Shortest Path Algorithms: An Evaluation using Real Road Networks

F. BENJAMIN ZHAN

Department of Geography and Planning, Southwest Texas State University, San Marcos, Texas 78666

CHARLES E. NOON

Management Science Program, The University of Tennessee, Knoxville, Tennessee 37996

The classic problem of finding the shortest path over a network has been the target of many research efforts over the years. These research efforts have resulted in a number of different algorithms and a considerable amount of empirical findings with respect to performance. Unfortunately, prior research does not provide a clear direction for choosing an algorithm when one faces the problem of computing shortest paths on real road networks. Most of the computational testing on shortest path algorithms has been based on randomly generated networks, which may not have the characteristics of real road networks. In this paper, we provide an objective evaluation of 15 shortest path algorithms using a variety of real road networks. Based on the evaluation, a set of recommended algorithms for computing shortest paths on real road networks is identified. This evaluation should be particularly useful to researchers and practitioners in operations research, management science, transportation, and Geographic Information Systems.

The computation of shortest paths is an important task in many network and transportation related analyses. The development, computational testing, and efficient implementation of shortest path algorithms have remained important research topics within related disciplines such as operations research, management science, geography, transportation, and computer science (DIJKSTRA, 1959; DIAL et al., 1979; GLOVER, KLINGMAN, and PHILIPS, 1985; AHUJA et al., 1990; GOLDBERG and RADZIK, 1993). These research efforts have produced a number of shortest path algorithms as well as extensive empirical findings regarding the computational performance of the algorithms (cf., for instance, GLOVER et al., 1985; GALLO and PALLOTTINO, 1988; MONDOU, CRAINIC, and NGUYEN, 1991; CHERKASSKY, GOLDBERG, and RADZIK, 1993).

When faced with the task of computing shortest paths, one must decide which algorithm to choose. Depending on the application, algorithm runtime can be an important consideration in the decision making process. Although a number of computational evaluations have been reported in the literature (e.g., HUNG and DIVOKY, 1988; GALLO and PALLOTTINO, 1988; CHERKASSKY et al., 1993), there is no clear answer as to which algorithm, or set of algorithms, runs fastest on real road networks, the most common type of network faced by practitioners. The primary goal of this paper is to identify which algorithms run the fastest on real road networks. A secondary goal is to better understand the sensitivity of algorithm performance to input data.

Past computational evaluations were mainly based on randomly generated networks. The methods for random network generation varied considerably. The resulting random networks ranged from complete networks with uniformly distributed arc lengths to highly structured grid networks. In comparison to real road networks, random networks often differ with respect to the degree of connectivity as indicated by the arc-to-node ratios. The real networks studied in this paper have arc-to-node ratios ranging from 2.66 to 3.28. This is different from many randomly generated networks described in the literature where arc-to-node ratios are reported as high as 10 (cf. GALLO and PALLOTTINO, 1988).

Transportation Science
Vol. 32, No. 1, February 1998

0041-1655/98/3201-0065 $01.25
© 1998 Institute for Operations Research and the Management Sciences

Another aspect in which random networks can differ from real networks stems from the fact that random network arc lengths are usually randomly drawn in an independent fashion. This can result in network irregularities whereby a node may be "close" to two adjacent nodes that are "far" apart. Such irregularities can strongly favor certain types of algorithms and drastically slow others. The random network generators reviewed in the literature had one characteristic which we felt resulted in significant differences in real versus random networks, namely, they apply a process for establishing connectivity or arc length generation in a *homogeneous* fashion across a network. Real network topology often contains areas of dense urban network surrounded by highly sub-networked suburban areas which are then further surrounded by a rural road structure. Certain methods for random network generation may replicate one particular area well, for example, grid network generators for downtown areas, but real networks contain a mixed pattern of different types of road network topologies which are virtually impossible to simulate.

We have tested a set of 15 shortest path algorithms using real road networks. The networks used for testing include road networks from 10 states across the Midwest and Southeast of the United States, and the U.S. National Highway Planning Network (NHPN) which spans the continental United States. Our relative ranking of the algorithms differs somewhat from past studies such as those of GALLO and PALLOTTINO (1988) and CHERKASSKY et al. (1993). The results should be useful for researchers and practitioners in different disciplines, such as operations research, management science, transportation, and Geographic Information Systems, who rely on shortest path computations within certain applications. Our study focuses on the relative speeds of the various algorithms. The issues of implementation and storage requirements are important, however, the availability of rigorously tested public domain codes allows practitioners to easily obtain and implement such codes into their own. The computational results for this paper were obtained using the set of public domain C source codes for computing shortest paths provided by CHERKASSKY et al. (1993) with only slight modifications. Their implementations proved to be fast with respect to computation time and efficient with respect to storage requirements.

