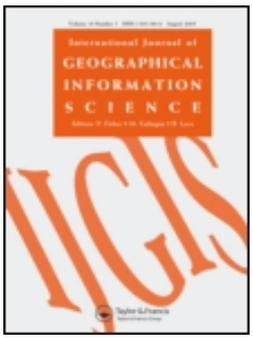

International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: https://www.tandfonline.com/loi/tgis20

Finding shortest paths on real road networks: the case for A*

W. Zeng & R. L. Church

To cite this article: W. Zeng & R. L. Church (2009) Finding shortest paths on real road networks: the case for A*, International Journal of Geographical Information Science, 23:4, 531-543, DOI: 10.1080/13658810801949850

To link to this article: https://doi.org/10.1080/13658810801949850

International Journal of Geographical Information Science
Vol. 23, No. 4, April 2009, 531–543

Taylor & Francis
Taylor & Francis Group

Research Article

Finding shortest paths on real road networks: the case for A*

W. ZENG† and R. L. CHURCH*‡
†Faculty of Information Engineering, China University of Geosciences, 388 Lumo Road, Hongshan District, Wuhan 430074, China
‡Department of Geography, University of California, Santa Barbara, CA 93106-4060, USA

(*Received 28 January 2008; in final form 28 January 2008*)

The problem of identifying the shortest path along a road network is a fundamental problem in network analysis, ranging from route guidance in a navigation system to solving spatial allocation problems. Since this type of problem is solved so frequently, it is important to craft an approach that is as efficient as possible. Based upon past research, it is generally accepted that several efficient implementations of the Dijkstra algorithm are the fastest at optimally solving the 'one-to-one' shortest path problem (Cherkassky *et al.* 1996). We show that the most efficient state-of-the-art implementations of Dijkstra can be improved by taking advantage of network properties associated with GIS-sourced data. The results of this paper, derived from tests of different algorithmic approaches on real road networks, will be extremely valuable for application developers and researchers in the GIS community.

Keywords: Shortest path algorithms; A* algorithm; Transportation networks

1. Introduction

One of the basic problems of network modeling is to find the shortest path from an origin to a destination. In 1955, George Dantzig presented a conference paper that included the first formulation of the shortest path problem. His paper was subsequently published in *Operations Research* (Dantzig 1957). Based upon that paper, Minty (1957) suggested a format for solving the shortest path problem using a network represented as a web of strings and knots, and Ford (1956) developed an algorithm to solve the shortest path problem in the presence of some negative arc lengths. Dijkstra (1959) followed the work of Minty and Ford with a new algorithm that appeared to be considerably more efficient and required less data storage. The algorithm of Dijkstra remains to this day one of the best approaches for optimally solving the simple shortest path problem where all arcs have nonnegative lengths.

In another early contribution, Hart *et al.* (1968) developed a search strategy, called A*, to solve for minimum cost paths. The A* approach differs from other methods as it incorporates an estimate of the cost of 'path-completion'. For certain classes of estimating functions, A* will find the optimal path.

Over the past 50 years, there has been continued and sustained interest in developing faster algorithms for solving the shortest path problem, because this problem is applied in a wide variety of areas including telephone call routing on

*Corresponding author. Email: church@geog.ucsb.edu

International Journal of Geographical Information Science
ISSN 1365-8816 print/ISSN 1362-3087 online
http://www.tandf.co.uk/journals
DOI: 10.1080/13658810801949850

communication networks and vehicle routing on road networks. The main objective of this paper is to provide a direct comparison of A* with what is considered to be the best implementations of shortest path algorithms in order to identify the most efficient approach when using data that are sourced from GIS. Over the last decade, there have been several notable attempts to compare shortest path algorithms (Cherkassky *et al.* 1996; Zhan and Noon 1998). Notably lacking from past work is a head-to-head comparison of A* and efficient implementations of Dijkstra. This is somewhat understandable, given that A* requires a function for estimating the completion cost of a path. Because of this, A* is not suitable for many shortest path applications; however, it is potentially ideal for an application using information sourced from GIS. One of the earliest tests of A* is due to Golden and Ball (1978) on Euclidean networks (i.e. a network defined in *r*-dimensional Euclidean space with the travel cost of each arc proportional to the straight line distance between the nodes it connects) where they found that A* expanded no more than about 10% of the nodes than what would be expanded by the Dijkstra algorithm. This possibility was also recognized by Shekhar *et al.* (1993, also see Shekhar and Fetterer 1996) where they conducted preliminary tests of A* applied to the street network of Minneapolis and compared it to the Dijkstra algorithm as well as a 'breadth first' search method. Their overall conclusion was unclear when they suggested that each algorithm appears to have some advantage over the others, depending upon the problem being solved and the resulting path length. While developing an approach to solve for shortest paths on German roads, Ertl (1998) essentially dismissed A* as a viable alternative when he stated that A*, at best, is modestly faster than the Dijkstra algorithm. Thus, virtually all previous research has either ignored the existence of A*, tested A* on a specialized problem or concluded that it is perhaps as good as or modestly better than the Dijkstra algorithm. In a recent review, Fu *et al.* (2006) stated that A* is competitive to Dijkstra-based approaches, but then emphasized the possible advantages of using A* as a heuristic instead of as an algorithm. Thus, it is impossible to discern from past work whether A* is a competitive technique to the most efficient encoding of the Dijkstra algorithm or a 'label-correcting' algorithm like the two queues method of Pallottino (1984). Our main objective is to demonstrate the relative value of A* in solving simple origin-to-destination shortest path problems on real road networks with respect to what is considered to be state-of-the-art encodings of Dijkstra's and label-correcting algorithms. The results of this paper, derived from real road networks, will be extremely valuable for application developers and researchers in the GIS community. In the next section, we present a detailed description of shortest path algorithms. Included in this is a description of the A* approach. We follow that with a discussion of how we implemented and tested A* and compared A* to other approaches. Then we present detailed comparisons using networks from two regions of California, USA (Los Angeles and Santa Barbara). We conclude with a set of recommendations in terms of further research and suggestions for embedding A* in GIS functionality.

