. The general idea is to look ahead to the next node and use the link travel time and the distance from the next node to the destination node to estimate $e_{(i,d)}$. The next node is defined in this context to be the one that minimizes $e_{(i,d)}$ for the current node under examination. The author contends that, for sparse networks with a low average out-degree, the computational effort for improving the heuristic function will be less than that for expanding unnecessary nodes because of using an inferior heuristic function. Several randomly generated and real networks are used in a computational study to support this assertion. The results are mixed for the randomly generated networks. However, in the majority of cases for real networks the proposed algorithm outperforms the standard A* algorithm in the range of 10–15%. Note that the idea of using a look-ahead heuristic has been widely used in the AI field for dealing with different types of problems [36–38].

Jacob et al. [25] conducted an extensive computational analysis on the performance of the A* algorithm using a large sized real network involving approximately 9,863 nodes and 14,750 links. They evaluated a range of heuristic functions from non-admissible (also called overdo heuristic) to admissible and concluded that it is possible to find an optimal non-admissible function which will make the algorithm run much faster without a significant loss in solution quality. They also found that the non-optimal solutions identified by the A* algorithm with a non-admissible heuristic function are acceptable and feasible alternatives from a practical point of view.

3.2. Decompose the search problem

It has been commonly recognized that the computational effort required to solve a generic search problem usually grows faster than the size of the problem. For example, the computational time required to find a shortest path from an origin node to a destination node depends on the number of nodes searched before reaching the destination node and is thus a high order polynomial function of the distance between the locations. As a result, if the original problem can be decomposed into smaller sub-problems, substantial computational savings can be realized. This section introduces how this strategy is implemented using the bi-directional search method and subgoal methods.

3.2.1. Bi-directional search method

Most traditional search methods are uni-directional in the sense that they seek the problem solutions from the origin node to the destination node (or vice versa). The bi-directional search method, first

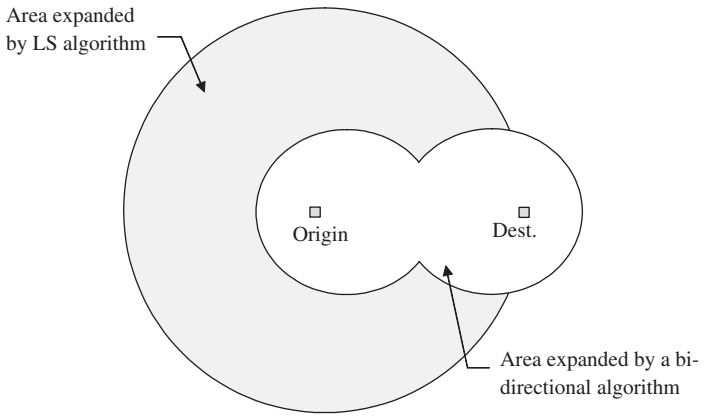

Fig. 4. Pruning power of the bi-directional search algorithms.

suggested by Dantzig [10] and latter proposed by Nicholson [39], attempts to divide the search procedure (problem) into two separate procedures (problems). One search proceeds forwards from the initial stage while another search proceeds backwards from the goal stage. The premise of this algorithm approach is that a solution is found when these two search procedures meet at some middle stage(s). This search idea is easily applied in the shortest path search problem for transportation networks. The algorithm would simultaneously build the shortest path trees outward from the origin node and the destination node until some stopping criterion is met. Fig. 4 schematically illustrates the concept and effectiveness of this method as compared to the LS algorithm.

There are two factors that influence the efficiency of a bi-directional algorithm. The first factor is the method by which the two forward and backward search processes alternate. Although iterating equally between both searches would be the simplest method, it is not the most efficient. The best strategy involves identifying the minimum path search that has the fewest nodes which have been examined but have not had the minimum path identified. That is, the computational effort is concentrated on the search having the least nodes to examine and sort during each iteration. Intuitively, the search that is in a space area of a network will get priority.