The remainder of this paper is organized as follows. Section 1 provides some background on the prior study of CHERKASSKY et al. (1993) and summarizes the algorithms tested in our study. Section 2 details the computational study and results. Section 3 concludes the paper with a set of recommendations regarding algorithm selection.

1. BACKGROUND

AMONG THE EVALUATIONS of shortest path algorithms reported in the literature (GLOVER et al., 1985; GALLO and PALLOTTINO, 1988; MONDOU et al., 1991; and, CHERKASSKY et al., 1993), a recent study by CHERKASSKY et al. (1993) is the most comprehensive and up-to-date. CHERKASSKY et al. reported an evaluation of 17 shortest path algorithms. In their experiment, CHERKASSKY et al. tested the 17 algorithms on a number of randomly generated networks with different characteristics. A main observation from their study was that no single algorithm consistently outperformed all others over the various classes of simulated networks. Among their conclusions, they suggested that the Dijkstra algorithm implemented with double buckets (DIKBD) is the best algorithm for networks with nonnegative arc lengths, and that the Goldberg–Radzik algorithm with distance updates during topological ordering (GOR1) is a good choice for networks with negative arc lengths.

We will use a test environment similar to that of CHERKASSKY et al. as a starting point for our research. Our evaluation differs from their evaluation in that we use real road networks rather than randomly generated networks. Of the 17 algorithms evaluated in the CHERKASSKY et al. paper, only 15 are included in our study. Inasmuch as we do not consider acyclic networks, the special-purpose algorithm for acyclic networks tested by CHERKASSKY et al. was excluded from our study. Also, after some preliminary testing, we found that an implementation using stack ordering of labeled node processing is significantly slower than the rest of the algorithms and, hence, it too was not considered.

Before continuing, let us formally introduce some notations and define the shortest path problem. A network is a graph $G = (N, A)$ consisting of an indexed set of nodes N with $n = |N|$ and a spanning set of directed arcs A with $m = |A|$. Each arc is represented as an ordered pair of nodes, in the form from node i to node j, denoted by (i, j). Each arc (i, j) has an associated numerical value, d_{ij}, which represents the distance or cost incurred by traversing the arc. In this paper, we assume that bidirectional travel between a pair of nodes i and j is represented by two distinct directed arcs (i, j) and (j, i). Given a directed network $G = (N, A)$ with known arc length d_{ij} for each arc $(i, j) \in A$, the *shortest path problem* is to find the shortest distance (least cost) path from a source node s to every other node in the node set N.

TABLE I

Summary of the Fifteen Algorithms Studied

Abbreviation	Implementation Description	Complexity*	Additional References
Bellman–Ford–Moore			
BF	Basic implementation	$O(nm)$	Bellman (1958)
BFP	With parent-checking	$O(nm)$	
Dijkstra			
DIKQ	Naive implementation	$O(n^2)$	Dijkstra (1959)
	Using buckets structure		
DIKB	Basic implementation	$O(m + nC)$	Dial (1969)
DIKBM	With overflow bag	$O(m + n(C/\alpha + \alpha))$	Cherkassky et al. (1993)
DIKBA	Approximate buckets	$O(m\beta + n(\beta + C/\beta))$	
DIKBD	Double buckets	$O(m + n(\beta + C/\beta))$	
	Using heap structure		
DIKF	Fibonacci heap	$O(m + n \log(n))$	Fredman and Tarjan (1987)
DIKH	k-array heap	$O(m \log(n))$	Corman et al. (1990)
DIKR	R-heap	$O(m + n \log(C))$	Ahuja et al. (1990)
Incremental Graph			
PAPE	Pape–Levit implementation	$O(n2^n)$	Pape (1974)
TWO_Q	Pallottino implementation	$O(n^2m)$	Pallottino (1984)
Threshold Algorithm			
THRESH		$O(nm)$	Glover et al. (1984, 1985)
Topological Ordering			
GOR	Basic implementation	$O(nm)$	Goldberg and Radzik (1993)
GOR1	With distance updates	$O(nm)$	

*n, the number of network nodes; m, the number of network arcs; C, the maximum arc length in a network; α and β, input parameters.

These *one-to-all* shortest paths can be represented as a directed out-tree rooted at the source node s. This directed tree is referred to as a *shortest path tree*.

