ALGORITHMS FOR ONE-TO-ONE TIME DEPENDENT SHORTEST PATH
ON REAL NETWORKS

Taehyeong Kim*
Advanced Transport Research Division
Korea Institute of Construction Technology
Goyang-Si, 411-712 KOREA
Telephone: +82-31-910-0596
Fax: +82-31-910-0339
E-mail: tommykim@kict.re.kr

Ali Haghani
Department of Civil and Environmental Engineering
University of Maryland
College Park, MD 20742 USA
Telephone: +1-301- 405-1963
Fax: +1-301- 405-2585
E-mail: haghani@umd.edu

Hyoungsoo Kim
Advanced Transport Research Division
Korea Institute of Construction Technology
Goyang-Si, 411-712 KOREA
Telephone: +82-31-910-0177
Fax: +82-31-910-0339
E-mail: hsookim@kict.re.kr

*corresponding author

3522 words + 4 Figures + 5 Tables = 5772 words

Nov. 15, 2012
ABSTRACT

In this research, we propose one-to-one time dependent shortest path (TDSP) algorithms, where the link flow speeds depend on time intervals. For this work, first three general shortest path algorithms which have proven to be fast and efficient algorithms in real networks are selected. These are Dijkstra’s algorithm with approximate buckets, Dijkstra’s algorithm with double buckets and Graph growth algorithm with two queues, and all three algorithms are to compute shortest paths from one node to all nodes in a network. These algorithms are modified to compute a shortest path from an origin node to a destination node in time dependent networks. Also, instead of travel time models, flow speed models and arrival time functions are used in the three algorithms to hold the First-In First-Out property.

Three modified algorithms are tested and evaluated on 3 data sets from real networks. Data set 1 consists of 10 low-detail state road networks and data set 2 consists of 10 high-detail state road networks in the United States. Data set 3 consists of 4 urban street networks for Anaheim, CA, Baltimore, MD, Chicago, IL, and Philadelphia, PA. Based on the computational results, among the three algorithms for TDSP, the best performing algorithms for solving one-to-one time dependent shortest path for urban street networks and for state wide networks, are modified Dijkstra’s algorithm with double buckets and modified Graph growth algorithm with two queues, respectively (232 words).

Keywords: Time dependent shortest path problem, one-to-one shortest path, Dijkstra’s algorithm, Graph growth algorithm
ALGORITHMS FOR ONE-TO-ONE TIME DEPENDENT SHORTEST PATH ON REAL NETWORK

1. INTRODUCTION

The classical shortest path problem (SPP) is an important part of the optimization problems and has been widely studied in operation research, management science, transportation, geography, and computer science for several decades. Many models and solution method for the classical shortest path problem have been developed by Dijkstra (1), Dial et al. (2), Pallottino (3), Glover et al. (4), Gallo and Pallottino (5), Hung and Divoky (6), Ahuja et al. (7), Cherkassky et al. (8), Goldberg and Radzik (9), Zhan and Noon (10, 11), and Zhan (12).

In classical shortest path problem it is assumed that travel speeds and travel times are constant. In reality, travel speeds and times in urban areas changes during the day because of congestion in certain parts of the road network. Therefore, time dependent shortest path problem (TDSPP) was extended from SPP to consider time dependent networks and the models and solution methods for TDSPP have been developed by Cooke and Halsey (13), Dreyfus (14), Orda and Rom (15), Kaumann and Smith (16), Ziliaskopoulos and Mahmassani (17), Chabini (18), Sung et al. (19), Ichoua et al. (20), Ding et al. (21), and Berger et al. (22). With the latest development of advanced information technologies such as automatic vehicle location (AVL), digital telecommunication, computers, and GIS, TDSPP can be applied to wide areas in the real life. Especially, developing more efficient algorithms for shortest paths on time dependent networks is an important task for scheduling and routing in transportation, public transit and logistics. In TDSPP, it is important that the model can satisfy the first-in first-out (FIFO) property. The FIFO property guarantees that two vehicles travelling on the same link will arrive at the end of the link in the same order as they start, even when some congestion occurs during the travel. Sung et al. (19) and Ichoua et al. (20) indicated the major drawback of using models that are based on discrete travel time and cost function and proposed travel speed model instead of travel time model.