The second factor, which is more critical than the first, is the criterion that decides when the two opposite search processes should stop. The stopping criterion that guarantees to find the shortest path is identified by Nicholson [39].

$$L^o_{(i)} + L^d_{(i)} \leqslant \min_{j \in N} \{L^o_{(j)}\} + \min_{j \in N} \{L^d_{(j)}\} \tag{6}$$

where the label $L_{(i)}$ with the superscript o and d represents the label of node i on the shortest path tree out of the origin node (o) and the label of node i on the shortest path tree out of the destination node (d), respectively. Unfortunately when this stopping criterion is used, the resulting bi-directional search algorithm is inferior to the uni-directional algorithms [40]. It was conjectured that the inefficiency was stemmed from the possibility that the two searches may proceed along different directions—a situation which was compared by Pohl [30] to missiles that pass each other. As a result, the nodes expanded by a bi-directional search process may grow into nearly completely uni-directional trees before satisfying the condition (Eq. (6)), instead of meeting in the "middle" between the origin and destination.

Pohl [30] proposed a heuristic bi-directional shortest path algorithm (called BHPA) which essentially includes two A* searches in opposite directions. The forward A* search uses a heuristic distance function to the destination, i.e., $e(i, d)$ while the backward A* search uses a heuristic distance function to the origin, i.,e., $e(o, i)$. By using these two front-to-end distance functions, the problem for the search frontiers to pass each could be avoided.

De Champeaux and Sint [41] and De Champeaux [42] extended the BHPA by using a front-to-front distance evaluator to guide the two search processes and showed that bi-directional heuristic search is efficient in terms of the number of nodes examined. The algorithm was further revisited by Kwa [43] and Kaindl and Kainz [44] for more efficient implementations. Note that all these algorithms were originated in the field of AI and they have not been tested extensively in solving the shortest path problem in a transportation network. As a result, the computational performance of these algorithms are unknown in the context of transportation networks.

Fu [26] evaluated the BHPA algorithm in solving the shortest path problem in a transportation network and found that it is in fact slower than the A* algorithm. A new bi-directional search method was proposed in which the algorithm terminates when the two search trees (A*) intersect at a given number of frontier nodes (M). The performance of this heuristic bi-directional algorithm is controlled by the parameter M. The larger the value of M, the more likely the optimal solution will be found and the longer the computation time will be. By selecting an appropriate value of M, a satisfactory balance between accuracy and efficiency can be reached. However, it was shown empirically by Fu [26] that the bi-directional heuristic algorithm was inferior to the branch pruning algorithm discussed previously in terms of both computational efficiency and solution quality. It should be noted that while the bi-directional search strategy does not hold a lot of promises when used alone, it is a critical element in the implementation of some hierarchical search algorithms discussed in Section 3.3.

3.2.2. Subgoal method

A subgoal may be defined as an intermediate state of the optimal solution to a problem. For the shortest path search problem in a road traffic network, subgoals could be those nodes or links that are located between the origin and destination locations between which a shortest path is to be identified. With advance knowledge of subgoal locations, the problem of finding the shortest path from the origin node to destination node can be decomposed into two or more smaller problems. For example, if there is one subgoal node, the original problem can be solved by solving two sub-problems: one is to find the shortest path from the origin node to the subgoal node while another is to find the shortest path from the subgoal node to the destination node [45,46].

The efficiency of the subgoal method depends on the number and location of the subgoal nodes. The more the subgoal nodes and the more uniformly they are distributed between the origin node and the destination node, the larger the computational saving will be. In an idealized situation if there are M subgoal nodes equally located between the origin node and the destination node in a uniform, infinitely large network, the reduction of the search area using the subgoal nodes would be $1/(M+1)$. For example, if a single subgoal node is known and is located at the middle of the shortest path from an origin to a destination, then the use of this subgoal node will reduce the search area by approximately 50% as compared to the LS algorithm as shown in Fig. 5. The computational savings would be even greater if one of techniques that limit the area searched was also used.

Although the computational efficiency of the subgoal method is obvious, the penalty of this search reduction is that the solution may not be optimal, i.e., the path going through the subgoal node may not be

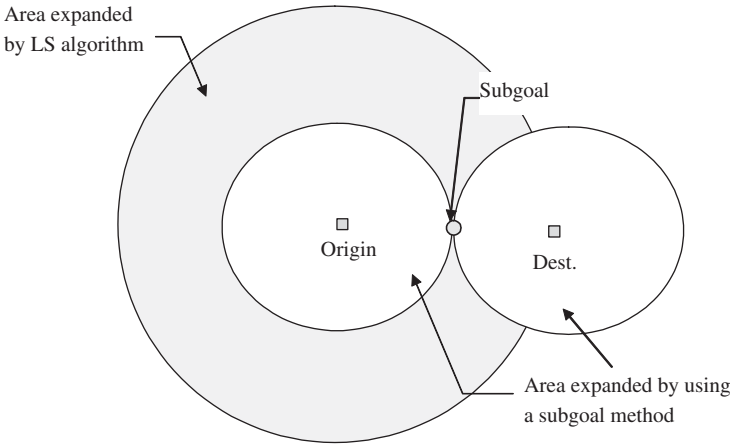

Fig. 5. Pruning power of using a sub-goal.