All of the algorithms evaluated in our study are based on the *labeling method*, but they differ according to the rules used to select labeled nodes for scanning and in the data structures used to manage the set of labeled nodes. Readers are referred to GALLO and PALLOTTINO (1988) and AHUJA, MAGNANTI and ORLIN (1993) for more comprehensive discussions of these issues. The specific algorithms evaluated in our study are summarized in Table I. Details of the algorithms and their implementations can be found in CHERKASSKY et al. (1993), or in the additional references listed in Table I. The algorithms are divided into the following five categories: 1) Bellman–Ford–Moore, 2) Dijkstra, 3) Incremental Graph, 4) Threshold, and 5) Topological Ordering. We further categorize the Dijkstra's implementations as either *naive*, *bucket structures*, or *heap structures*. It should be noted that the worst-case computational complexities of the tested algorithms include *polynomial* (polynomial in m and n), *pseudo-polynomial* (polynomial in n, m, and C), and *exponential* (PAPE algorithm).

The Dijkstra algorithm has a node selection rule that is distinct from the other algorithms. The rule ensures that the shortest path tree is constructed by "permanently labeling" one node at a time. Once a node is permanently labeled, its optimal shortest path distance from the source node is known. Hence, if it is only necessary to find the shortest path from one node to some other node (the *one-to-one* shortest path problem), then Dijkstra's algorithm can be terminated as soon as the destination node is permanently labeled. All other algorithms guarantee optimal shortest path distance to any destination only upon termination with the full shortest path tree.

2. COMPUTATIONAL STUDY AND RESULTS

THE DESIGN OF THE experiment includes the preparation of the network data and the computational testing itself. Road networks from ten states in the Midwest and Southeast of the United States, and a road network consisting of the NHPN, covering the entire continental United States, were used for testing the shortest path algorithms. The ten states chosen for testing provide a good range of rural, suburban, and urban topology.

Two road network data sets were created and used in our study. The two sets differ in the size of networks included. Data set 1 consists of ten *low-detail* road networks, one for each of the ten states in our study. The set was generated using the three highest levels of roads, namely, interstate highway, principal arterial roads, and major arterial roads from U.S. Geological Survey's Digital Line Graphs. Figure 1 displays the Missouri road network from data set 1. Data set 2 consists of ten *high-detail* state road networks and a U.S. NHPN (abbreviated as US).

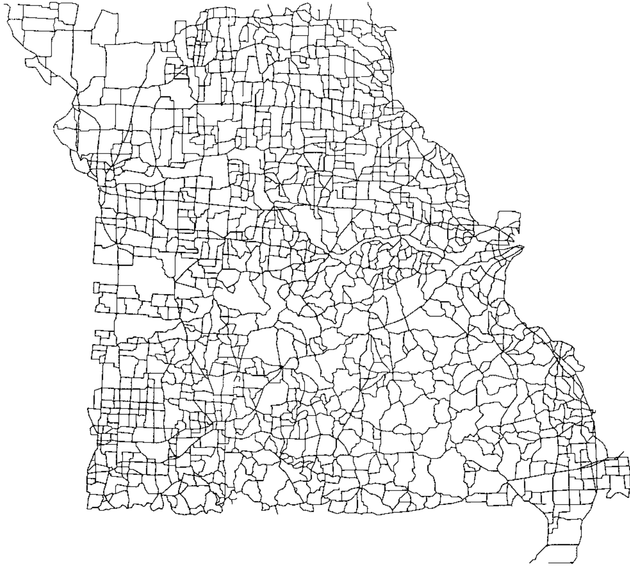

Fig. 1. Low-detail road network for Missouri from data set 1.

The ten high-detail state networks were generated by adding a fourth level of roads identified as rural minor arteries to the networks in data set 1. Figure 2 illustrates the Missouri road network from data set 2.

The road networks were stored and maintained as a set of nodes and bidirectional links in a geographic information system. The nodes, links and link-lengths were downloaded from the geographic information system into ASCII files. Before downloading the networks to files, a check was made to ensure that the road networks were fully connected. Two directed arcs were created for each bidirectional link in the data sets, hence, the number of arcs was always equal to twice the number of links. Characteristics of the 21 test networks are given in Table II. We found no notable difference in the arc-to-node ratio across the two data sets. The arc lengths of the networks are given in decimal geographic degrees. Since the input to the shortest path codes required integer distances, the arc lengths were multiplied by a *scaling factor*, and the resulting arc lengths were truncated to integers. This type of scaling and truncation affects the size of the arc lengths, which may have performance implications depending on the algorithm. A study of algorithm sensitivity to the scaling factor is described in a later part of this section.

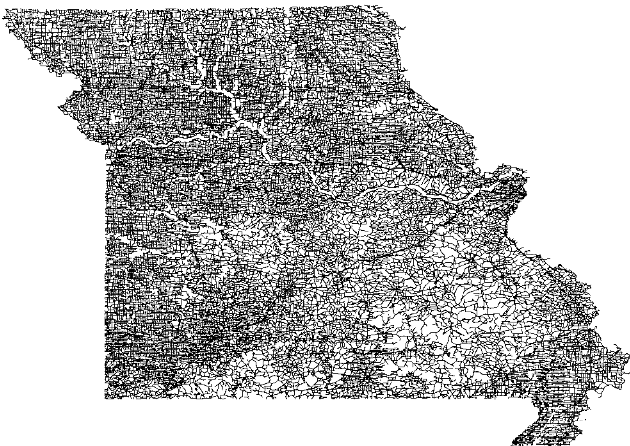

Fig. 2. High-detail road network for Missouri from data set 2.