2. Background

The literature on shortest path algorithms (SPA) is quite long and detailed. For more details than we provide here, the interested reader should consult Gallo and Pallottino (1986), Cherkassky *et al.* (1993), Dreyfus (1969) and Fu *et al.* (2006). The problem of concern is to find the shortest distance route from a pre-specified origin to a pre-specified destination on a road network that either is sourced from a GIS or

resides in a GIS. We make no distinction here as to whether the SPA is to be tightly coupled or loosely coupled with the GIS.

Before discussing details of various SPAs, it is important to first introduce some general notation about a network. Following Gallo and Pallottino (1986), we utilize the symbol $G(N,A;l)$ to represent a directed network with $n=|N|$ nodes, $m=|A|$ arcs and a length function $l{:}A{->}R$ where $R\geqslant 0$ (i.e. l is a set of arc lengths of real values greater than or equal to zero). We denote the length of a specific arc (i,j) by $l(i,j)$. For node i, we define arc set $\mathrm{FS}(i)=\{(i,\ j)|\mathrm{arc}(i,\ j)\in A\}$ as its forward star, and $\mathrm{SUC}(i)=\{j|\mathrm{arc}(i,j)\in A\}$ as the set of i's successors. The forward star is a commonly used data structure which is a list of those arcs that are rooted at node i and are directed to other nodes. We refer to the source of a shortest path or the root of a shortest path tree (SPT) as node s. In the context of the one-to-one shortest path problem, we refer to the destination as node t.

Gallo and Pallottino (1986) stated that almost all SPT algorithms of practical interest can be derived from one single prototype method. This prototype can be structured as follows:

Step 1: Begin with any directed tree T rooted at s and for each $v\in N$, let $d(v)$ be the length of the path from s to v in T;
Step 2: Let $(i,j)\in A$ be an arc for which $d(i)+l(i,j)<d(j)$, then adjust the vector d by setting $d(j)=d(i)+l(i,j)$, and update the tree T by replacing the current arc incident into node j by the new arc (i,j);
Step 3: Repeat step 2 until $d(i)+l(i,j)\geqslant d(j)$ for every $(i,j)\in$ A.

The basic idea behind this prototype is that one first begins with a tree T rooted at origin s and then iteratively updates this tree by replacing the arcs until all the arcs in the tree meet Bellman's optimality conditions (Bellman 1958), which is to say that for every arc (i,j), the path length from origin s to node i plus the length of arc (i,j) is no less than the path length from s to j.

Most of the shortest path algorithms follow the above prototype. In fact, all of the SPAs that are considered to be efficient in past tests can be defined within the prototype structure. The basic differences only occur in determining the order in which the improvements of step 2 are made and in the data structure used to store the list of candidates (arcs or nodes) for possible improvement. The famous Dijkstra algorithm is also consistent with this prototype. It applies a best-first selection strategy at step 2. Essentially, with a best first strategy, nodes of the shortest path tree are identified in order of the distance from the origin. Thus, the optimality criterion is satisfied in order of distance from the origin. We will denote the Gallo and Pallottino prototype as the *GP algorithm class* for the SPT problem, which includes all label-setting algorithms such as Dijkstra (1959), label correcting algorithms such as Pallottino's algorithm with two queues (TWO-Q) and threshold based algorithms such as Glover and Klingman (1985).

