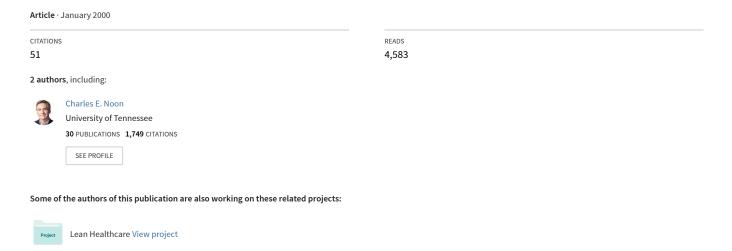# A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths

Some of the authors of this publication are also working on these related projects:

Project Lean Healthcare View project

A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths[*]

F. Benjamin Zhan
Department of Geography
Southwest Texas State University
San Marcos, TX 78666
e-mail: fz01@swt.edu

Charles E. Noon
Management Science Program
The University of Tennessee
Knoxville, TN 37996
e-mail: cnoon@utk.edu

Abstract: *It has been a general belief that label-setting algorithms are better choices than label-correcting algorithms for computing the shortest path distance between a source node and a destination node (one-to-one) on a network. The reason is that iterations in label-setting algorithms can be terminated whenever the destination node is scanned and permanently labeled. In this paper, a fast-performing implementation of each type of algorithm is selected based on the results of a recent study (Zhan and Noon 1998) and compared for computing one-to-one shortest paths using 10 large real road networks. The selected label-setting algorithm is Dijkstra's algorithm implemented with approximate buckets (DIKBA) and the selected label-correcting algorithm is Pallottino's graph growth algorithm implemented with two queues (TWO-Q). It is concluded that, in some situations, the TWO-Q algorithm is a better choice for computing one-to-one shortest paths on real road networks.*

Keywords: shortest path algorithms, networks, transportation, vehicle routing.

---

1   Introduction

The computation of shortest paths between different source and destination nodes on a network is a central and computationally-intensive task in many transportation and network analysis problems. With the advancement of Geographic Information Systems (GIS) technology and the availability of high quality network data, network and transportation analyses within a GIS environment have become a common practice in many areas (e.g., Ralston *et al.* 1994; Erkut 1996; Loureiro and Ralston 1996). These analyses often involve large road networks containing hundreds of thousands or even millions of nodes. As a consequence, high performance shortest path algorithms are necessary for accomplishing these types of analysis tasks. Depending on the nature of the problem, it is sometimes necessary to compute shortest paths from a source node to every other node (*one-to-all*) or from every node to every other node (*all-to-all*) on a network. Sometimes, it is only necessary to compute shortest paths from a source node to a destination node (*one-to-one*) or from a source node to some destination nodes (*one-to-some*) on a network. One-to-one or one-to-some shortest paths are often needed in several areas of transportation analysis, for instance, in vehicle routing, emergency response and product delivery, just to name a few.

Empirical evaluation of shortest path algorithms has been an extensively researched topic in the literature of operations research and management science (see, e.g., Dial *et al.* 1979; Glover *et al.* 1985; Gallo and Pallottino 1988; Cherkassky *et al.* 1996; and Zhan and Noon 1998). Among these evaluations, the most recent comprehensive evaluations are the ones conducted by Cherkassky *et al.*, and by Zhan and Noon. In both studies, the algorithms were categorized into two groups: *label-setting* and *label-correcting*. Both groups of algorithms are iterative and both employ the *labeling method* in computing one-to-all shortest paths (discussed in Section 2). The two groups of algorithms differ, however, in the ways in which they update the estimate (i.e., upper bound) of the shortest path distance associated with each node at each iteration and in the ways in which they converge to the final optimal one-to-all shortest paths. In label-setting algorithms, the final optimal shortest path distance from the source node to the destination node is determined once the destination node is scanned and permanently labeled. Hence, if it is only necessary to compute a one-to-one shortest path, then a label-setting algorithm can be terminated as soon as the destination node is scanned, and there is no need to exhaust all nodes on the entire network. In contrast, a label-correcting algorithm treats the shortest path distance estimates of all nodes as temporary and converges to the final one-to-all optimal shortest path distances at its final step when the shortest paths from the source node to all other nodes are determined. This characteristic of label-correcting algorithms implies there is no difference in the computational time used for computing a one-to-one shortest path and one-to-all shortest paths.