This paper is mainly focused on developing one-to-one time dependent shortest path algorithms, where the link flow speeds depends on the time interval. For this work, three general shortest path algorithms which have proven to be fast and efficient algorithms in real networks are selected. These three algorithms are Dijkstra’s algorithm with approximate buckets, Dijkstra’s algorithm with double buckets, and Graph growth algorithm with two queues. All three algorithms are used to compute shortest paths from one node to all nodes in a network. These algorithms are extended to compute a shortest path from an origin node to a destination node in time dependent networks. The three extended algorithms are tested and evaluated on 3 data sets from real networks.

The remainder of this paper is organized as follows. In the next section, we discuss three general shortest path algorithms which are selected for TDSPP. After that we focus on how to extend these three
algorithms to compute time dependent shortest path in section 3. We will present results of a computational study in section 4. In the last section, we summarize the results and provide ideas for further research.

2. THREE SELECTED ALGORITHMS

Cherkassky et al. (8) provides one of the most comprehensive evaluations of shortest path algorithms. They evaluated the 17 one-to-all shortest path algorithms on a number of randomly generated networks with different characteristics. The result of their study was that no single algorithm performs consistently well on all simulated networks. And they suggested that Dijkstra’s algorithm implemented with double buckets (DIKBD) is the best algorithm for networks with nonnegative arc lengths.

Zhan and Noon (10) evaluated 15 algorithms out of 17 algorithms tested by the Cherkassky et al. (8) on real road networks rather than randomly generated networks. For real road networks, 2 data sets were created from 10 states in the United States and the two sets differ in the size of networks included. Data set 1 consists of 10 low-detail state road networks and data set 2 consists of 10 high-detail state road networks. Based on the evaluation, Graph growth algorithm with two queues (TWO_Q) was recommended for one-to-all shortest path, Dijkstra’s algorithm with approximate buckets (DIKBA) and Dijkstra’s algorithm with double buckets (DIKBD) were recommended for one-to-one or one-to-some shortest path.

In Cherkassky et al. (8) and Zhan and Noon (10), objective of the algorithms is to find the shortest path having minimum distance. 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 length and a study of algorithm sensitivity to the scaling factor was conducted.

A comparison of DIKBA and TWO_Q algorithms to compute one-to-one shortest paths was done by Zhan and Noon (11) based on the results of Zhan and Noon (10). They concluded that, in some situations, the TWO_Q is a better choice to compute one-to-one shortest paths on real road networks. Therefore, three general shortest path algorithms namely, TWO_Q, DIKBA, and DIKBD, that had high performances were selected and extended to develop algorithms for TDSPP in this research. A summary of the selected general shortest path algorithms is presented in Table 1.
TABLE 1
Summary of the selected general shortest path algorithms for TDSPP

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Implementation Description</th>
<th>Complexity*</th>
<th>Additional References</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIKBA</td>
<td>Dijkstra’s algorithm using approximate buckets</td>
<td>O(m β+n(β+C/β))</td>
<td>Cherkassky et al.(8) and Zhan(12)</td>
</tr>
<tr>
<td>DIKBD</td>
<td>Dijkstra’s algorithm using double buckets</td>
<td>O(m+n(β+C/β))</td>
<td>Cherkassky et al.(8) and Zhan(12)</td>
</tr>
<tr>
<td>TWO_Q</td>
<td>Graph growth using two queues</td>
<td>O(n^2 m)</td>
<td>Pallottino(5), Cherkassky et al.(8), and Zhan(12)</td>
</tr>
</tbody>
</table>

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

3. MODIFICATION OF THREE ALGORITHMS FOR TIME DEPENDENT NETWORKS

In this section, three general shortest path algorithms which are selected for TDSPP are modified to compute the shortest path from an origin node to a destination node in a time dependent network.