One of the most comprehensive reviews and tests of SPAs was published by Cherkassky *et al.* (1996). In that research, a number of algorithms were tested along with several proposed refinements. To their credit, they attempted to produce a level playing field, where all tested algorithms were coded in C with a concerted effort to make each as efficient as possible. In addition, all code was made available as open source. They tested 17 implementations of SPAs on a number of randomly generated networks and concluded that no single algorithm consistently beat all others over all problem classes. Overall, they suggested that the Dijkstra algorithm

implemented with a double-level bucket structure (DIKBD) is the best algorithm for networks with nonnegative arc lengths. The double-bucket structure represents the approach used to track and identify the next node to add to the shortest path tree (i.e. the next closest node to the origin). Using the open source codes written by Cherkassky *et al.* (1996), Zhan and Noon (1998) conducted an evaluation of 15 of the 17 algorithms on a variety of real road networks. They concluded that TWO-Q, DIKBD and the Dijkstra algorithm incorporating approximate buckets (DIKBA, which is a different variant of the bucket approach than DIKBD) are the three fastest one-to-all shortest path algorithms. In a subsequent study, Zhan and Noon (2000) compared these three algorithms for the one-to-one shortest path problem on 10 different road networks. They suggested that DIKBA is the best choice for the case when shortest paths are somewhat short and that Pallottino's TWO-Q is the best choice in situations where the shortest paths are relatively long. Given the work of Cherkassky *et al.* (1996) and Zhan and Noon (1998, 2000), one can conclude that the top three candidates for SPA application on real road networks are two versions of Dijkstra's algorithm (DIKBA and DIKDB), and Pallottino's TWO-Q algorithm. This past research, although meticulous, falls short in terms of three major areas. First, all real networks that were tested can be considered relatively small when compared to many GIS networks. Second, all tests performed on large networks involved random graphs, which lack geographical integrity. These two factors together could certainly skew the overall results and therefore the conclusions as well. Third, no attempt was made to test A* as a viable approach. Therefore, a balanced test of the top SPA candidates does not exist and conclusions drawn in the previous work may be misguided. In the next section, we will introduce the A* approach. We will also provide what we think is a plausible explanation as to why it has been ignored in previous tests of solving the one-to-one shortest path problem.

3. The A* heuristic

The A* algorithm was originally presented by Hart *et al.* (1968). It was designed to solve the shortest path problem between an origin and a destination. Suppose that in addition to the network of nodes and arcs, we have information in which we can estimate the distance from any one node to the destination. Formally, let $h(i)$=the estimate of completing the path from node i to the destination node t. One candidate for $h(*)$ is the Euclidean distance measure. The Euclidean distance measure is always a lower bound on what it would take in distance to complete a path from i to node t on a Cartesian plane (for a large region, it would be the distance of the great circle arc). The A* algorithm for solving the one-to-one shortest path problem can be described as follows:

Step 1: Begin by setting $d(i)=+\infty$ for each $i \in N$; next set $g(s)=0$ and $d(s)=g(s)+h(s)$; finally, let $S=\{s\}$;
Step 2: Node selection: identify node $v \in S$ where $d(v) \leqslant d(i)$ for all $i \in S$; set $S=S-\{v\}$;
Step 3: Stopping criteria: if $v=t$, stop as the shortest path to destination t has been found; otherwise go to step 4;
Step 4: Expanding: for each node w where arc $(v,w) \in A$, if $g(w)>g(v)+l(v, w)$ then update $g(w)=g(v)+l(v,w)$, set $d(w)=g(w)+h(w)$ and when $w \notin S$ let $S=S+\{w\}$; after all updates have been made go to step 2.

Like the Dijkstra algorithm, the A* algorithm maintains a set S of candidate nodes (nodes which have yet to be selected as the next closest node to the origin) and

applies a best-first method to select from this list the next node to expand/scan. When viewed from a myopic perspective, A* is exactly the same as Dijkstra's; however, A* also includes a forward-looking component, which is an estimate of the length to complete the path to the destination from a specific node. When node i is placed on the candidate list S, the A* algorithm establishes a temporary distance label with a function consisting of two parts: $d(i)=g(i)+h(i)$ where $g(i)$ is the estimated length of the shortest path from the origin s to node i, and $h(i)$ is the estimated length of the shortest path from i to the destination t. If $h(i)=0$ for all nodes i, then A* is essentially the same as Dijkstra's SPA. One can think of the estimated completion costs, $h(*)$, as a type of penalty, where nodes closer to the destination are penalized less than nodes that are further away from the destination. The $h(*)$ component of the node label helps direct the search space towards the destination, whereas the Dijkstra algorithm tends to expand the search space uniformly in all directions. Although both procedures identify and label nodes in order of distance, Dijkstra labels nodes in order of increasing distance from the origin and A* in order of the increasing distance from the origin plus the estimate of the completion distance to the destination.

Up to this point, we have not said much about the definition of $h(*)$ or its implementation. In fact, this is the key issue which controls both the efficiency and optimality of the A* algorithm. We have already defined the $h(i)$ function as an estimate for completing the path from node i to the destination, node t. Depending upon the definition of $h(*)$, A* is either a heuristic or an optimal algorithm. If for every node i, $h(i)$ does not exceed the actual shortest path distance from i to destination t, then $h(i)$ is said to be *admissible*. Hart *et al.* (1968) proved that if $h(i)$ is admissible, then A* is an optimal algorithm. Furthermore, if for every arc (i,j), $h(*)$ satisfies the triangle inequality: $h(i) \leqslant l(i,j)+h(j)$ and $h(t)=0$, then, $h(*)$ is said to be *consistent*. If $h(*)$ is consistent, every node enters set S at most once (Nilsson 1971; Pearl 1988). Consistency can help in reducing the total amount of node selection and expanding, thus making A* faster than what it otherwise would be. Therefore, A* is an algorithm when $h(*)$ is admissible, and labels are set and not changed when $h(*)$ is consistent.