If a label-setting code and a label-correcting code had equivalent performance in computing one-to-all shortest paths, then clearly the label-setting code would be preferred in computing a one-to-one shortest path. Such equivalence, however, was not observed in the computational study of Zhan and Noon (1998). In that study, the best-performing label-

setting algorithm was observed to be over 50% slower than the best-performing label-correcting algorithm for computing one-to-all shortest paths on real road networks. It was concluded that Dijkstra's algorithm implemented with approximate buckets (DIKBA) is the best-performing algorithm in the group of label-setting algorithms, and Pallottino's graph growth algorithm implemented with two queues (TWO-Q) is the best-performing algorithm from the group of label-correcting algorithms. This finding leads to several interesting questions with respect to the performance of algorithms DIKBA and TWO-Q in computing one-to-one shortest paths.

1) For a given set of arbitrarily (e.g., randomly) chosen pairs of source and destination nodes, when algorithms DIKBA and TWO-Q are used to compute a one-to-one shortest path distance, which algorithm is faster on average?

2) Because algorithm DIKBA selects the node with the least shortest distance estimate as the next node to be scanned at each iteration, when a destination node is *sufficiently* close to a given source node, algorithm DIKBA has an apparent advantage over TWO-Q. On the other hand, if a destination node is *sufficiently* far away from a given source node, then TWO-Q should be faster in computing one-to-one shortest paths. This observation leads to the following question: Would there be a threshold distance from the source node within which algorithm DIKBA is highly likely to be faster, and beyond which algorithm TWO-Q is highly likely to be faster in computing one-to-one shortest paths? These two threshold distances are called the *lower end* and *upper end* threshold distances, respectively. In addition, would there be a threshold distance at which the speed of the two algorithms is about equal? This threshold distance is called the *middle* threshold distance.

3) If there exist such relative threshold distances, what is the likelihood that DIKBA is faster than TWO-Q for some given relative threshold distances?

This article aims to provide answers to the above three questions through an empirical comparison of the best-performing label-setting and label-correcting algorithms in computing one-to-one shortest paths. The exclusive focus of this article is on the relative speed of the two algorithms. Algorithms DIKBA and TWO-Q were tested on 10 large real road networks ranging in size from 35,793 nodes to 92,792 nodes. The algorithms are compared using C code implementations provided in the rigorously tested public domain codes described in Cherkassky *et al.* (1996). This study should contribute to the practical understanding of the behavior of these label-setting and label-correcting algorithms in computing one-to-one shortest paths on real road networks. The results from this article should be useful to practitioners in areas such as GIS, transportation, operations research and management science.

The rest of the article is organized as follows. In Section 2, an overview of the labeling method is given along with details on the tested algorithms. Computational test procedures and results are discussed in Section 3; and, finally a summary is given in Section 4.

2 Overview of the Algorithms

A network consisting of a set of nodes N and a set of arcs A is represented as a directed graph $G = (N, A)$, with $n = |N|$ denoting the number of nodes and $m = |A|$ denoting the number of directed arcs. Each arc (i, j) is represented as an ordered pair of nodes traveling from node i to node j. Each arc (i, j) has an associated numerical value, d_{ij}, representing the distance or cost incurred by traversing the arc. In this article, it is assumed that it is possible to travel between a pair of connected nodes i and j in both directions, and hence two directed arcs (i, j) and (j, i) are distinguished for each pair of connected nodes i and j. The one-to-all shortest paths are normally represented as a directed *out-tree* rooted at the source node s. This directed tree is referred to as a *shortest path tree*. Unless otherwise noted, a shortest path tree and one-to-all shortest paths are used interchangeably in this article.

Central to all shortest path algorithms discussed in this article is the *labeling Method* (Gallo and Pallottino 1988; Ahuja *et al.* 1993). Both the label-setting algorithms and label-correcting algorithms iteratively employ the labeling method to construct a shortest path tree. In the labeling method, an out-tree from the source node s is constructed and improved until no further improvements can be made. At termination, the shortest path from s to node i is represented by a unique path from s to i on the out-tree. While constructing and improving the out-tree, three pieces of information are maintained for each node i in the labeling method: a distance label $d(i)$, a parent node $p(i)$, and a node status $S(i) \in \{unreached, labeled, scanned\}$. As the labeling method progresses, the distance label $d(i)$ is used to record the upper bound of the s-to-i path distance in the out-tree. Upon termination of the algorithm, the distance label $d(i)$ represents the shortest path distance from node s to node i. The parent node $p(i)$ records the node that immediately precedes node i in the out-tree. An *unreached* node is a node whose distance label has a value of infinite (sufficiently large). A *labeled* node has a distance label that has been updated at least once, that is, its distance label is different from infinite. A node is said to be *permanently labeled* if its distance label represents the final and optimal shortest path from the source node. A node is considered *scanned* if it has undergone a scanning operation (explained below) and its distance label has subsequently not been updated.