The terms and symbols used in this research are defined as follows:

\[ d(i) = \text{arrival( starting) time at node } i \text{ (time unit of } d(i) \text{ is second)}, \]

\[ (i, j) = \text{arc, where from-node is node } i \text{ and to-node is node } j, \]

\[ \text{ArrivalTime}(d(i), (i, j)) = \text{the arrival time at node } j\text{ starting from node } i \text{ at time } d(i), \]

\[ \text{IntervalLength} = \text{the length of time interval}, \]

\[ \text{source} = \text{the origin node}, \]

\[ \text{ending} = \text{the destination node}, \]

\[ f\_node = \text{from node of a link}, \]

\[ t\_node = \text{to nodes of a link}, \]

\[ N = \text{the set of nodes}, \]

\[ A(i) = \text{the set of arcs whose from-node is node } i \]

3.1 Link flow speed model

In this research, travel times are subject to change according to the time of the day. Travel times from one location to another are not necessarily the same in both directions. We assume that we have link flow speeds within each time interval which is based on historical data in network. Given link flow speeds we can calculate the expected travel time between origin and destination at starting time using a time dependent shortest path algorithm. For holding the FIFO property, flow speed model and arrival time function used by Sung et al. (19) are adopted for this problem as follows:

\[ \text{Function ArrivalTime}(d(i), (i, j)) \]

\[ \text{temp} = d(i) \times 1.0/60; \]

\[ \text{IntervalNumber} = \text{temp} / \text{IntervalLength} ; \]

\[ \text{Res.length} = \text{length of link}(i, j); \]

\[ \text{temp.speed} = \text{Link.Speed.Function}(\text{IntervalNumber}, (i, j)); \]

\[ \text{Res.length} = \text{Res.length} - \frac{\text{temp.speed}}{60} \times ((\text{IntervalNumber} + 1) \times \text{IntervalLength} - \text{temp}); \]
while (Res_length > 0) do
begin
    IntervalNumber := IntervalNumber + 1;
    temp_speed := Link.Speed_Function(IntervalNumber, (i,j));
    Res_length := Res_length − temp_speed * 60 * IntervalLength;
end

arrival_time := (IntervalNumber + 1) * IntervalLength * 60 + \frac{Res_length}{temp_speed} * 3600;

Return arrival_time;

It is assumed that there are three types of roads in a network. The first one is highways on which the speed limit is 60 mph, the second one is major roads on which the speed limit is 40 mph, and the last one is minor roads on which the speed limit is 30 mph. Each link belongs to one of them. Also it is assumed that link flow speeds of highways and major roads are varied according to time intervals as shown in Figure 1. The length of a time interval is 10 minutes and there are 144 time intervals in a day.

FIGURE 1
The Variations of link flow speeds.

3.2 Modified Graph growth algorithm with two queues

The selected Graph growth algorithm with two queues (TWO_Q) is modified for TDSPP. ArrivalTime function introduced in the previous section is added in this algorithm. Also threshold value is added to avoid unnecessary expansion in scanning nodes in a network and to reduce the calculation time of shortest path. At beginning, threshold is set to infinity. While scanning nodes, if the destination node is scanned, threshold value is updated as the scanned value of destination node. If any node that has larger value than
threshold is scanned, it is not included in list (queue). The main procedure of modified TWO_Q (M-
TWO_Q) is described as follows.