Although A* could be used as a heuristic, depending upon the definition of $h(*)$, our intention here is to employ A* as an algorithm and solve for the optimal shortest path connecting an origin and a destination. Thus, our selection of an $h(*)$ function needs to be admissible. In the implementations of A* in this paper, we define $h(i)$ as the Euclidian distance between node i and t: $h(i)=[(x_i-x_t)^2+(y_i-y_t)^2]^{1/2}$, where (x_r, y_r) represents the coordinates for node r. It is obvious that for this definition, $h(*)$ is both admissible and consistent. We can compute $h(i)$ using positional coordinates for nodes i and t (data which are essential to GIS functionality in plotting a map of the network).

Like all efficient implementations of Dijkstra, an efficient implementation of A* needs a 'priority queue' for storage of the candidate (or opened) nodes (for example, double buckets or approximate buckets. The priority queue data structures that are adopted in label-setting algorithms (e.g. double buckets or approximate buckets), with some modifications, can also be used in A*. In our experiments, we utilize three kinds of data structures to implement priority queues for A* algorithm, namely, k-array heap, double-level buckets and approximate buckets. To do this, we utilized the shortest path codes that have been made available by Cherkassky *et al.* (1996). In the next section, we describe an environment in which

we test A* along with what is considered to be the best SPA, namely, DIKBA, DIKDB and TWO-Q.

The principal test of an SPA is the computational time that is taken to solve a given set of problems. For Cherkassky *et al.* (1996), the problem set was composed of a set of random graphs. Randomly generated graphs do not reflect any form of geographical reality. They are comprised of a set of nodes and arcs where the arcs and their distances are determined randomly. In general, searching for the shortest path on a random network can be a computational challenge, presenting in many cases a worst-case type of problem. Thus, from a theoretical point of view, such testing can be considered ideal. But from a practical point of view, the most efficient SP algorithm for random networks may not be the best algorithm for geographically realistic problems. It makes sense to test SPAs on real road networks, like the work of Zhan and Noon, in order to identify nuances between approaches as well as identify those algorithms that perform the best. Unfortunately, the past research has concentrated on analyzing algorithms primarily published in the operations research literature and ignored A*. In fact, even research reviews concentrate on SPAs presented in this literature (e.g. Gallo and Pallottino 1986). Even when SPAs are tested on geographically real networks, they are often limited to OR-based algorithms (Zhan and Noon 1998). Thus, the A* algorithm that is found in the artificial intelligence literature has been virtually ignored. To be fair, A* has probably been ignored by many as it relies on information that is not present in randomized graphs. But if the problem has a geographical basis, like a real road network, then a comparison of algorithms should include A*.

4. Experimental environment

In previous work, both Cherkassky *et al.* (1996) and Zhan and Noon (1998) tested a number of shortest path algorithms. What makes these two papers important is the fact that Cherkassky *et al.* spent considerable effort at developing efficient codes of 17 different SPA approaches involving combinations of algorithms (e.g. Dijkstra) and priority queue data structures (e.g. double buckets) and the fact that Zhan and Noon tested these same implementations on real road networks. Their conclusions were quite similar, in that they both found that the top three performers were the double bucket version of the Dijkstra algorithm (DIKDB), the approximate bucket version of the Dijkstra algorithm (DIKBA) and Pallottino's TWO-Q algorithm. Because of the meticulous work of Cherkassky, it is clear that these three techniques are candidates for the best optimal, one-to-one SPA applied to road networks.

Because Cherkassky *et al.* (1996) have placed in the public domain the codes they developed, we began our research with this library of codes. Cherkassky *et al.* coded every algorithm in C to run on the Sun UNIX operating system (Solaris) as a console program. We integrated the DIKBA, DIKDB, TWO-Q and the Dijkstra algorithm using *k*-Heap (DIKH) codes of Cherkassky *et al.* into one MS Windows graphical user interface (GUI) application. We included the DIKH method as it is perhaps the most popular implementation of Dijkstra, is considered to be efficient, is easy to implement and is incorporated in many existing GIS applications. All the codes were compiled with MS Visual Studio 6.0 using the O2 optimization option (maximizing speed). Since our tests involved repeated execution of the SPAs for selected sets of origin and destination pairs, we modified the original codes to eliminate cumulative memory occupancy.