The labeling method begins by setting $d(i) = \infty$, $p(i) = \emptyset$, and $S(i) = unreached$ for every node i. To initialize the method, information associated with a source node s are set as $d(s) = 0$ and $S(s) = labeled$. The method progresses by applying a *scanning* operation to labeled nodes commencing from s until either the destination node is reached and permanently labeled (DIKBA) or until all nodes have been scanned and permanently labeled (TWO-Q). The method terminates with a shortest path tree over either a subset of nodes (DIKBA) or the entire set of nodes (TWO-Q). In scanning node i, an attempt is made to lower the distance labels for any node j such that $(i, j) \in A$. If the distance label for node j can be lowered, the out-tree is adjusted by changing node j's parent node label by setting $p(j) = i$ and node j is considered labeled. Formally, the scanning operation for node i can be defined as follows:

Label-setting and label-correcting algorithms are based on the labeling method and

```
for all (i, j) ∈ A do
begin
     if d(i) + d_ij < d(j) then
     begin
          d(j) = d(i) + d_ij
          p(j) = i
          S(j) = labeled
     end
end
set S(i) = scanned
```

employ the scanning operation as described. The two algorithms differ primarily in the data structures used for managing the set of labeled nodes and selecting nodes for scanning. Brief overviews of the data structures for DIKBA and TWO-Q are given below. A more detailed description of the data structures and procedures related to these two algorithms can be found in Zhan (1997).

In the original Dijkstra algorithm, the node selected for scanning is a labeled node with the minimum distance label (Dijkstra 1959; Ahuja et al. 1993, p.109). A result of this selection criteria is that, once a node is scanned, it becomes permanently labeled and is never again selected for scanning. In Dijkstra's original implementation of the algorithm, the set of labeled nodes is managed as an unordered list. This is clearly a bottleneck operation since all labeled nodes must be checked during an iteration in order to select the node with the minimum distance label. A natural enhancement of the original Dijkstra algorithm is to maintain the labeled nodes in a data structure such that the nodes are either exactly or approximately sorted according to distance label. The *bucket* data structure is one such structure.

In the *approximate* bucket implementation of the Dijkstra algorithm (DIKBA), a bucket i contains those labeled nodes whose distance labels are within the range of $[i*\beta, (i+1)*\beta-1]$, where β is a chosen constant. The bucket structure is considered approximate since the values of the distance labels in a bucket are not exactly the same, but are within a certain range. Nodes within each bucket are maintained in a FIFO ordered list. Algorithm DIKBA requires a maximum of $(C/\beta)+1$ buckets and has worst case complexity of $O(m\beta + n(\beta + C/\beta))$, where C is the length of the longest arc. It is important to note that, in the DIKBA implementation, a node can be scanned more than once, but never more than β times. For example, a node can be scanned out of its current bucket and then become labeled during the scanning operation of another node in the same bucket. When all nodes within a current bucket have been completely scanned out, their distance labels become permanent. For the tested implementation of DIKBA, the bucket width parameter β was set in the same fashion as in Cherkassky *et al.* (1996), i.e. the number of buckets (C/β) was set up to, but not exceeding, 2^{11}.

Label-correcting algorithms manage the set of labeled nodes in a very different fashion.

5

In TWO-Q, the labeled nodes are stored in one of two ordered lists (or *queues*), Q_1 and Q_2. By definition, a node becomes *labeled* if its distance label is changed during a scanning operation. In the TWO-Q algorithm, a node that becomes labeled is treated differently depending on whether or not it has ever been scanned before. If it has been scanned before, it is assigned to the tail of Q_1. If it has never been scanned before, it is assigned to the tail of Q_2. The node selected for scanning is always the head node of Q_1 or, in the case that Q_1 is empty, the head of Q_2. When a node is scanned, it is removed from the set of labeled nodes. However, if the node's distance label is later changed by a scanning operation, the node is returned to the set of labeled nodes. For this reason, the nodes are not considered permanently labeled until the entire set of labeled nodes (consisting of Q_1 and Q_2) is empty. The worst case complexity of the TWO-Q algorithm is given as $O(n^2 m)$.

The differences in the running times associated with DIKBA and TWO-Q depend on the number of scans and the time required to select nodes for scanning. Dijkstra-based algorithms tend to have either exactly one or slightly greater than one (as in the general case of DIKBA) scan per permanently labeled node. The TWO-Q implementations can have considerably more scans per node, however, the effort required to select a node for scanning is minimal. In the case of DIKBA, however, buckets must be checked sequentially until one is found that contains a labeled node eligible for scanning. Having to check an excessive number of empty buckets can be costly in terms of wasted effort.