the actual shortest path. As a result, application of this method depends on whether or not such valuable information are available. In a road network routing environment, potential subgoals include bridges, intermediate stops, driver's preference, etc.

3.3. Limit the search links

As discussed in Section 2.1, in the shortest path search procedure the main decision during each iteration involves the identification of the appropriate link emanating from each node. In a traditional shortest path algorithm, when a node is selected for expansion, all links from this node will be examined regardless of how likely they are to be on the shortest path(s) or used as part of the optimal path in practical situations. The basic idea of limiting the links searched is to systematically skip the examination of the links that have a low probability of being either on the shortest path or used in practical situations. This idea can be effectively implemented by using the hierarchical search method discussed in the following section.

3.3.1. Hierarchical search method

The hierarchical search strategy is well known in the AI field and is also known as an 'abstraction' problem solving strategy. Plolya [47] is probably the first to coin the search concept in the context of road network. The concept was further explored by Sacerdoti [48], Korf [49] and many other researchers [50–55]. The basic idea behind the hierarchical search is that in order to effectively find a solution to a complex problem, the search procedure should at first concentrate on the essential features of the problem without considering the lower level details, and then complete the details later. This method is similar to how a driver manually finds a route between two locations on a map. Typically, the driver first searches the major roads in the area adjacent to the origin location and destination location, and then find the access roads to the major road from both origin and destination locations [56]. There is a large body of literature on cognitive mapping and spatial knowledge, which is beyond the scope of this paper (see, e.g. [57,58]).

While the basic idea behind the hierarchical search strategy is easy to describe, its implementation for solving the shortest path algorithm is much more complex as compared to the heuristic shortest path

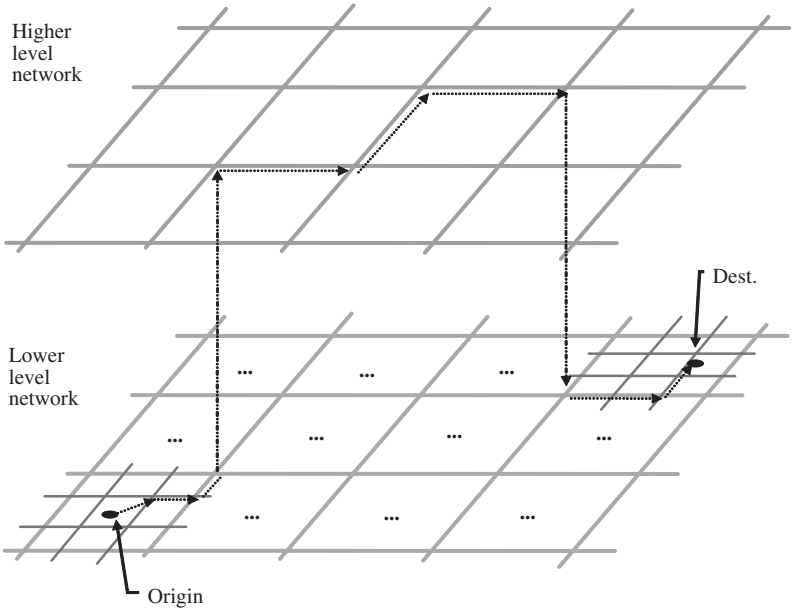

Fig. 6. Shortest path search in a two level hierarchical network.

algorithms discussed in the previous sections. The complexity is mainly due the following two issues [51]:

(a) How to cast a real road network into a hierarchical framework? (and how many layers should be used?).
(b) How to control the search transition between the layers—when to stay on a level and when to move onto the next level?