Table 1. Algorithms Tested

Class	Abbreviation	Description
Dijkstra	DIKH	Dijkstra's algorithm with k-array heap
	DIKBD	Dijkstra's algorithm with double-level buckets
	DIKBA	Dijkstra's algorithm with approximate buckets
Graph Growth	Two-Q	Pallottino's algorithm with two queues
	ASH	A* algorithm with k-array heap
A*	ASBD	A* algorithm with double buckets
	ASBA	A* algorithm with approximate buckets

All of the original codes read network data from an ASCII file. This file included data fields for nodes, arcs and arc lengths, but contained no position information of the node or arc features. We augmented the file format to integrate a small amount of geographical information, i.e. the positional coordinates of nodes on the road network. We transferred road network data from a GIS database into the expanded file format, including positional coordinates for all nodes. In the process of transferring data from the GIS, data checks were made and each bidirectional link was mapped as two directed arcs. The arc lengths were magnified by a factor of 1000 and truncated to integers.

We then developed three different codes of A* based upon the type of priority-queue implementation. These new codes are A* with *k*-heap (ASH), A* with double buckets (ASBD) and A* with approximate buckets (ASBA). We took advantage of the Cherkassky *et al.* encodings of priority-queue structures in our codes, so the final codes represent extended versions of what was developed by Cherkassky *et al.* We developed these three codes in order to not only test A* against competitive SPAs, but also investigate which implementation would be best for A*. Table 1 summarizes the seven different algorithms tested in our experiments. This includes the three codes that Cherkassky *et al.* and Zhan and Noon found as the most efficient and three different versions of A* and the classical DIKH.

Finally, we used road network data from two counties in California, USA (Santa Barbara and Los Angeles) for testing the algorithms. Details concerning the two networks are shown in Table 2. The Santa Barbara network is a representative of a network of medium size and the Los Angeles network is considered to be relatively large. In tests conducted by Zhan and Noon (1998), no network exceeded a size of 100,000 nodes. Using the Los Angeles dataset provides better insight in solving problems on large networks. For each data set, we randomly selected 500 pairs of points, where each pair represented an origin node and a destination node. All algorithms were tested 500 times on each dataset, using the same sets of origins and destinations. All computational results were developed on a 2.8 GHz Pentium processor, using 2 GB of RAM and running Windows XP Professional version 5.1.

Table 2. Road network characteristics.

Abbreviation	County name	No. of nodes	No. of arcs	Arc/node ratio
SB	Santa Barbara	33,074	78,144	2.36
LA	Los Angeles	195,233	532,178	2.73

Table 3. Average performance for each tested algorithm in terms of average execution time (seconds) and ratios of execution times between A* and Dijkstra when employing the same priority queue structure.

	Data	
Algorithm	SB	LA
DIKH	0.00746	0.0603
DIBD	0.00805	0.03497
DIBA	0.00417	0.02663
TWO-Q	0.00875	0.08262
ASH	0.00473	0.02334
ASBD	0.00510	0.01531
ASBA	0.00281	0.01312
	Ratio of execution times (%)	
ASH/DIKH	63.4	38.7
ASBD/DIKBD	63.4	43.8
ASBA/DIKBA	67.4	49.3

5. Experimental results

For a given road network, we ran all seven algorithms for each of the 500 source-destination node pairs. We used mean execution time as the metric of algorithm efficiency, so for each individual source-destination pair, each algorithm was run 50 times and the mean runtime was calculated. Table 3 presents results for the 1000 different shortest path problems solved (500 problems on each of two datasets) using the seven different algorithms. The upper half of the table presents results in terms of mean execution time (seconds). Notice for both networks that the fastest routine on average is a version of A* (ASBA). Also note that TWO-Q has the worst performance on average. The lower half of the table presents results of A* relative to similarly encoded versions of Dijkstra. That is, ASH is compared to DIKH, ASBD is compared to DIKBD and ASBA is compared to DIKBA. The results presented in the lower half of the table indicate that a particular encoding of A* always represents an improvement over its counterpart encoding of Dijkstra.

For the medium size network of Santa Barbara, the double-level bucket implementation of the priority-queue tends to perform poorly for both Dijkstra and A*. However, for a larger network such as Los Angeles, performance of the double-level bucket implementation approaches the efficiency of the approximate bucket implementation. For the smaller dataset of Santa Barbara, the runtimes of A* are 60–70% of those of Dijkstra's. However, in the case of the Los Angeles data, A* implementations appear to be of obvious superiority, where solution times average less than half of what is needed for a similarly encoded Dijkstra algorithm. For the heap implementations, ASH spends just 38.7% of the time of Dijkstra employing *k*-heap.