3   Computational Test and Results

Ten large real road networks were obtained for testing the algorithms. These 10 networks are state-level road networks compiled from U. S. Geological Surveys' Digital Line Graphs. These 10 states are Alabama (AL), Florida (FL), Georgia (GA), Iowa (IA), Louisiana (LA), Minnesota (MN), Missouri (MO), Mississippi (MS), Nebraska (NE), and South Carolina (SC). Each of the 10 networks consists of four levels of roads, including interstate highways, principal arterials, major arterials and one additional level of more detailed roads. An example of the 10 networks, Mississippi, is shown in Figure 1. The road networks were stored and maintained in the Arc/Info[1] GIS. The nodes, links and link-lengths of the networks were downloaded from Arc/Info into ASCII files. It was made sure that the networks were fully connected before they were downloaded to files. Some characteristics of the 10 networks are summarized in Table 1. The 10 networks cover a wide range of different sizes ranging from 35,793 nodes to 92,792 nodes. The arc to node ratios of the networks vary from 2.66 to 3.28.

(Figure 1 and Table 1 about here)

The two algorithms, namely DIKBA and TWO-Q, were coded in the C programming language. The C programs were the set of one-to-all shortest path C source codes provided by

---

[1]Arc/Info is a GIS developed and distributed by Environmental Systems Research Institute (ESRI), Redlands, California, USA.

Figure 1. The Mississippi road network used in the study.

Cherkassky *et al.* (1996) with only minor modifications. The input networks are represented using the forward-star data structure (Ahuja *et al.* 1993, pp.35-37). The programs were compiled with the GNU `gcc` compiler version 2.5.6 using the O4 optimization option. The experiments were conducted on a stand-alone SUN Sparc-20 workstation, model HS21 with a 125MHZ Hypersparc processor and 64 Megabytes of RAM running under the Solaris 2.4 environment. Because the C programs require input arc lengths to be integral, the arc lengths were multiplied by a scaling factor of 1000, and the resulting arc lengths were then truncated to integers as the input arc lengths to the C programs. Three sets of experiment were conducted. Each aims at answering one of the three questions introduced in Section 1.

3.1   Average Performance for Computing One-to-One Shortest Paths

In order to answer the first question, CPU times for both algorithms to compute one-to-one shortest paths must be obtained. Because computing the one-to-one shortest path between each possible pair of source and destination nodes on each network involves enormous computation time, only five sample sets of nodes were selected from each network. Each sample set consists of 100 randomly selected nodes. For each sample set, the following experiment was conducted. For algorithm DIKBA, each of the 100 randomly selected nodes in each sample set on each network was treated as a *source node* in turn, and the remaining 99 randomly selected nodes were treated as destination nodes. Thus, there are a total of 9900 pairs of source and destination nodes for each network in each sample set. The CPU time for computing the one-to-one shortest path between each of the 9900 pairs of source and destination nodes was obtained on each network. In order to obtain the CPU time, a shortest path tree rooted at each source node was constructed. The CPU time for computing a one-to-one shortest path was obtained by recording the CPU time used to reach and permanently label a destination node from the source node.

After the minimum, maximum, mean CPU time, the standard deviation and the ratio between the maximum and mean CPU time for computing one-to-one shortest paths on each network were obtained for each sample set of nodes, the average of these five parameters from the five sets of sampled nodes were then computed. These average results are summarized in Table 2. As an example, let us take a look at the results associated with the Louisiana network. Among the five sets of results, the average minimum is 0.0167 seconds, the average maximum is 0.2933 seconds, the average mean is 0.1177 seconds, the average standard deviation is 0.0591 seconds, and the average maximum to mean ratio is 2.49.

(Table 2 about here)

Similar results were obtained for algorithm TWO-Q. The only difference is that the CPU time used for computing one-to-all shortest paths is treated as equivalent to the CPU time used for computing a one-to-one shortest path because a complete shortest path tree rooted at the source node must be constructed in both cases when using TWO-Q. Again, for each of the five sample sets, each of the 100 randomly selected nodes was treated as a source node in turn,

7

and the CPU time for the algorithm to construct each of the 100 shortest path trees on each network was obtained. For the part associated with algorithm TWO-Q, each row in Table 2 represents the results associated with each of the 10 networks.

For each algorithm, the overall average of the minimum, maximum, mean CPU time, standard deviation and the maximum to mean ratio across all networks were computed. The ratios between the values of the five parameters of the two algorithms for computing a one-to-one shortest path were also obtained (bottom of Table 2). It can be easily seen that, on average, the label-setting algorithm DIKBA is approximately 14% slower than the label-correcting algorithm TWO-Q for computing a one-to-one shortest path for a set of pairs of randomly selected source and destination nodes.