The programs were compiled with the GNU gcc compiler version 2.5.6 using the O4 optimization option. Our experiments were conducted on a standalone SUN Sparc-20 workstation, model HS21 with a 125 MHz Hypersparc processor and 64 Megabytes of RAM running under the Solaris 2.4 environment.

The reported runtimes represent the CPU time for computing the shortest path trees and do not include data input or solution output. For each network, a sample of 100 nodes was randomly selected at the outset and designated as the *sample source nodes* for that network. For a given combination of road network, algorithm, and scaling factor, an individual time estimate for generating each of the 100 shortest path trees was computed. To ensure accuracy, the time estimate corresponding to a single source node was made by averaging the time to generate 1000 identical trees from the source node for the networks in data set 1. For data set 2, the average time to generate 10 (rather than 1000) identical trees represented a source node estimate. Once the 100 individual source node estimates were compiled, an average and a ratio of the maximum individual time to the average were computed.

In our first set of computational results, we summarize individual algorithm performance corresponding to a scaling factor of 1000. In a later part of this section, we analyze the effect of arc lengths on certain algorithms by altering the scaling factor. Tables III and IV display the relative speeds of the algorithms on data sets 1 and 2, respectively, with a scaling factor of 1000. For each table, the networks are given in order of increasing number of nodes and the algorithms are ordered by increasing overall relative speed ratio (the column displayed in bold). The last row in each table gives the average cpu time per shortest path tree for the best performing algorithm for a given network. The rows corresponding to the algorithms give the ratio of the average cpu time per tree for the algorithm to the time of the best performing algorithm for a network. For example, in Table III, PAPE was the best performing algorithm for Nebraska (NE) and had an average cpu time per tree of 0.46 milliseconds. The worst performing algorithm on the Nebraska network was DIKF which

TABLE II

Characteristics of the Networks

Network Name (abbreviation)	Number of Nodes	Number of Arcs	Arc/Node Ratio	Arc Length*		
				Maximum	Mean	Stnd. Dev.
Data set 1: 10 state networks with 3 levels of roads						
Nebraska (NE1)	523	1646	3.14	0.874764	0.215551	0.142461
Alabama (AL1)	842	2506	2.98	0.650305	0.128870	0.114031
Minnesota (MN1)	951	2932	3.08	0.972436	0.175173	0.132083
Iowa (IA1)	1003	2684	2.68	0.573768	0.119900	0.113719
Mississippi (MS1)	1156	3240	2.80	0.498810	0.095443	0.100703
South Carolina (SC1)	1784	5128	2.88	0.413163	0.062156	0.064389
Florida (FL1)	2155	6370	2.96	0.923088	0.075247	0.076590
Missouri (MO1)	2391	7308	3.06	0.494730	0.090977	0.064761
Louisianna (LA1)	2437	6876	2.82	1.021526	0.060662	0.067557
Georgia (GA1)	2878	8428	2.92	0.478579	0.068333	0.005668
Data set 2: 10 state networks with 4 levels of roads and the U.S. National Highway Planning Network						
Louisianna (LA2)	35793	98880	2.76	0.360678	0.013874	0.015297
Mississippi (MS2)	39986	120582	3.02	0.232062	0.015412	0.014000
Nebraska (NE2)	44765	146476	3.28	0.528283	0.018039	0.015652
Florida (FL2)	50109	133134	2.66	0.416212	0.011207	0.015264
South Carolina (SC2)	52965	149620	2.82	0.163557	0.009975	0.010198
Iowa (IA2)	63407	208134	3.28	0.269823	0.015733	0.009220
Minnesota (MN2)	65491	209340	3.20	0.410925	0.017202	0.014107
Alabama (AL2)	66082	185986	2.82	0.298232	0.011383	0.012410
Missouri (MO2)	67899	204144	3.00	0.212470	0.015542	0.013266
U.S. NHPN (US2)	75417	205998	2.74	1.500361	0.066084	0.094758
Georgia (GA2)	92792	264392	2.84	0.174245	0.010511	0.000107

*Arc lengths are in decimal degrees of a geographic coordinate system.

had an average runtime per tree that was 7.59 times greater than PAPE. Hence, the cpu time for DIKF can be figured as 7.59 times 0.46, or 3.49 milliseconds. The columns under the heading "Overall Performance" display the total of the cpu-time-per-source-node averages for an algorithm across all networks and that value's corresponding speed ratio relative to the fastest algorithm.