Road networks commonly inherit hierarchical topology and can thus be fairly straightforwardly cast into a hierarchical representation [50]. For example, transportation networks links are usually specifically designed to meet certain travel functions. Freeways and major arterials are designed to provide for long distance movements while local streets and collectors are used primarily for land access. Therefore, the roadway links may be conveniently classified according to their functions, which can then be used within the search process. Chou et al. [53] proposed using link length as a criterion and extracted the long links to form a high-level sub-network and grouped the shorter links and their nodes into a lower-level sub-networks. Liu [52] and Jagadeesh et al. [54] suggested grouping all roads into two levels according to their attributes such as speed limit and number of lanes.

Shortest path search in a hierarchical network could be implemented in different ways. Shapiro et al. [59] and Car and Frank [51] proposed an approach that starts by first finding two shortest paths from the origin and destination nodes to the nearest entry points to the next higher level and then calculating the shortest path between these entry points using an ordinary LS algorithm. This can be illustrated by using a simple two level hierarchical network as shown in Fig. 6. The search starts from the lower level network where the origin node and destination node are located. The shortest path trees are constructed

forward from the origin node and backward from the destination node on the lower level network. The searches on the lower level network (i.e., land access links) are bounded by the links (or nodes) which also reside in the higher level network. Once the search trees touch the higher level network (that is, the nearest entry nodes are found), an LS algorithm is called to identify the shortest path between the entry points.

This technique may work well for long distance trips, but will penalize short and medium distance trips because the nearest entry points may not be the best locations to link the origin and destination on the higher level. Liu [52], recognized this problem and proposed an algorithm that considers all access points to the major roads surrounding the origin and destination. Their hierarchical algorithm considers a two-level hierarchy and begins with stitching the low-level sub-grid networks where the origin and destination nodes are located to the higher level network before invoking an ordinary shortest path search algorithm (A*).

Similar to Liu [52], Jagadeesh et al. [54] also considered a two-level hierarchical network. Their algorithm starts with creating an augmented network by combining the higher level network with a set of dummy links (also called logical links) that link the origin and destination nodes to the corresponding entry nodes. The cost of each dummy link is the minimum travel time of the corresponding travel path identified by a short path algorithm. Their algorithm also includes a network pruning mechanism, similar to the branch pruning technique, to further reduce the computational time.

One issue inherited with a hierarchical search algorithm is that it usually does not allow any shortcuts such as moving from one arterial road to another by using a residential road. In a traffic network there usually exist many types of shortcuts and some of them may be unavoidable. For example, it is usually necessary to shift between two parallel freeways by going through an arterial road which connects them. In this situation, the search procedure discussed above will miss the actual shortest path (the path which has a shortcut). Car and Frank [51] and Jagadeesh et al. [54] suggested a pre-step that checks if the origin and destination nodes are located within the same sub grid or two adjacent grids and if so, invokes a non-hierarchical shortest path algorithm. This method, of course, does not address the problem completely as long trips can also have shortcuts with the paths. Liu [52] included a simple grid checking step to consider these shortcuts that transverse a single low level road linking one major intersection with another major intersection. This method was found quite effective in identifying paths with shortcuts, but it increases the computational burden of the search process. Kuznetsov [6] suggested the idea of using the number of nodes searched at each level as a control; however the details of their algorithm was not revealed.

Hierarchical search strategy has proved very effective in reducing the complexity of large problems, and may well be the only one which has potential to beat the nonlinear increase of computation time in terms of problem size for many problems [49]. Car and Frank [51] showed that using a hierarchical network structure would reduce the computational complexity of a shortest path algorithm by several orders of magnitude (e.g., from $O(m^2 \log m)$ of the Dijkstra's algorithm to $O(\log_k m)$, where m is the number of links). The actual computational savings of a hierarchical algorithm are however a function of the number of network topology (layers and connections), search rules incorporated and trip length. Car and Frank [60] conducted a computational study on a series of small networks (less than 400 nodes and 600 links) and found that their hierarchical algorithm was faster than a non-hierarchical algorithm by a factor of 2. However, they also found that the paths identified by their hierarchical algorithm were on average 50% longer than the optimal paths. The large error might be due to the fact that the networks they tested were relatively small and their implementation of connecting the origin/destination nodes to the higher level network is problematic.

Liu [52] tested their algorithm on a real network involving 12,697 nodes and 30,867 links, from which a major road network including 1096 nodes and 2574 links was created. Their algorithm was found to be 3–4 times faster than a non-hierarchical A* algorithm if shortcuts are evaluated and otherwise 8–10 times faster. The former produced routes around 9% longer than the optimal routes while the later was able to find the optimal solutions in most cases with an average error of 1%. Based on a computational study performed on 11,742 nodes and 30, 108 links, Jagadeesh et al. [54] found that their hierarchical algorithm (with a pruning step) was over 50 times faster than the conventional algorithm on a test network while providing routes that were longer than the optimal routes by 3.31% on average.