3.2   Lower End, Middle and Upper End Relative Threshold Distances

In order to answer the second question, five sets of randomly sampled nodes were generated from each network again. Each set contained 100 randomly selected nodes. For each set of nodes, the following experiment was carried out. First, each of the 100 nodes was treated as a source node in turn, and the CPU time for computing a shortest path tree rooted at each source node on each network was obtained for algorithm TWO-Q. That is, 100 shortest path trees were constructed on each network using TWO-Q, and the CPU time for TWO-Q to construct each of the 100 shortest path trees was obtained for each network. Second, algorithm DIKBA was tested against TWO-Q. Similar to the first step, each of the 100 randomly selected nodes in each set of nodes on every network was treated as a source node in turn. For each source node, a shortest path tree was constructed using algorithm DIKBA, and the CPU time used to permanently label each of the 99 destination nodes from the source node in question was recorded. Also each of the 99 one-to-one shortest path distances was obtained. In addition, the longest shortest path distance on each shortest path tree rooted at the current source node was obtained. This distance is called the *longest distance.*

The three types of data associated with each source node are: the longest distance, the list of 99 CPU times, and the list of 99 one-to-one shortest path distances. These two lists were sorted together using the CPU time as a sorting key. Because DIKBA permanently labels the node with the minimum distance label at each iteration, the values of the elements in the list of 99 one-to-one shortest path distances should increase as the values in the list of 99 CPU times increase. Hence, there is always a node whose associated CPU time is immediately larger than the CPU time used by algorithm TWO-Q to construct a shortest path tree rooted at the same source node. The shortest-path distance associated with this node is called the *threshold distance* and is reported here in proportion to the longest distance; for example a threshold distance half the longest distance is reported as a *threshold ratio* of 0.5. For each set of 100 randomly selected nodes, 100 such threshold ratios were obtained on each network.

For each set of sampled nodes on each network, the minimum, mean, median, maximum, and standard deviations of the threshold ratios were computed along with a maximum to

minimum ratio. This process was repeated for all five sets of nodes on each network, and the average of the values of the above six parameters during the five runs are tabulated in Table 3.

(Table 3 about here)

The overall averages of the six parameters across all networks were then computed (bottom of Table 3). The results are relatively consistent across all networks. The overall average of the minimum (*low end*) threshold ratio is 0.24. For the sake of simplicity, the low end threshold ratio is assumed to be 0.20. This choice is an adequate one as will be seen in the results obtained in the next subsection. Similarly, the mean *middle end* and maximum *upper end* threshold ratios are 0.40 and 0.70. In summary, it can be said that when the threshold ratio is 0.20 or less, DIKBA is highly likely to be faster than TWO-Q. The speed of the two algorithms is roughly equivalent when the threshold ratio increases to 0.40. At threshold ratios above 0.70, TWO-Q is highly likely to run faster than DIKBA.

3.3 The Likelihood for DIKBA to be Faster Than TWO-Q for Given Threshold Ratios

In order to answer the third question, the likelihood for DIKBA to be faster than TWO-Q across six threshold ratio values was computed. These six threshold ratios are 0.20, 0.30, 0.40, 0.50, 0.60 and 0.70, and they were chosen at equal intervals between the lower end and upper end ratios identified in the last subsection. For each network, five additional sets of nodes were drawn randomly, and each set also consists of 100 nodes. For each set of nodes on every network, each node was treated as a source node in turn, and 100 shortest path trees were constructed. The likelihood that DIKBA is faster than TWO-Q within each of the six threshold ratios was computed. The averages of the five resulting likelihoods corresponding to particular threshold ratios from the five sets of nodes were obtained and are summarized in Table 4.

(Table 4 about here)

Based on the data shown in Table 4, one can observe that: a) when the threshold ratio is 0.20 or less, there is at least a 98% chance that DIKBA is faster than TWO-Q; b) when the threshold ratio reaches 0.40, there is a 50% chance for DIKBA to be faster than TWO-Q; c) when the threshold ratio increases to 0.50, there is only a less than 20% chance for DIKBA to be faster than TWO-Q; d) when the threshold ratio approaches 0.60, there is a less than 6% chance for DIKBA to run faster than TWO-Q; and e) when the threshold ratio exceeds 0.70, the chance for DIKBA to be faster than TWO-Q is less than 2%.