Algorithm E-TWO_Q:

begin
  initializing source node and other nodes;
  threshold := \infty
  INIT_QUEUE(source)
  /* main loop */
  while (NONEMPTY_QUEUE) do
    begin
      EXTRACT_FIRST(f_node);
      for each successor node(t_node) of f_node do
        arr
time: = ArrvialTime(arrival time at f_node, arc_if);
        if (arrival time at t_node) then
          begin
            arrival time at t_node := arrival time at t_node;
            parent of t_node := f_node;
            if (arrival time at t_node < Threshold)
              begin
                if (t_node = ending) then
                  threshold := arrival time at t_node;
                if (! NODE_IN_QUEUE(t_node))
                  begin
                    if (NODE_WAS_IN_QUEUE(t_node))
                      begin
                        INSERT_TO_ENTRY(t_node);
                      else
                        INSERT_TO_ENTRY(t_node);
                      end
                    end
                end
              end
      end
    end
  end
end

INIT_QUEUE(source): Create an empty queue
EXTRACT_FIRST(node): Find and return a minimum value of node
NONEMPTY_QUEUE: Check whether queue is empty
NODE_IN_QUEUE(node): Check whether node is already in queue
NODE_WAS_IN_QUEUE(node): Check whether node has been in queue
NODE_TO_ENTRY(node): Insert node into the enter of queue
NODE_TO_BACK(node): Insert node into the back of queue
3.3 Modified Dijkstra’s algorithm with approximate buckets

The selected Dijkstra’s algorithm with approximate buckets (DIKBA) is modified for TDSPP and the main procedure of modified DIKBA (M-DIKBA) is described as follows:

Algorithm E-DIKBA;

begin
  /* initialization */
  d(j):= ∞ for each node j ∈ N;
  d(source):= 0;
  pred(source):= 0;
  INIT_BHEAP(source);
  /* main loop */
  while(1) do
    begin
      EXTRACT_MIN(i);
      if (i = NULL) break;
      for each (i, j) ∈ A(i) do
        begin
          value = ArrivalTime(d(i), (i, j));
          if (value < d(j)) then
            TIME_TO_POS(value, pos_new);
            if (NODE_IN_BHEAP(j)) then
              if (ins = (pos_old ≠ pos_new)) then
                REMOVE_FROM_BHEAP(j, pos_old);
              else ins:=1;
              if (ins) then INSERT_TO_BHEAP(j, pos_new);
            d(j):= value
            pred(j):= i;
            if (j = ending) then break;
        end
    end
end

INIT_BHEAP(source): Create an empty double heap
EXTRACT_MIN(node): Find and return a minimum value of node
REMOVE_FROM_BHEAP(node, pos): Delete a value of node on posth label in heap
INSERT_TO_BHEAP(node, pos): Insert a new value of node on posth label in heap
TIME_TO_POS(travel_time, pos): Find the location in heap for new travel time

3.4 Extended Dijkstra’s algorithm with double buckets

The selected general Dijkstra’s algorithm with double buckets (DIKBD) is modified for TDSPP and the main procedure of modified DIKBD (M-DIKBD) is exactly the same as the procedure of modified DIKBA (M-DIKBA). But, M-DIKBA uses approximate buckets and M-DIKBD uses double buckets as data structure to manipulate the set of temporarily labeled node in algorithm.
4. COMPUTATIONAL RESULTS

4.1 Test Network

Three modified algorithms are tested and evaluated on 3 data sets of real networks. Data set 1 and 2 are the same networks as that were used by Zhan and Noon (10). Data set 3 consists of 4 urban street networks for Anaheim, CA, Baltimore, MD, Chicago, IL, and Philadelphia, PA. The networks for Anaheim, CA, Chicago, IL, and Philadelphia, PA, were obtained from Bargera (23). Baltimore network was obtained from North East America data of commercial program, ArcLogistics Route. Figure 2 illustrates road network of Baltimore City and Baltimore County, MD from data set 3.

FIGURE 2
Road network for Baltimore City and Baltimore County.