Finally, we should note that on another line of research related to the hierarchical search strategy discussed above, Shekhar et al. [61] proposed a hierarchical network model that provides an alternative approach to abstracting and structuring a topographical road map in a hierarchical fashion. It partitions the whole network into sub-graphs based on the road classification and pre-computes and stores the shortest path lengths between the boundary nodes of individual sub-graph. Huang et al. [62] and Jung and Pramanik [55] proposed a similar graph partition approach for structuring large topographical road maps for efficient storage and path computation. These hierarchical approaches focus more on how to partition and manage a transportation network and less on algorithmic implementation and are therefore not discussed in details in this review.

4. Conclusions and future research

There are a number of reasons for wishing to increase the speed of one-to-one minimum path algorithms. The most obvious is that in order to realize the full benefits of ITS hardware, the software components have to be able to operate in a real-time environment. This is true for individual in-vehicle RGS applications and in real-time multiple vehicle routing systems such as demand-responsive transit. This paper has surveyed various heuristic search strategies developed in the past to identify shortest paths in a transportation network. The different search strategies are categorized into three general classifications, including: (i) limit the area searched (branch pruning and A*), (ii) decompose the search problem (sub-goal and bi-directional search), (iii) limit the links searched (hierarchical search), and (iv) some combination of above. The following general conclusions can be obtained from this survey:

1. While the A* algorithm is the most popular heuristic among all heuristic algorithms, its computational efficiency in real transportation networks appears to be bounded by a factor of 2 (or 50% saving in computational time as compared to an ordinary LS). The idea of integrating a pruning step into a LC algorithm is innovative, but again the computational efficiency of the resulting algorithm itself is limited (60% or less). However, it should be noted that these heuristic strategies could work effectively with other heuristics such as bi-directional and hierarchical search to gain additional computational efficiency.
2. The idea of breaking down a shortest path problem into two sub-problems of smaller size (bi-directional search) is intuitively attractive, but its computational advantage in transportation networks has yet to be proved. Under some special implementations, a bi-directional shortest path algorithm could be faster than its counterpart—a uni-directional algorithm. However, its total computational efficiency in transportation networks is likely limited (less than 20%). While the bi-directional search method is

not of much advantage when used alone, it becomes effective when combined with the hierarchical search strategy.

3. By concentrating the search on the higher functional class links and exploring the lower class links only at the vicinity of the trip origin and destination, the hierarchical shortest path algorithm has the potential to break the shortest path problem into one with linear complexity. It has been shown that the computational time savings of a hierarchical algorithm could be by several orders of magnitude.

Past research has suggested that, by utilizing knowledge about the transportation network and the trip itself, heuristic search strategies have the potential to speed up a shortest path algorithm significantly. Further research is however needed in order to realize the full potential of these heuristic search algorithms. For example, most heuristic algorithms discussed in this paper have pros and cons in terms of computational efficiency and solution quality, it may be meaningful to investigate if these heuristics can work together in a single search engine for complementary gains. Also, in realistic transportation environments there exist various sources of additional information that may be helpful for reducing search time and improving the efficiency of the algorithms. It is expected to be fruitful to structure such knowledge and develop a formal framework for integrating it into a shortest path search process.

Furthermore, as discussed in Introduction, one of the major motivations for a heuristic shortest path algorithm is the need to quickly identify the best driving routes for a specific traveler. However, different travelers usually have different route choice preferences, often involving multiple decision factors such as travel time, costs, reliability and road types. Future research is therefore needed to apply the heuristic search strategies for solving the constrained shortest path problem with multiple routing objectives.

Lastly, most heuristic algorithms developed in the past have focused on solving the shortest path problem in networks where link travel costs are static and deterministic. In a realistic road traffic environment, however, travel times often change by time and have random variations. It is increasingly recognized that such dynamic and stochastic variations need to be explicitly considered in many transportation applications. Future research is needed to explore the possibility of extending the various heuristic search strategies to dynamic and/or stochastic networks [63,64].