4   Summary, Discussion and Recommendation

The focus of this paper has been a comparison of two high-performance shortest path algorithms for solving shortest distance problems on large, real-world networks. Many problems faced by practitioners (for example, time-window vehicle routing) are concerned with finding shortest *time* paths through networks. These problems are typically modeled by assigning arc travel speeds (usually as a function of road class) and then computing arc travel time to be used as the arc length metric. Both DIKBA and TWO-Q can be used for solving shortest time path problems, however, the relative speeds of the two algorithms on such problems may be different than observed in this study. In particular, TWO-Q may be negatively affected by an increase in the number of scans as was observed by Cherkassky *et al.* (1996) for networks which lacked triangle inequality. Problems in which arc travel times vary as a function of time (real time dispatching or IVHS applications) present additional complexities that must be taken into account when selecting a solution approach (see Horn (2000) and Kaufman and Smith (1993)).

Based on the results presented in Section 3, the following answers to the three questions posed in Section 1 can now be summarized.

1) For a set of randomly chosen pairs of source and destination nodes, algorithm TWO-Q is generally about 14% faster than DIKBA on average in computing a one-to-one shortest path.

2) The lower end, middle and upper end threshold ratios are 0.20, 0.40 and 0.70, respectively. These ratios suggest that the distance range from a source node to a destination node within which DIKBA is faster than TWO-Q is less than the distance range within which TWO-Q is faster than DIKBA on any given network.

3) For some given threshold ratios, the following conclusions can be drawn: a) When the ratio is 0.20 or less, it is highly likely (98%) that DIKBA is faster than TWO-Q for computing one-to-one shortest path distances on real road networks; b) when the ratio reaches 0.40, there is a 50% chance for DIKBA to be faster than TWO-Q; c) when the ratio approaches 0.60, there is a 20% chance for DIKBA to run faster than TWO-Q; and d) when the ratio exceeds 0.70, there is a 98% chance for algorithm DIKBA to be slower than TWO-Q.

These results can be easily generalized to the situation of computing *one-to-some* shortest paths. Clearly, when the shortest path distances of *all* destination nodes have a threshold ratio of 0.40 or less, DIKBA would likely be faster than TWO-Q, on average. However, when *at least one* of the destination nodes falls beyond a threshold ratio of 0.40, TWO-Q would likely be faster. Overall, there is no clear advantage to use DIKBA instead of TWO-Q for computing one-to-one shortest paths. The reason is that the threshold ratio at which the speed of the two algorithms is about equal is 0.40. This middle threshold ratio favors algorithm

TWO-Q when the distribution nature of network nodes in a two-dimensional space is taken into consideration.

The above argument can be illustrated by the following simplified example. Suppose that there is a network covering a circular area, and the circle has a radius of one unit. The most favorable condition for DIKBA would be that the source node is always at the center of the circle. Furthermore, let us assume that all nodes on the network are distributed uniformly in the area, and that the distance between any pair of nodes increases linearly from the center of the circle toward its edge. If the possibility for any node (other than the source node) being the destination node is equal, then the percentage of destination nodes for which DIKBA is almost absolutely faster is only 4% (corresponding to the lower end threshold ratio of 0.20); the percentage of destination nodes for which DIKBA is about as fast as TWO-Q is only 16% (corresponding to the middle threshold ratio of 0.40); and, the percentage of destination nodes for which TWO-Q is almost certainly faster than DIKBA is 51% (corresponding to the upper end threshold ratio of 0.70).

Based on the answers to the three questions and the foregoing discussions, it is concluded that the TWO-Q algorithm is generally a better choice if one faces the task of computing *one-to-one* or *one-to-some* shortest path distances. Therefore, our recommendations are: a) When there is no prior knowledge about the network-based shortest path distance between a source node and a destination node relative to the longest shortest path distance on a shortest path tree rooted at the source node, use TWO-Q to compute one-to-one (some) shortest paths on a network; and, b) DIKBA may be chosen if it is known that the shortest path distance from a source node to a destination node is within 40% of the longest shortest path distance on the shortest path tree rooted at the source node. The choice of TWO-Q is perhaps most appropriate in situations where it is necessary to compute a large number of relatively long routes. An example would be in setting up distribution system optimization models. An obvious choice for DIKBA would be situations in which the paths are short relative to the extent of the network. An example would be the case of a centralized dispatching system for emergency response. An emergency call (destination) would be assigned to a close facility and then a path would be calculated. Given the variety of applications and the relative closeness in performance, it is perhaps prudent to test both types of algorithms before making a selection.

Acknowledgment

References

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993) *Network Flows: Theory, Algorithms and Applications.* Englewood Cliffs: Prentice Hall.