The incremental graph algorithms (PAPE and TWO_Q) dominate all other algorithms across both data sets. The nearest competing algorithm for both data sets is the Threshold (THRESH) algorithm

TABLE III

Relative Performance Summary for Data Set 1 with a Scaling Factor of 1000

Algorithm	Relative Performance by Network										Overall Performance		Average Max-to-Mean Ratio
	NE1	AL1	MN1	IA1	MS1	SC1	FL1	MO1	LA1	GA1	Total Time	Ratio	
PAPE	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	15.12	**1.00**	1.10
TWO_Q	1.08	1.08	1.08	1.08	1.09	1.08	1.07	1.07	1.08	1.06	16.22	**1.07**	1.10
THRESH	1.87	1.58	1.52	1.63	1.49	1.44	1.51	1.46	1.46	1.33	22.26	**1.47**	1.13
BFP	1.43	1.57	1.59	1.75	2.36	1.95	1.99	2.04	2.46	2.51	31.96	**2.11**	1.67
DIKBA	4.33	2.86	2.91	2.82	2.34	2.15	2.31	2.22	2.15	1.95	35.41	**2.34**	1.08
DIKB	4.33	2.87	2.92	2.85	2.35	2.16	2.33	2.23	2.16	1.97	35.61	**2.36**	1.09
GOR	2.47	2.47	2.49	2.46	2.59	2.50	2.42	2.52	2.50	2.47	37.61	**2.49**	1.10
DIKBM	4.63	3.20	3.14	3.29	2.66	2.42	2.53	2.61	2.33	2.31	39.98	**2.64**	1.28
DIKBD	3.75	3.48	3.29	3.36	2.95	2.90	2.96	3.06	2.88	2.67	45.25	**2.99**	1.04
BF	1.74	2.09	2.21	2.25	3.10	2.65	3.34	3.15	3.89	4.05	48.35	**3.20**	1.79
DIKQ	3.24	4.39	4.07	4.15	4.17	5.12	3.93	6.06	5.16	5.88	75.09	**4.97**	1.31
DIKH	4.61	5.37	4.71	5.12	4.87	5.26	4.68	5.37	5.28	5.31	77.47	**5.12**	1.48
DIKR	6.37	6.08	5.82	5.98	5.20	5.20	5.24	5.49	5.25	4.78	80.69	**5.34**	1.02
DIKF	7.59	8.25	7.82	8.19	7.57	8.42	7.73	8.94	8.57	8.23	124.63	**8.24**	1.07
GOR1	6.90	7.70	7.26	7.67	7.59	8.51	8.83	9.03	9.57	8.86	129.70	**8.58**	1.29
CPU time of minimum	0.46	0.73	0.90	0.86	1.09	1.63	2.08	2.24	2.25	2.87	15.12		

The values represent the ratio of the cpu time of an algorithm/network combination to the time of the best performing algorithm on the network. The given cpu times are in milliseconds.

Column in bold: The algorithm are ordered by increasing overall relative speed ratio.

TABLE IV

Relative Performance Summary for Data Set 2 with a Scaling Factor of 1000

Algorithm	Relative Performance by Network											Overall Performance		Average Max-to-Mean Ratio
	LA2	MS2	NE2	FL2	SC2	IA2	MN2	AL2	MO2	US2	GA2	Total Time	Ratio	
TWO_Q	1.05	1.02	1.00	1.00	1.01	1.02	1.00	1.00	1.01	1.00	1.00	2.95	**1.00**	1.16
PAPE	1.19	1.00	1.12	1.08	1.00	1.00	1.08	1.03	1.00	1.05	1.00	3.05	**1.03**	1.33
THRESH	1.00	1.33	1.39	1.06	1.42	1.72	1.47	1.28	1.59	1.36	1.43	4.11	**1.39**	1.13
DIKBA	1.17	1.60	1.56	1.14	1.58	1.95	1.69	1.44	1.75	1.30	1.60	4.53	**1.53**	1.11
DIKB	1.18	1.60	1.56	1.14	1.60	1.95	1.69	1.44	1.76	1.30	1.60	4.55	**1.54**	1.11
DIKBM	1.18	1.62	1.57	1.14	1.60	1.98	1.71	1.44	1.79	1.33	1.62	4.60	**1.56**	1.12
GOR	1.69	1.65	1.51	1.65	1.65	1.53	1.57	1.72	1.63	1.66	1.69	4.79	**1.62**	1.14
DIKBD	1.34	1.74	1.73	1.24	1.68	2.11	1.83	1.53	1.87	1.49	1.69	4.92	**1.67**	1.10
DIKR	1.78	2.18	2.26	1.58	2.07	2.78	2.40	1.93	2.39	2.02	2.13	6.35	**2.15**	1.12
DIKH	2.31	2.72	2.72	1.95	2.62	3.52	3.03	2.46	3.04	2.41	2.76	7.97	**2.70**	1.18
DIKF	4.04	4.66	4.23	3.32	4.21	5.23	4.80	4.07	4.79	3.80	4.44	12.73	**4.31**	1.14
GOR1	9.59	11.41	8.24	10.38	10.18	9.12	9.46	10.58	11.04	11.24	10.82	30.15	**10.21**	1.40
BFP	9.24	10.86	9.61	17.80	12.21	10.08	10.48	14.92	13.74	21.99	16.02	41.54	**14.06**	1.94
DIKQ	5.98	13.45	20.69	4.82	18.88	39.89	33.76	21.48	30.88	19.67	32.35	71.21	**24.11**	1.73
BF	19.35	24.54	22.75	37.69	26.23	23.83	23.95	31.57	32.59	44.66	34.28	90.44	**30.62**	1.99
CPU time of minimum	0.12	0.15	0.22	0.21	0.24	0.28	0.30	0.30	0.30	0.39	0.43	2.95		