The results demonstrate that A* is not only competitive, but appears to be superior to what has been considered best for real road networks. This means that spatial data in the form of coordinates of the nodes can be used to estimate completion costs in a form that is not only admissible (using Euclidean distance formula) and therefore optimal, but can save on overall computational effort, even when accounting for the added computational burden of making a significant number of Euclidean distance calculations. The reason for this is that the algorithm

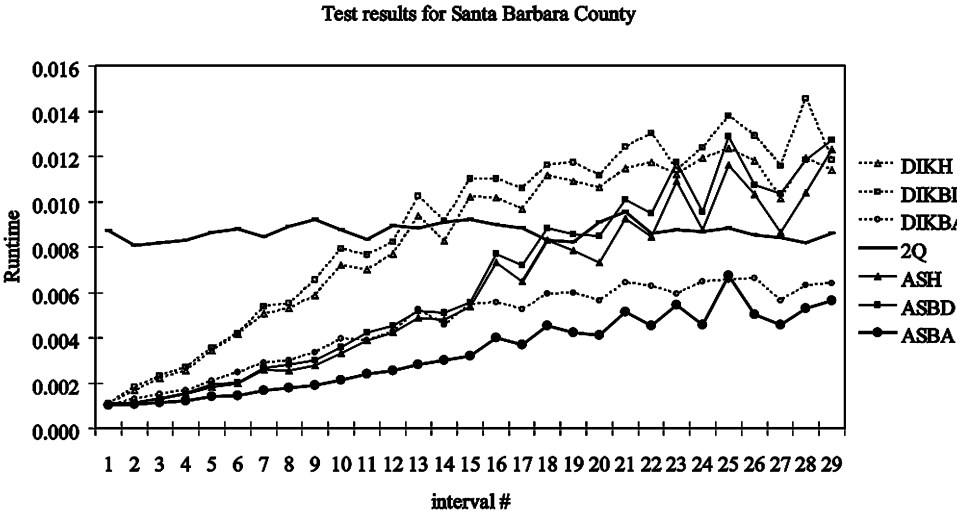

Figure 1. A plot corresponding to the dataset of Santa Barbara that illustrates the mean execution time as a function of the number of nodes within the shortest path.

tends to search among a considerably smaller set of nodes before identifying the shortest path to the destination than what is required by the Dijkstra algorithm.

Figures 1 and 2 present details based upon path complexity. These two figures give plots corresponding to the Santa Barbara and Los Angeles datasets, respectively. Each graph is a plot of the mean execution time as a function of the number of nodes within the shortest path (a surrogate measure for path complexity and length). For the 500 shortest paths computed for each dataset, the paths were analyzed in order to count the number of nodes traversed along the path. Nodes encountered within the paths ranged from 8 to 636 for Santa Barbara and from 25 to 696 for Los Angeles. We parceled out the resulting counts into 30 equal intervals,

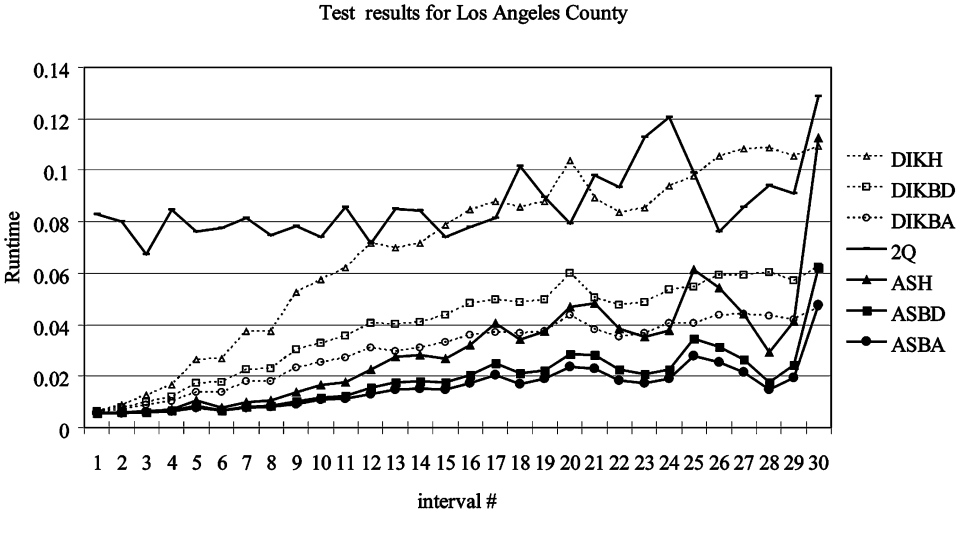

Figure 2. A plot corresponding to the dataset of Los Angeles that illustrates the mean execution time as a function of the number of nodes within the shortest path.

which were used for the plots in Figures 1 and 2. For each interval, the point represents the mean solution time for the respective algorithm for all paths assigned to that interval. Note that the plot for Santa Barbara depicts only 29 intervals, as one of the equal sized intervals did not include any paths. As the number of nodes used in the shortest path increases, the complexity and possibly the distance of the path increases. There is a notable trend for all algorithms, except TWO-Q, where execution time increases with increasing path complexity/distance.