References

[1] Hart EP, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Transaction, System Science and Cybernetics 1968;SSC-4(2):100–7.
[2] Nilsson JN. Problem-solving methods in artificial intelligence. New York: McGraw-Hill; 1971.
[3] Newell A, Simon HA. Human problem solving. Englewood Cliffs, NJ: Prentice-Hall; 1972.
[4] Pearl J. Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Publishing Company; 1984.
[5] Guzolek J, Koch E. Real-time route planning in road network. Proceedings of VINS, September 11–13, 1989. Toronto, Ontario, Canada, p. 165–9.
[6] Kuznetsov T. High performance routing for IVHS. IVHS America 3rd Annual Meeting, Washington, DC, April, 1993.
[7] Ahuja RK, Magnanti TL, Orlin JB. Network flows: theory, algorithms, and applications. Upper Saddle River, NJ: 07458: Prentice-Hall; 1993.
[8] Gallo G, Pallottino S. Shortest path methods: a unifying approach. Mathematical programming study 1986;26:38–64.
[9] Dijsktra EW. A note on two problems in connexion with graphs. Numerische Mathematik 1959;1:269–71.
[10] Dantzig GB. On the shortest route through a network. Management Science 1960;6:187–90.
[11] Whiting PD, Hillier JA. A method for finding the shortest route through a road network. Operational Research Quarterly 1960;11:37–40.

[12] Dial BR. Algorithm 360: shortest path forest with topological ordering. Communications of the Association for Computing Machinery 1969;12:632–3.
[13] Tarjan ER. Data structure and network algorithms. Philadelphia: SIAM; 1983.
[14] Ford LR. Network flow theory. Report P-923, Rand Corp., Santa Monica, CA, 1956.
[15] Moore EF. The shortest path through a maze. Proceedings of the international symposium on theory of switching. Cambridge: Harvard University Press, 1957. p. 285–92.
[16] Ford LR, Fulkerson DR. Flows in networks. Princeton, NJ: Princeton University Press; 1962.
[17] Bellman RE. On a routing problem. Quarterly of Applied Mathematics 1958;16:87–90.
[18] Pape U. Implementation and efficiency of moore algorithms for the shortest root problem. Mathematical Programming 1974;7:212–22.
[19] Glover F, Klinkgman D, Philips N. A new Polynomial bounded shortest path algorithm. Operations Research 1985;33: 65–73.
[20] Gallo G, Pallottino S. Shortest path methods. In: Florian M, editor. Transportation planning models. Amsterdam: Elsevier Science Publishers; 1984. p. 227–56.
[21] Hung SM, Divoky JJ. A computational study of efficient shortest path algorithms. Computers and Operations Research 1988;15(6):567–76.
[22] Vuren VT, Jansen GRM. Recent developments in path finding algorithms: a review. Transportation Planning and Technology 1988;12:57–71.
[23] Cherkassky BV, Goldberg AV, Radzik T. Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming 1996;73(2):129–74.
[24] Zhan FB, Noon CE. Shortest path algorithms: an evaluation using real road networks. Transportation Science 1998;32(1): 65–73.
[25] Jacob R, Marathe M, Nagel K. A computational study of routing algorithms for realistic transportation networks. Journal of Experimental Algorithmics 1999;4:1–19.
[26] Fu L. Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks. Unpublished Ph.D. Dissertation, University of Alberta, Edmonton, Alberta, 1996.
[27] Karimi HA. Real-time optimal route computation: a heuristic approach. ITS Journal 1996;3(2):111–27.
[28] Korf RE. Depth-first iterative deepening: an optimal admissible tree search. Artificial Intelligence 1985;27(1):97–109.
[29] Lysgaard J. A two-phase shortest path algorithm for networks with node coordinates. European Journal of Operational Research 1995;87(2):368–74.
[30] Pohl I. Bi-directional search. Machine Intelligence 1971;6:127–40.
[31] Golden LB, Ball M. Shortest paths with euclidean distance: an explanatory model. Networks 1978;8:297–314.
[32] Sedgewick R, Vitter JS. Shortest path in euclidean graphs. Algorithmica 1986;1:31–48.
[33] Nicosia G, Oriolo G. An approximate A* algorithm and its application to the SCS problem. Theoretical Computer Science 2003;290(3):2021–9.
[34] Bander JL, White III CC. A heuristic search algorithm for path determination with learning. IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans 1998;28(1):131–4.
[35] Adler JL. A best neighbor heuristic search for finding minimum paths in transportation networks. Presented at the 77th transportation research board annual meeting, Washington DC, 1998.
[36] Korf RE. Real-time heuristic search. Artificial Intelligence 1990;42(2–3):189–211.
[37] Reinefeld A, Marsland TA. Enhanced iterative-deepening search. IEEE Transactions on Pattern Analysis and Machine Intelligence, July, 1994.
[38] Lark JW, White III CC, Syverson K. A best-first search algorithm guided by a set-valued heuristic. IEEE Transactions on Systems, Man, and Cybernetics 1995;25(7):1097–101.
[39] Nicholson JAT. Finding the shortest route between two points in a network. Computer Journal 1966;9:275–80.
[40] Dreyfus ES. An appraisal of some shortest path algorithms. Operations Research 1969;17:395–412.
[41] De Champeaux D, Sint L. An improved bi-directional heuristic search algorithm. Journal of Association for Computing Machinery 1977;24(2):177–91.
[42] De Champeaux D. Bidirectional heuristic search again. Journal of Association for Computing Machinery 1983;30(1): 22–32.
[43] Kwa J. BS*: an admissible bi-directional staged heuristic search algorithm. Artificial Intelligence 1989;42(2,3):189–212.