The values represent the ratio of the cpu time of an algorithm/network combination to the time of the best performing algorithm on the network. The given cpu times are in seconds.

Column in bold: The algorithms are ordered by increasing overall relative speed ratio.

with an average time per tree that is roughly 40% larger than that of the incremental graph algorithms.

The Dijkstra bucket implementations (DIKBA, DIKB, DIKBD, DIKBM) perform fairly well on data set 1 with running time ratios ranging from 2.34 to 2.99, compared to the running time of PAP. For the larger networks of data set 2, the running times of the bucket implementations range a steady 1.53 to 1.67 that of TWO_Q.

The Dijkstra heap implementations (DIKF, DIKH, DIKR) are clustered together on both data sets. On data set 1, they perform poorly with relative running time ratios ranging from 5.12 to 8.24. On data set 2, their relative performance improves for DIKR and DIKH with ratios of 2.15 and 2.70, respectively. The relative ratios of DIKF lag behind the other heap implementations. They are 8.24 and 4.31 for data sets 1 and 2, respectively.

Compared to the best performing algorithms, the naive implementation of the Dijkstra algorithm (DIKQ) is roughly five times slower on the small networks of data set 1 and over 24 times slower on the large networks of data set 2. The topological ordering algorithms (GOR and GOR1) turn in lackluster performance on data set 1 with ratios of 2.49 and 8.58. On data set 2, GOR1 stays relatively slow with a ratio of over 10, whereas GOR closes the performance gap with a ratio of 1.62.

The Bellman–Ford–Moore implementations (BFP and BF) run 2 to 3 times longer than PAPE on data set 1 but then perform exceedingly poorly on data set 2 with relative time ratios of approximately 14 and 31.

The last column in both Tables III and IV provides a measure of algorithm predictability. For each combination of algorithm and network, individual times were calculated for generating shortest path trees for 100 source nodes and the ratio of the maximum-to-mean time was computed. The last column gives an average of the maximum-to-mean ratios across each set of networks. A high average ratio would imply that the algorithm took a significantly longer amount of time on some source nodes when compared to the average per-node time. The Bellman–Ford–Moore implementations had some of the highest average ratios for both data sets. The naive implementation of Dijkstra has a somewhat low ratio for data set 1, but then has a relatively high ratio for data set 2. Most of the other algorithms have relatively low ratios for both data sets, which suggest they maintain a consistent speed performance irrespective of source node.

The preceding results suggest that different forms of bucket implementations of the Dijkstra algorithm constitute a set of relatively fast shortest path algorithms. In such implementations, however, the number of buckets is directly related to the maximum arc length of a network. Thus, a natural question concerning the results obtained so far is how a different range of arc lengths would affect the performance of the algorithms whose worst-case complexity is a function of the maximum arc length. The approaches in question are the four Dijkstra bucket

TABLE V

Total cpu Time (in seconds) for Selected Algorithms under Various Arc Length Scaling Factors

	Scaling Factors											
	10 State Networks in Data Set 1				10 State Networks in Data Set 2 (the U.S. NHPN is excluded)				The U.S. NHPN Network			
Algorithm	100	1000	10,000	100,000	100	1000	10,000	100,000	100	1000	10,000	100,000
DIKBA	0.029	0.035	0.047	0.121	3.473	4.024	3.999	4.032	0.496	0.504	0.500	0.510
DIKB	0.030	0.036	0.115	1.128	3.477	4.040	3.932	4.759	0.502	0.505	0.545	1.443
DIKBD	0.037	0.045	0.057	0.075	3.593	4.341	4.583	5.002	0.524	0.580	0.619	0.819
DIKBM	0.034	0.040	0.104	0.525	3.506	4.078	3.975	4.626	0.503	0.518	0.539	1.219
DIKR	0.055	0.081	0.105	0.127	4.099	5.559	6.668	7.699	0.642	0.786	0.916	1.051

CPU times are the average total cpu times for constructing 100 shortest path trees on all given networks in a data set.

implementations (DIKBA, DIKB, DIKBD, and DIKBM) and the Dijkstra algorithm implemented with an R-heap structure (DIKR).