The Characteristics of the test networks are given in Table 2. In this table, units of length are miles. As Zhan (12) mentioned, the degree of connectivity measured by the arc-to-node ratios is one important characteristic representing a real road network. The difference between the arc-to-node ratios in real road networks and simulated networks should be observed because the number of scans in constructing a
shortest path tree has close relation to arc-to-node ratios. In data set 1, the arc-to-node ratio ranges from 2.68 to 3.15, in data set 2, from 2.66 to 3.28, and in data set 3, from 2.20 to 3.01. Also, there is no notable difference in the arc-to-node ratio across the 3 data sets.

<table>
<thead>
<tr>
<th>CHARACTERISTICS OF THE TEST NETWORKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Name</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>State networks- Low detail (data set 1)</td>
</tr>
<tr>
<td>Alabama(AL1)</td>
</tr>
<tr>
<td>Florida(FL1)</td>
</tr>
<tr>
<td>Georgia(GA1)</td>
</tr>
<tr>
<td>Iowa(IA1)</td>
</tr>
<tr>
<td>Louisiana(LA1)</td>
</tr>
<tr>
<td>Minnesota(MN1)</td>
</tr>
<tr>
<td>Mississippi(MS1)</td>
</tr>
<tr>
<td>Missouri(MO1)</td>
</tr>
<tr>
<td>Nebraska(NE1)</td>
</tr>
<tr>
<td>South Carolina(SC1)</td>
</tr>
</tbody>
</table>

State networks- High detail (data set 2) |
<table>
<thead>
<tr>
<th>Network Name</th>
<th>Number of Nodes</th>
<th>Number of Arcs</th>
<th>Arc/Node Ratio</th>
<th>Arc Length Maximum</th>
<th>Arc Length Minimum</th>
<th>Arc Length Mean</th>
<th>Arc Length S. D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama(AL2)</td>
<td>66082</td>
<td>185986</td>
<td>2.81</td>
<td>0.298323</td>
<td>0.000004</td>
<td>0.11388</td>
<td>0.01</td>
</tr>
<tr>
<td>Florida(FL2)</td>
<td>50109</td>
<td>133132</td>
<td>2.66</td>
<td>0.416212</td>
<td>0.001012</td>
<td>0.11211</td>
<td>0.01</td>
</tr>
<tr>
<td>Georgia(GA2)</td>
<td>92792</td>
<td>264392</td>
<td>3.28</td>
<td>0.174245</td>
<td>0.001012</td>
<td>0.01573</td>
<td>0.01</td>
</tr>
<tr>
<td>Iowa(IA2)</td>
<td>63407</td>
<td>208134</td>
<td>2.76</td>
<td>0.360678</td>
<td>0.001112</td>
<td>0.02876</td>
<td>0.02</td>
</tr>
<tr>
<td>Louisiana(LA2)</td>
<td>35793</td>
<td>98880</td>
<td>3.20</td>
<td>0.410925</td>
<td>0.001373</td>
<td>0.01387</td>
<td>0.02</td>
</tr>
<tr>
<td>Minnesota(MN2)</td>
<td>65491</td>
<td>209340</td>
<td>3.02</td>
<td>0.232062</td>
<td>0.000141</td>
<td>0.01541</td>
<td>0.01</td>
</tr>
<tr>
<td>Mississippi(MS2)</td>
<td>39896</td>
<td>120582</td>
<td>2.80</td>
<td>0.360678</td>
<td>0.001112</td>
<td>0.02876</td>
<td>0.02</td>
</tr>
<tr>
<td>Missouri(MO2)</td>
<td>67899</td>
<td>204144</td>
<td>3.01</td>
<td>0.21247</td>
<td>0.000222</td>
<td>0.01554</td>
<td>0.01</td>
</tr>
<tr>
<td>Nebraska(NE2)</td>
<td>44765</td>
<td>149620</td>
<td>3.27</td>
<td>0.528283</td>
<td>0.00153</td>
<td>0.01804</td>
<td>0.02</td>
</tr>
<tr>
<td>South Carolina(SC2)</td>
<td>52965</td>
<td>149620</td>
<td>2.82</td>
<td>0.163557</td>
<td>0.000076</td>
<td>0.00998</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Urban area networks (data set 3) |
<table>
<thead>
<tr>
<th>Network Name</th>
<th>Number of Nodes</th>
<th>Number of Arcs</th>
<th>Arc/Node Ratio</th>
<th>Arc Length Maximum</th>
<th>Arc Length Minimum</th>
<th>Arc Length Mean</th>
<th>Arc Length S. D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>416</td>
<td>914</td>
<td>2.20</td>
<td>1.786239</td>
<td>0.049896</td>
<td>0.50867</td>
<td>0.33</td>
</tr>
<tr>
<td>Baltimore</td>
<td>63356</td>
<td>142483</td>
<td>2.25</td>
<td>0.994554</td>
<td>0.029395</td>
<td>0.06198</td>
<td>0.05</td>
</tr>
<tr>
<td>Chicago Regional</td>
<td>12982</td>
<td>39018</td>
<td>3.01</td>
<td>9.99</td>
<td>0.02</td>
<td>0.69328</td>
<td>0.64</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>13389</td>
<td>40003</td>
<td>2.99</td>
<td>7.25</td>
<td>0.01</td>
<td>0.45776</td>
<td>0.45</td>
</tr>
</tbody>
</table>