[44] Kaindl H, Kainz G. Bidirectional heuristic search reconsidered. Journal of Artificial Intelligence Research 1997;7: 283–317.
[45] Bander JL, White III CC. A new route optimization algorithm for rapid decision support. Proceedings of Vehicle Navigation Information Systems Conference, Detroit, MI; 1991. p. 709–28.
[46] Dillenburg JF, Nelson PC. Improving search efficiency using possible subgoals. Mathematical and Computer Modelling 1995;22(4–7):397–414.
[47] Plolya, 1945.
[48] Sacerdoti ED. Planning in a hierarchy of abstraction spaces. Artificial Intelligence 1974;5:115–35.
[49] Korf RE. Planning as search: a quantitative approach. Artificial Intelligence 1987;33(1):65–8.
[50] Timpf S, Volta GS, Pollock DW, Egenhofer MJ. Conceptual model of wayfinding using multiple levels of abstraction. In: Frank A et al., editor. Proceedings of the GIS - from space to territory. Springer; 1992. p. 348–67.
[51] Car A, Frank AU. General principles of hierarchical spatial reasoning—the case of way-finding. Proceedings of the sixth international symposium on spatial data handling, vol. 2, 1994. p. 646–64.
[52] Liu B. Route finding by using knowledge about the road network. IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans 1997;27(4):436–48.
[53] Chou YL, Romeijn HE, Smith RL. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. INFORMS Journal on Computing 1998;10(2):163–79.
[54] Jagadeesh GR, Srikanthan T, Quek KH. Heuristic techniques for accelerating hierarchical routing on road networks. IEEE Transactions on Intelligent Transportation Systems 2002;3(4):301–9.
[55] Jung S, Pramanik S. An efficient path computation model for hierarchically structured topographical road maps. IEEE Transactions on Knowledge and Data Engineering 2002;14(5):1029–46.
[56] Bovy PHL, Stern E. Rout choice: wayfinding in transport networks. Dordrecht: Kluwer Academic Publishers; 1990.
[57] Palmer ES. Hierarchical structure in perceptual representation. Cognitive Psychology 1977;9:441–74.
[58] Hirtle SC, Jonides J. Evidence of hierarchies in cognitive maps. Memory and Cognition 1985;13(3):208–17.
[59] Shapiro J, Waxman J, Nir D. Level graphs and approximate shortest path algorithms. Networks 1992;22:691–717.
[60] Car, Frank, 1999.
[61] Shekhar S, Fetterer A, Goyal B. Materialization trade-offs in hierarchical shortest path algorithms. Advances in Spatial Databases, Fifth International symposium, SSD'97. Proceedings. Lecture Notes in Computer Science, vol. 1262, Springer. ISBN 3-540-632238-7, 1997. p. 94–111.
[62] Huang YW, Jing N, Rundensteiner EA. A hierarchical path view model for path finding in intelligent transportation systems. GeoInformatica 1997;1(2):125–59.
[63] Chabini I, Lan S. Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks. IEEE Transactions on Intelligent Transportation Systems 2002;3(1):60–74.
[64] Wellman MP, Larson K, Ford M, Wurman PR. Path planning under time-dependent uncertainty. Proceedings of the eleventh conference on uncertainty in artificial intelligence (UAI-95). The association for uncertainty in AI, Morgan Kaufmann, August, 1995. p. 532–9.