[2] Cherkassky, B. V., Goldberg, A. V., and Radzik, T. (1996) Shortest Paths Algorithms: Theory and Experimental Evaluation, *Mathematical Programming*, 73, 129-174.

[3] Dial, R. B., Glover, F., Karney, D., and Klingman, D. (1979) A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees, *Networks*, 9, 215-248.

[4] Dijkstra, E. W. (1959) A Note on Two Problems in Connection with Graphs, *Numeriche Mathematik*, 1, 269-271.

[5] Erkut, E. (1996) The Road Not Taken, *ORMS Today*, 23(6), 22-28.

[6] Gallo, G., and Pallottino, S. (1988) Shortest Paths Algorithms, *Annals of Operations Research*, 13, 3-79.

[7] Glover, F., Klingman, D., and Philips, N. (1985) A New Polynomially Bounded Shortest Paths Algorithm, *Operations Research*, 33, 65-73.

[8] Horn, M. E. (2000) Efficient Modelling of Travel in Networks with Time-Varying Link Speeds, *Networks*. (forthcoming)

[9] Kaufman, D. E., and Smith, R. L. (1993) Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application, *IVHS Journal*, 1(1), 1-11.

[10] Loureiro, C. F., and Ralston, B. (1996) Investment Selection Model for Multicommodity Multimodal Transportation Networks, *Transportation Research Record*, 1522, 38-46.

[11] Ralston, B., Tharakan, G., and Liu, C. (1994) A Spatial Decision Support System for Transportation Policy Analysis, *Journal of Transport Geography*, 2, 101-110.

[12] Zhan, F. B. (1997) Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures, *Journal of Geographic Information and Decision Analysis*, 1(1), 69-82.

[13] Zhan, F. B., and Noon, C. E. (1998) Shortest Path Algorithms: An Evaluation Using Real Road Networks, *Transportation Science*, 32, 65-73.

List of Captions

Figure 1. The Mississippi road network used in the study.

Table 1. Characteristics of the 10 road networks.

Table 2. CPU time used by algorithms DIKBA and TWO-Q for computing one-to-one shortest paths between randomly selected source and destination nodes on the 10 real road networks (CPU time in seconds).

Table 3. Lower end, middle and upper end threshold ratios. Each ratio represents a threshold distance as a proportion of the longest distance. A threshold distance is the shortest path distance on a shortest path tree within which algorithm DIKBA is faster than algorithm TWO-Q. The longest distance is the longest shortest path distance from the source node on a one-to-all shortest path tree.

Table 4. The likelihood that DIKBA is faster than TWO-Q for some given threshold ratios.

Table 1: Characteristics of the 10 road networks.

Network name (abbreviation)	Number of nodes	Number of arcs	Arc /Node Ratio	Maximum arc length	Mean arc length	Stnd. Dev. of arc lengths
Louisiana (LA)	35793	98880	2.76	0.3607	0.0139	0.0153
Mississippi (MS)	39986	120582	3.02	0.2321	0.0154	0.0140
Nebraska (NE)	44765	146476	3.28	0.5283	0.0180	0.0157
Florida (FL)	50109	133134	2.66	0.4162	0.0112	0.0153
South Carolina (SC)	52965	149620	2.82	0.1636	0.0100	0.0102
Iowa (IA)	63407	208134	3.28	0.2698	0.0157	0.0092
Minnesota (MN)	65491	209340	3.20	0.4109	0.0172	0.0141
Alabama (AL)	66082	185986	2.82	0.2982	0.0114	0.0124
Missouri (MO)	67899	204144	3.00	0.2125	0.0155	0.0133
Georgia (GA)	92792	264392	2.84	0.1742	0.0105	0.0001

Note: arc lengths are in decimal degrees of a geographic coordinate system.

Table 2: CPU time used by algorithms DIKBA and TWO-Q for computing one-to-one shortest paths between randomly selected source and destination nodes on the 10 real road networks (CPU time in seconds).