As a label-correcting algorithm, the TWO-Q algorithm maintains a relatively stable performance with respect to the lengths of shortest paths. This is not a surprising result, because TWO-Q cannot stop until it has found all shortest paths from the origin node to all other nodes, rather than just the selected destination node. For the Santa Barbara data, the average time of TWO-Q is lower than DIKH and DIKBD for modest to high complexity paths. It is also better than ASH and ASBD for paths of high complexity. However, the average time of TWO-Q is always higher than those of DIKBA and ASBA. For the Los Angeles data, the performance of TWO-Q is a bit more erratic and tends to outperform only DIKH some of the time.

It is important to understand the phenomena depicted in the graph of Figure 2 for Los Angeles as path complexity/distance increases. At the far rightmost interval, you can observe that the solution times for the A* implementations tend to converge to their counterpart Dijkstra implementations. Although this may seem somewhat counterintuitive, it is to be expected. When the source node and destination node are so far apart (in terms of nodes used in the optimal path) that the shortest path between them almost forms a diagonal of the network (i.e. one of the longest, shortest paths possible), the search space of A* tends to approach that of Dijkstra, as both path searches cover most of the network. This reduces the effectiveness of using a function to estimate the completion costs of the path, i.e. $h(*)$, especially considering that it takes time to compute $h(*)$.

Overall, the results indicate that DIKBA is the best Dijkstra algorithm implementation. This is especially true since DIKBA's computing time gently increases as path complexity increases. Its performance is slightly better than ASH and ASBD for some of the less complex path problems found in the Santa Barbara dataset. The performance of DIKBA, however, is eclipsed by the faster implementations of A*. With respect to the three A* implementations, ASBA has the lowest average time as well as the lowest times among all seven tested algorithms on both networks. Given that Zhan and Noon concluded that DIKBA and DIKBD are best for real road networks and given that we have used essentially the same codes of DIKBA and DIKBD in our comparison with the A* implementations developed as a part of this research, the computational results presented in this paper clearly demonstrate that A* implementations are superior when applied to geographical data.

6. Conclusion

We tested four algorithms of the Gallo–Pallottino (GP) class of SPAs and three different implementations of the A* algorithm applied to road networks from two counties of California, USA. Of the four 'GP class' of SPAs, three of them are considered to be the fastest shortest path algorithm implementations tested in literature. From the experimental results presented here, we can conclude that on real road networks, the A* algorithm is more efficient than the GP class algorithms.

As the size of the network increases, A*'s performance tends to improve over that of the best of the GP class. Among the three implementations of A*, A* with approximate buckets (ASBA) has the best performance. Among the seven algorithms tested, ASBA virtually dominated all of the other algorithms for both networks.

Even though the Dijkstra algorithm is one of the earliest developed SPAs, it has continued to be proven as one of the most efficient methods based upon the use of an efficient method to track the search list. Because it has survived the test of time, it has been applied in many domains and has been the algorithm of choice within the GIS community for many years. Unfortunately, the Dijkstra process is oblivious to spatial layout and does not take advantage of the spatial attributes readily available in a GIS setting. The A* algorithm, however, can take advantage of spatial coordinates in trimming the search for the shortest path. It is this fact that allows the A* approach to outperform Dijkstra for real road networks. Our experiments demonstrate that on real road networks, A* outperforms the best implementations of the Dijkstra algorithm by a significant margin. In fact, a simple implementation of the A* algorithm can save more than half of the runtime when the network is relatively large. In that sense, one may conclude that implementations of Dijkstra in GIS tend to spend considerably more time than what is necessary for solving shortest path problems. Consequently, future GIS applications requiring a SPA should incorporate A* constructs.

We believe that we have conclusively determined that A* is one of the most efficient SPAs and outperforms other classic shortest path algorithms when spatial coordinates are present in the database. However, our implementation was somewhat simple in that it used the Euclidean distance metric as an estimate of completion costs. It may be possible to further improve the performance of the A* algorithm for solving the shortest path problem, by developing an improved lower bound heuristic function, $h(*)$. The more accurate the estimate of completion cost, the fewer nodes the A* algorithm is likely to visit and the faster the algorithm will be. It may also be possible to reduce the computations necessary to compute $h(*)$ (see the discussion of Hart *et al.* 1968 and Nilsson 1971) as well as to apply a better implementation of the priority queue list of open nodes. For example, Cherkassky *et al.* (1999) developed a structure called a 'hot queue' which is a combination of multi-level buckets and heap. A 'hot queue' implementation of A* might lead to even better performance than what has been achieved in this paper. Furthermore, other techniques such as the bi-directional search approach of Pohl (1971) and the strategies discussed in Fu *et al.* (2006) could prove to be very fruitful directions in future research.

Although we have shown that A* is an approach that should be part of the GIS tool box, it is not clear whether there may be other productive ways in which to take advantage of geometrical and geographical information of road networks during the state-space search. Road networks are planar (or nearly planar), spatially coherent (nodes which are close to some node are also close to each other) and are often somewhat sparse graphs. If these characteristics are encoded to direct a shortest path search, then perhaps A* can be outclassed by a spatially coherent algorithm. In fact, several studies have been conducted, which utilize geographical or geometrical characteristics of networks to speed up path search. Among them, the most remarkable is the radius constrained search method of Ertl (1998) and the reach method of Gutman (2004) to prune expanded nodes and reduce the effective search

space. Since both of these methods require a time-consuming preprocessing step to construct 'guidance' data for the network, they are probably limited to solving static networks.