Further Reading

[65] Bast H, Mehlhorn K, Schafer G, Tamaki H. A heuristic for Dijkstra's algorithm with many targets and its use in weighted matching algorithms. Algorithmica 2003;36:75–88.
[66] Car AH, Mehner, Taylor G. Experimenting with hierarchical wayfinding. Technical report 01. Department of Geomatics, University of Newcastle upon Tyne, 1999.
[67] Denardo EV, Fox BL. Shortest-route methods: 1. reaching, pruning and buckets. Operations Research 1979;27:161–86.
[68] Fu L, Rilett LR. Dynamic O-D travel time estimation by using artificial neural network. Proceedings of vehicle navigation and information system (VINS). 1995 Annual Meeting, Seattle, Washington, 1995.
[69] Gen M, Cheng R, Wang D. Genetic algorithms for solving shortest path problems. Proceedings of IEEE International Conference on Evolutionary Computation 1997;401–6.
[70] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Publishing Company; 1989.

[71] Huang Y, Jin N. Evaluation of hierarchical path finding techniques for ITS route guidance. Proceedings of the Annual Meeting of ITS America 1996;1:340–50.
[72] Ikeda T, Hsu MY, Imai H, Nishimura S, Shimoura H, Hashimoto T, Tenmoku K, Mitoh K. A fast algorithm for finding better routes by AI search techniques. Proceedings of IEEE Vehicle Navigation and Information Systems Conference 1994;291–6.
[73] Kanoh H. Real-time route selection using genetic algorithms for car navigation systems. Proceedings of IEEE International Conference on Intelligent Vehicles 1998;207–12.
[74] Kanoh H, Nakamura T. Knowledge based genetic algorithm for dynamic route selection. Proceedings of fourth international conference on knowledge-based intelligent engineering systems and allied technologies, vol. 2, 2000. p. 616–9.
[75] Kanoh H, Nakamura T. Route guidance with unspecified staging posts using genetic algorithm for car navigation systems. Proceedings of IEEE Intelligent Transportation Systems 2000;2000:119–24.
[76] Liu B. Intelligent route finding: combining knowledge, cases and an efficient search algorithm. Proceedings of the 12th international conference on artificial intelligence, 1996. p. 380–4.
[77] Liu B, Tay J. Using knowledge about the road network for route finding. Proceedings of IEEE 11th conference on artificial intelligence for applications, 1995. p. 306–12.
[78] Liu B, Choo SH, Lok SL, Leong SM, Lee SC, Poon FP, Tan HH. Integrating case-based reasoning, knowledge-based approach and Dijkstra algorithm for route finding. Proceedings of the tenth conference on artificial intelligence for applications, 1994. p. 149–55.
[79] Liu B, Choo SH, Lok SL, Leong SM, Lee SC, Poon FP, Tan HH. Finding the shortest route using cases, knowledge, and Djikstra's algorithm. IEEE Expert 1994;9(5):7–11.
[80] Pallottino S. Shortest-path methods: complexity interrelations and new propositions. Networks 1984;14:257–67.
[81] Park CK, Sung K, Doh S, Park S. Finding a path in the hierarchical road networks. Proceedings of IEEE Intelligent Transportation Systems Conference; 2001. p. 25–9.
[82] Passino KM, Antsaklis PJ. A metric space approach to the specification of the heuristic function for the A* algorithm. IEEE Transactions on Systems, Man, and Cybernetics 1994;24(1):159–66.
[83] Polya G. How to solve it. second ed., Princeton University Press; 1971.
[84] Rilett LR, Blumentritt C, Fu L. Minimum path algorithms for in-vehicle route guidance systems. Proceedings of the IVHS AMERICA conference; 1994.
[85] Scharnow J, Tinnefeld K, Wegener I. Fitness landscapes based on sorting and shortest paths problems. Lecture Notes in Computer Science 2002;2439:54–63.
[86] Zhao Y, Weymouth TE. An adaptive route guidance algorithm for intelligent vehicle highway system. Proceedings of the American control conference, vol. 3, 1991. p. 2568–73.