In order to see whether the findings obtained in the first set of runs (with a scaling factor of 1000) hold when the arc lengths change, additional scaling factors of 100, 10,000, and 100,000 were applied and the selected codes were rerun on the resulting networks. The rationale for analyzing performance over these four scaling factors can be easily understood by examining the maximum and mean arc length of all networks in Table II. A scaling factor smaller than 100 will generate networks with many arc lengths equal to zero. Arc lengths generated by a scaling factor of 100,000 will have a precision greater than that normally needed in practice. A scaling factor larger than 100,000 for the data sets would likely result in integer overflow difficulties within many of the shortest path codes.

The scaling factor of 100,000 resulted in maximum arc lengths with values as high as 102,153 in data set 1, 52,828 in data set 2 excluding the U.S. network, and 150,032 for the U.S. network. Such a scaling factor would provide a much greater degree of accuracy than would normally be needed in practice. However, we wanted to test the algorithms with these extreme arc lengths to examine their robustness.

The results are summarized in Table V, which shows the total cpu time (in seconds) used to construct 100 shortest path trees on all given networks for each implementation on each data set. Note that the U.S. network is reported separately from data set 2 because the U.S. arc lengths differ significantly from the arc lengths of the states. As expected, the cpu times increase along with an increase in scaling factor. However, the degree of increase differs across implementations and data sets. All five of the implementations seem to maintain consistent performance on the data set 2 networks, with DIKR showing the greatest fall-off in performance. This is due to the fact that the data set 2 arc lengths stay relatively lower with a maximum value of only 52,828.

For the data set 1 networks and, to some extent, the U.S. network, the performances of all algorithms drop significantly as the scaling factor is increased. The implementations that appear to be most sensitive to arc length are DIKB and DIKBM, each displaying a dramatic slowdown between the scaling factors of 10,000 and 100,000. The Dijkstra's implementations with approximate (DIKBA) and double (DIKBD) buckets exhibit relatively little degradation in performance under increasing arc lengths up to a scaling factor of 10,000. At a scaling factor of 100,000, DIKBD incurs additional slowdown, yet relatively less as compared to DIKBA.

The results in this section suggest that implementation performance can be greatly impacted by arc length. Hence, an understanding of the network data is needed before choosing one of the above implementations.

3. CONCLUSIONS AND RECOMMENDATIONS

SHORTEST PATH ALGORITHMS are central to many network and transportation analysis problems. When computing shortest paths for solving practical problems, one often faces a task of choosing the fastest shortest path algorithm for a problem at hand. Unfortunately, past evaluations of existing shortest path algorithms were mainly based on randomly generated networks which may not accurately reflect the characteristics of real road networks. Hence, the results regarding the performance of algorithms from these past evaluations may be misleading when they are used as guidelines to choose an algorithm for computing shortest paths on real road networks. The aim of this paper has been to fill this void and to obtain a set of recommended shortest path algorithms for computing shortest paths on real road networks.

We have tested a set of 15 shortest path algorithms on two data sets generated from real road networks chosen from 10 states across the Midwest and Southeast of the United States, and from the U.S. National Highway Planning Network covering the entire continental United States. Based on speed and stability corresponding to different arc lengths, we recommend a number of algorithms for networks with varying sizes.

The best performing implementations for solving the one-to-all shortest path problem are PAPE and TWO_Q. Both implementations are extremely fast for large or small networks and their per-shortest-path tree runtimes showed little variation as a function of source node. Furthermore, their performance is not a function of arc length magnitude. The fact that TWO_Q has a polynomial worst case complexity versus PAPE's exponential worst case gives a slight edge to TWO_Q.

If it is only necessary to compute a one-to-one shortest path or the shortest paths from a source node to a subset of the nodes (one-to-some), it may be worthwhile to consider one of the Dijkstra's implementations. As mentioned in Section 1, the Dijkstra implementations have the advantage that they can be terminated as soon as the destination node(s) become permanently labeled. This often yields significant computational savings if the destination nodes are in relatively close proximity to the source node. Our recommendations from the set of Dijkstra implementations depend on the maximum size of the network arc lengths.