Networks and No. of Nodes		Minimum	Maximum	Mean	STDEV	Max/Mean
Algorithm DIKBA						
Louisiana (LA)	(35793)	0.0167	0.2933	0.1177	0.0591	2.49
Mississippi (MS)	(39986)	0.0167	0.4367	0.1798	0.0942	2.43
Nebraska (NE)	(44765)	0.0233	0.5367	0.2406	0.1255	2.23
Florida (FL)	(50109)	0.0300	0.4700	0.1982	0.0972	2.37
South Carolina (SC)	(52965)	0.0300	0.6100	0.2675	0.1391	2.28
Iowa (IA)	(63407)	0.0467	0.8133	0.3685	0.1925	2.21
Minnesota (MN)	(65491)	0.0433	0.7900	0.3589	0.1857	2.20
Alabama (AL)	(66082)	0.0467	0.7433	0.3220	0.1655	2.31
Missouri (MO)	(67899)	0.0467	0.8200	0.3716	0.1932	2.21
Georgia (GA)	(92792)	0.0633	1.1133	0.5047	0.2560	2.20
Algorithm average		**0.0363**	**0.6627**	**0.2930**	**0.1508**	**2.29**
Algorithm TWO-Q						
Louisiana (LA)	(35793)	0.1000	0.1533	0.1231	0.0116	1.24
Mississippi (MS)	(39986)	0.1100	0.1800	0.1472	0.0156	1.22
Nebraska (NE)	(44765)	0.1800	0.3767	0.2209	0.0335	1.70
Florida (FL)	(50109)	0.1800	0.2467	0.2169	0.0151	1.14
South Carolina (SC)	(52965)	0.1900	0.2667	0.2398	0.0145	1.11
Iowa (IA)	(63407)	0.2567	0.3100	0.2828	0.0110	1.10
Minnesota (MN)	(65491)	0.2400	0.3733	0.2970	0.0299	1.26
Alabama (AL)	(66082)	0.2633	0.3600	0.3067	0.0210	1.17
Missouri (MO)	(67899)	0.2633	0.3433	0.3052	0.0162	1.12
Georgia (GA)	(92792)	0.3667	0.4833	0.4365	0.0257	1.11
Algorithm average		**0.2150**	**0.3093**	**0.2576**	**0.0194**	**1.22**
Ratio (DIKBA/TWO-Q)		**0.17**	**2.14**	**1.14**	**7.78**	**1.88**

Table 3: Lower end, middle and upper end threshold ratios. A threshold ratio is defined as the ratio of the threshold distance to the longest distance. A threshold distance is the shortest path distance on a shortest path tree at which algorithm TWO-Q becomes faster than DIKBA. The longest distance is the longest shortest path distance from the source node on a shortest path tree.

Networks and No. of Nodes		Minimum	Mean	Median	Maximum	STDEV	Max/Mean
Louisiana (LA)	(35793)	0.22	0.47	0.46	0.89	0.14	1.91
Mississippi (MS)	(39986)	0.25	0.42	0.41	0.72	0.08	1.72
Nebraska (NE)	(44765)	0.16	0.36	0.31	0.79	0.16	2.21
Florida (FL)	(50109)	0.23	0.43	0.40	0.89	0.13	2.06
South Carolina (SC)	(52965)	0.22	0.43	0.44	0.69	0.10	1.61
Iowa (IA)	(63407)	0.27	0.39	0.39	0.55	0.06	1.42
Minnesota (MN)	(65491)	0.22	0.35	0.34	0.62	0.08	1.74
Alabama (AL)	(66082)	0.22	0.41	0.39	0.68	0.12	1.67
Missouri (MO)	(67899)	0.28	0.43	0.43	0.59	0.07	1.36
Georgia (GA)	(92792)	0.28	0.42	0.42	0.58	0.06	1.39
Overall Average		**0.24**	**0.41**	**0.40**	**0.70**	**0.10**	**1.71**

Table 4: The likelihood that DIKBA is faster than TWO-Q for given threshold ratios.

Threshold ratios		0.20	0.30	0.40	0.50	0.60	0.70
Louisiana (LA)	(35793)	98.20%	84.60%	59.60%	32.60%	13.20%	4.60%
Mississippi (MS)	(39986)	100.00%	95.40%	59.80%	15.60%	2.40%	0.60%
Nebraska (NE)	(44765)	88.00%	49.80%	33.00%	24.20%	17.80%	7.80%
Florida (FL)	(50109)	100.00%	89.00%	54.40%	26.40%	7.80%	1.60%
South Carolina (SC)	(52965)	99.80%	88.00%	58.00%	25.40%	5.20%	0.80%
Iowa (IA)	(63407)	100.00%	95.80%	41.60%	2.60%	0.00%	0.00%
Minnesota (MN)	(65491)	99.80%	71.20%	32.80%	14.40%	1.80%	0.00%
Alabama (AL)	(66082)	100.00%	76.00%	49.40%	26.00%	9.40%	1.40%
Missouri (MO)	(67899)	100.00%	93.60%	59.00%	15.80%	0.80%	0.00%
Georgia (GA)	(92792)	100.00%	97.40%	58.60%	14.60%	0.00%	0.00%
Overall Average		**98.58%**	**84.08%**	**50.62%**	**19.76%**	**5.84%**	**1.68%**