Acknowledgements

This work was carried out while Dr Wen Zeng was visiting the Department of Geography at the University of California, Santa Barbara, CA, USA. We wish to acknowledge the generous support of the China University of Geosciences that was provided to Dr Wen Zeng for study leave at UCSB. We also would like to acknowledge Boris V. Cherkassky, Andrew V. Goldberg and Tomasz Radzik for making their encodings of SPAs available in the public domain.

References

BELLMAN, R., 1958, On a routing problem. *Quarterly of Applied Mathematics*, **16**, pp. 87–90.

CHERKASSKY, B.V., GOLDBERG, A.V. and RADZIK, T., 1993, Shortest path algorithms: theory and experimental evaluation. Technical Report STAN-CS-93-1480, Department of Computer Science, Stanford University, Stanford, CA.

CHERKASSKY, B.V., GOLDBERG, A.V. and RADZIK, T., 1996, Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming: Series A and B*, **73**, pp. 129–174.

CHERKASSKY, B.V., GOLDBERG, A.V. and SILVERSTEIN, C., 1999, Buckets, heaps, lists and monotone priority queues. *SIAM Journal on Computing*, **28**, pp. 1326–1346.

DANTZIG, G.B., 1957, Discrete-variable extremum problems. *Operations Research*, **5**, pp. 266–277.

DIJKSTRA, E.W., 1959, A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, pp. 269–271.

DREYFUS, S.E., 1969, An appraisal of some shortest-path algorithms. *Operations Research*, **17**, pp. 395–412.

ERTL, G., 1998, Shortest path calculation in large road networks. *OR Spektrum*, **20**, pp. 15–20.

FORD, L.R., 1956, Network flow theory. Technical Report P-932 (Santa Monica, CA: The Rand Corporation).

FU, L., SUN, D. and RILETT, L.R., 2006, Heuristic shortest path algorithms for transportation applications: state of the art. *Computers and Operations Research*, **33**, pp. 3324–3343.

GALLO, G. and PALLOTTINO, S., 1986, Shortest path methods: a unifying approach. *Mathematical Programming Study*, **26**, pp. 38–64.

GLOVER, F., KLINGMAN, D. and PHILLIPS, N., 1985, A new polynomial bounded shortest path algorithm. *Operations Research*, **33**, pp. 65–73.

GOLDEN, L.B. and BALL, M., 1978, Shortest paths with Euclidean distance: an exploratory model. *Networks*, **8**, pp. 297–314.

GUTMAN, R., 2004, Reach-based routing: a new approach to shortest path algorithms optimized for road networks. In *Proceeding of the 6th International Workshop on Algorithm Engineering and Experiments (ALENEX)*. New Orleans, LA, January 2004, pp. 100–111.

HART, E.P., NILSSON, N.J. and RAPHAEL, B., 1968, A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Science and Cybernetics*, **4**, pp. 100–107.

MINTY, G., 1957, A comment on the shortest route problem. *Operations Research*, **5**, pp. 724.

NILSSON, J.N., 1971, State-space search methods. In *Problem-solving Methods in Artificial Intelligence*. Chapter 3, pp. 43–79 (New York: McGraw-Hill).

PALLOTTINO, S., 1984, Shortest-path methods: complexity, interrelations, and new Propositions. *Networks*, **14**, pp. 257–267.

PEARL, J., 1988, On the discovery and generation of certain heuristics. In *Readings from the AI Magazine*, pp. 58–66 (Menlo Park, CA: American Association for Artificial Intelligence).

POHL, I., 1971, Bi-directional search. In *Machine Intelligence*, B. Meltzer and D. Michie (Eds), pp. 127–140 (New York: American Elsevier).

SHEKHAR, S., COYLE, M. and KOHLI, A., 1993, Path computation algorithms for advanced traveler information system. In *Proceeding of the International Conference on Date Engineering (IEEE Computer Society)*. Available online at: http://www.spatial.cs.umn.edu/paper_list.html (accessed 7 September 2006).

SHEKHAR, S. and FETTERER, A., 1996, Path computation in advanced traveler information systems. In *Proceeding 9th International IEEE Conference on Intelligent Transportation Systems* Available online at: http://www.spatial.cs.umn.edu/paper_list.html (accessed 7 September 2006).

ZHAN, F.B. and NOON, C.E., 1998, Shortest path algorithms: an evaluation using real road networks. *Transportation Science*, **32**, pp. 65–73.

ZHAN, F.B. and NOON, C.E., 2000, A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. *Journal of Geographic information and decision analysis*, **4**, pp. 1–13.