If the network arc lengths stay within some reasonable maximum, say 1500, the Dijkstra approximate buckets implementation (DIKBA) is recommended. This implementation is the fastest among the Dijkstra codes for problems generated with a scaling factor of 1000 and, hence, maximum arc lengths of around 1500. For problems with a maximum arc length greater than 1500, the Dijkstra double buckets (DIKBD) implementation should also be considered because it appears to be less sensitive on problems in data set 1 with very large arc lengths.

A summary of our recommendations is given in Table VI. Certain implementations should be avoided altogether when solving shortest paths on real networks, namely, BF, BFP, and DIKQ. The Bellman–Ford–Moore implementations (BF and BFP) have serious difficulties on large networks. In the case of the U.S. network, their required cpu times were roughly 45 and 22 times larger than the best performing algorithms. In addition, they have the largest ratios of maximum-to-mean source node times. The naive implementation of Dijkstra's algorithm (DIKQ) also encounters difficulties as the network size increases and should thus be avoided.

TABLE VI

Summary of Recommendations

Situation	Recommended Implementation
One-to-all shortest paths	Graph Growth by Pallottino (TWO_Q)
One-to-one or one-to-some with:	
$C^* \leqslant 1500$	Dijkstra's Approximate Buckets (DIKBA)
$C^* \geqslant 1500$	DIKBA and Dijkstra's Double Buckets (DIKBD)
Implementations that Should be Avoided	
Bellman–Ford–Moore (BF)	
Bellman–Ford–Moore with Parent Checking (BFP)	
Dijkstra's Naive Implementation (DIKQ)	

*C is the maximum arc length.

ACKNOWLEDGMENTS

WE WOULD LIKE to thank Boris V. Cherkassky, Andrew V. Goldberg and Tomasz Radzik for making their one-to-all shortest paths C source codes available. Benjamin Zhanis was partly supported by a faculty research enhancement grant from Southwest Texas State University.

REFERENCES

AHUJA, R. K., T. L. MAGNANTI AND J. B. ORLIN, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

AHUJA, R. K., K. MEHLHORN, J. B. ORLIN AND R. E. TARJAN, "Faster Algorithms for the Shortest Path Problem," *JACM* **37,** 213–223 (1990).

BELLMAN, R. E., "On a Routing Problem," *Q. Applied Math.* **16,** 87–90 (1958).

CHERKASSKY, B. V., A. V. GOLDBERG AND T. RADZIK, Shortest Paths Algorithms: Theory and Experimental Evaluation, Technical report 93-1480, Computer Science Department, Stanford University, 1993.

CORMAN, T. H., C. E. LEISERSON AND R. L. RIVERST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

DIAL, R. B., "Algorithm 360: Shortest Path Forest with Topological Ordering," *Com. ACM* **12,** 632–633 (1969).

DIAL, R. B., F. GLOVER, D. KARNEY AND D. KLINGMAN, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," *Networks* **9,** 215–248 (1979).

DIJKSTRA, E. W., "A Note on Two Problems in Connection with Graphs," *Numeriche Mathematik* **1,** 269–271 (1959).

FREDMAN, M. L. AND R. E. TARJAN, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms," *JACM* **34,** 596–615 (1987).

GALLO, G. AND S. PALLOTTINO, "Shortest Paths Algorithms," *Ann. Oper. Res.* **13,** 3–79 (1988).

GLOVER, F., R. GLOVER AND D. KLINGMAN, "Computational Study of an Improved Shortest Path Algorithm," *Networks* **14,** 25–37 (1984).

GLOVER, F., D. KLINGMAN AND N. PHILIPS, "A New Polynomially Bounded Shortest Paths Algorithm," *Oper. Res.* **33,** 65–73 (1985).

GOLDBERG, A. V. AND T. RADZIK, "A Heuristic Improvement of the Bellman–Ford Algorithm," *Appl. Math. Lett.* **6,** 3–6 (1993).

HUNG, M. H. AND J. J. DIVOKY, "A Computational Study of Efficient Shortest Path Algorithms," *Comp. Oper. Res.* **15,** 567–576 (1988).

MONDOU, J.-F., T. G. CRAINIC AND S. NGUYEN, "Shortest Path Algorithms: A Computational Study with the C Programming Language," *Comp. Oper. Res.* **18,** 767–786 (1991).

PALLOTTINO, S., "Shortest-Path Methods: Complexity, Interrelations, and New Propositions," *Networks* **14,** 257–267 (1984).

PAPE, U., "Implementation and Efficiency of Moore Algorithms for the Shortest Root Problem," *Math. Program.* **7,** 212–222 (1974).

(Received: August 1996; revisions received: November 1996; accepted November 1996)