The three algorithms were coded in the C++ program language. They are modified from C source codes of one-to-all shortest paths provided by Cherkassky et al. (8). The input networks are represented using the forward star data structure. All computations were carried out on a machine with 2.0GHZ Intel Core 2 Duo CPU and 3GB memory in Windows XP environment.

For the comparison of the performance of three algorithms, a sample of 100 pairs of origin and destination nodes was randomly selected. Also, two starting times for each pair of origin and destination nodes were randomly chosen. One is within off-peak time and the other is within peak time.

4.2 Results

Table 3, 4, 5 and Figure 3 summarize the calculation times of three time dependent shortest path algorithms for low detail state networks, high detail state networks and urban street networks, respectively.

In case of low detail state networks, M-TWO_Q is faster than other two algorithms, M-DIKBA and M-DIKBD. M-TWO_Q is 83% faster than M-DIKBA and 45% faster than M-DIKBD. But, average max-to-mean ratio of M-TWO_Q is about 2 times larger than M-DIKBA and M-DIKBD. It shows that calculation
times of M-TWO_Q are more fluctuating according to the lengths of travel times than those of M-DIKBA and M-DIKBD.

Also, the results for the case of high detail state networks are similar to those for the case of low detail networks. M-TWO_Q is faster than the other two algorithms, M-DIKBA and M-DIKBD. M-TWO_Q is 3.7 times faster than M-DIKBA and 29% faster than M-DIKBD. But, average max-to-mean ratio of M-TWO_Q is about 2 times larger than M-DIKBA and M-DIKBD.

In case of urban street networks, M-DIKBD has better performances than M-TWO_Q and M-DIKBA. M-DIKBD is 14~15% faster than M-TWO_Q and M-DIKBA. There is not much difference of calculation times between M-TWO_Q and M-DIKBA. Also, there is not much fluctuation of computing times for M-DIKBA and M-DIKBD.

**TABLE 3**
Calculation times for state wide low detail networks

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Calculation times by networks(ms)</th>
<th>Overall average</th>
<th>Average max-to-mean ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AL</td>
<td>FL</td>
<td>GA</td>
</tr>
<tr>
<td>M-TWO_Q</td>
<td>3.09</td>
<td>8.1</td>
<td>11.07</td>
</tr>
<tr>
<td>M-DIKBA</td>
<td>5.55</td>
<td>14.84</td>
<td>20.29</td>
</tr>
<tr>
<td>M-DIKBD</td>
<td>4.8</td>
<td>11.32</td>
<td>14.84</td>
</tr>
</tbody>
</table>

**TABLE 4**
Calculation times for state wide high detail networks

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Calculation times by networks(ms)</th>
<th>Overall average</th>
<th>Average max-to-mean ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AL</td>
<td>FL</td>
<td>GA</td>
</tr>
<tr>
<td>M-TWO_Q</td>
<td>277.71</td>
<td>256.91</td>
<td>370.48</td>
</tr>
<tr>
<td>M-DIKBA</td>
<td>1230.22</td>
<td>832.86</td>
<td>1810.43</td>
</tr>
<tr>
<td>M-DIKBD</td>
<td>330.69</td>
<td>236.64</td>
<td>471.27</td>
</tr>
</tbody>
</table>

**TABLE 5**
Calculation times for urban street networks

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Calculation times by networks(ms)</th>
<th>Overall average</th>
<th>Average max-to-mean ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Anaheim</td>
<td>Baltimore</td>
<td>Chicago</td>
</tr>
<tr>
<td>M-TWO_Q</td>
<td>1.35</td>
<td>289.30</td>
<td>71.57</td>
</tr>
<tr>
<td>M-DIKBA</td>
<td>1.84</td>
<td>305.51</td>
<td>78.05</td>
</tr>
<tr>
<td>M-DIKBD</td>
<td>1.80</td>
<td>252.27</td>
<td>74.35</td>
</tr>
</tbody>
</table>
(a) Calculation times for state wide low detail networks

(b) Calculation times for state wide high detail networks
Comparison of calculation times.

Figure 4 shows average computing times of three algorithms. For state wide networks, M-TWO_Q and M-DIKBD are faster than M-DIKBA and M-TWO_Q is a little faster than M-DIKBD. Also, for urban street networks, data set 3, performance of M-DIKBD is better than that of M-TWO_Q and M-DIKBA. Based on computation results, among the three algorithms for TDSP, the best performing algorithms for solving one-to-one time dependent shortest path for urban street networks and for state wide networks are M-DIKBD and M-TWO_Q, respectively.
5. CONCLUSIONS

One-to-one time dependent shortest path algorithms, where the link flow speeds depend on the time intervals were proposed in this paper. For this work, three general shortest path algorithms which have proven to be fast and efficient algorithms in real networks were selected. These three algorithms are Dijkstra’s algorithm with approximate buckets, Dijkstra’s algorithm with double buckets, and Graph growth algorithm with two queues. These algorithms were modified to compute one-to-one shortest path in time dependent networks. Three modified algorithms, M-TWO_Q, M-DIKBA and M-DIKBD, were tested and evaluated on 3 data sets from real networks. In case of state wide networks, M-TWO_Q is faster than the other two algorithms, M-DIKBA and M-DIKBD. But, the calculation times of M-TWO_Q are more fluctuating according to the lengths of travel times than those of M-DIKBA and M-DIKBD. For urban street networks, performance of M-DIKBD is better than that of M-TWO_Q and M-DIKBA. Therefore, among three algorithms for TDSP, M-DIKBD is recommended to solve one-to-one time dependent shortest path for urban street networks and M-TWO_Q for state wide networks, respectively.

These developed algorithms for shortest paths on time dependent networks can be applied to the scheduling and routing in transportation, public transit and logistics. Actually M-DIKBD was applied to a large-scale multi-depot dial-a-ride problem proposed by Kim and Haghani (24, 25) and M-DIKBD could contribute to saving total calculation time to solve this dial-a-ride problem. For future research, there is potential for developing time dependent shortest path algorithms using heuristic methods such as decomposition, limiting search area, genetic algorithm, ant colony, and possible others, and comparing the results of heuristic methods with those of optimal methods.

ACKNOWLEDGEMENTS

This research was supported by a grant from a Strategic Research Project (Developing a microscopic simulation testbed on ITS environments) funded by Korea Institute of Construction Technology.

REFERENCES