Fast Algorithm to Solve the Most Economical Path Problem in Sparse Matrices

K. Ambrus-Somogyi*, A. Bakó**, P. Földesi*** and J. Gáti**
*Óbuda University/Rejtő Sándor Faculty, Institute of Media Technology and Light Industry, Budapest, Hungary
**Óbuda University, Budapest, Hungary
***Széchenyi University, Faculty of Engineering Science, Győr, Hungary

a_somogyi.kornelia@rkk.uni-obuda.hu, bako@uni-obuda.hu, foldesi@sze.hu, gati@uni-obuda.hu

Abstract—Several variations exist of the shortest path problem depending on the type of the graph. The most common problems are the shortest path problem between two points and between every pair of points (the so called multiterminal minimal path problem). Beside this several other variations are known. One of them is the shortest path in a network having gains, the time dependent path, the shortest path in a network, where the travelling time depends on the actual traffic flow.

In this paper a powerful algorithm is presented in large scale, sparse network. In long term road network planning problem sometimes the size of the network is very large – about hundred thousand. The network is usually sparse, one point is connected with average 3 or 4 points.

It will be shown that the presented algorithm in this case is much more powerful.

Keywords—SP problem, sparse matrices, long range road network planning

I. INTRODUCTION

The shortest path problem (SP) is a difficult task when the size of the network is large, the network is sparse and several shortest paths have to be determined. In this case the number of points of the network could be 100,000, and the number of the path determined should be about the same.

Several types of the problem are known.

The most simple problem is the shortest path between two points of the given graph, determination the minimal spanning tree from one point of a graph or determine the shortest path between each pair of points of a graph – the so called multiterminal minimal path problem.

Some other variations are known. One of them is the shortest path in a network having gains (see Bakó [2, 3]).

The other problem variation is the time dependent SP problem (Ayed et al. [1]). The other is where the travelling time depends on the actual traffic which is quite realistic. This algorithm is used in the Traffic Assignment Problem. Balasubramanian [4] presents two algorithms for computing distances along convex and concave polyhedral surfaces. A fuzzy hierarchical decomposition method was developed by Katz et al. [11].

One of the most complete summarization of the SP problem was given in the Stanford University (Oldham-Pratt [12]).

Several authors present algorithm for solving the SP problem between two points of a network. One of the earlier algorithm was given by Ford [10]. Some years later Dijkstra [9] and D’Esopo [8] presented better solutions for this problem. Some generalization of his algorithm was published by Deng et al. [6] and Delling et al.[7]. These algorithms are proper and fast in general networks, but theirs methods were not powerful in the case of large scale network. The transportation network has a special structure: only 3 or 4 points are connected to one point of the network. By using this fact a powerful algorithm is presented. The running time of this algorithm is less than the previously mentioned algorithms.

The presented algorithm has much faster execution time because it takes into consideration the special structure of the road network.

In the next section we present the special storing method of the network which is needed by the fast SP algorithm.

The next section contains the steps of the suggested SP algorithm. The last section demonstrates the running time of the faster SP algorithm and compares the previous mentioned algorithms.

II. THE GRAPH TRANSFORMATION

The presented algorithm was developed for a long range traffic planning and forecasting package. The task was to determine the future public traffic and the public network for Budapest and some other towns. First the public network was determined. After that this network was transformed into a single network. The road network in a city consists of nodes and directed arcs. The travelling time of an arc in a road network depends on the length of the arc. In a town the travelling time beside this depends on the travelling time going through the endpoint of the arc. This time depends on the direction where the traffic goes. It is demonstrated in Figure 1.

Figure 1. Sample crossing point
The network consists of 5 point and 8 arcs (part A of the Figure 1). The travelling time is going from \((e,a)\) to \((a,c)\) different from \((e,a)\) to \((a,b)\) and \((e,a)\) to \((a,d)\), because in a city the travelling time mostly depends on the way we go through the point (traffic lamps, pedestrians,…). A transformation of the network is needed in order to reach the real traffic situation mentioned above.

The real traffic situation is presented in Figure 1/B. This situation can be handled with a little modification of the original network.

Figure 2. Modified network

Let us to build up the dual graph to the original one. Add a new artificial point to each arc. Let us denote the new artificial point by \(b_1\), \(c_1\), \(d_1\) and \(e_1\) (see Figure 2). Connect these points to each other, thus we get some new artificial arcs going at \(e_1\) \((e_1,c_1)\), \((e_1,b_1)\), \((e_1,d_1)\). Similarly 3 times 3 new arcs could be given going at \(c_1\), \(d_1\) and \(b_1\). That is 4 new points and 12 new edges present the real traffic situation instead of 1 point and 4 edges. The final result of this transformation is demonstrated in Figure 3.

Figure 3. Transformed network

The travelling time of the new artificial arcs are computed. It depends on the length of the original arc and the travelling time going through the point – in our case \(a\). So the input data of this type of network are the lengths of the original arcs and the travelling time going through the point \((a\) to \((a,d)\). The travelling time of the new artificial arcs are \(t_{e_1}\) and \(t_{c_1}\) and \(t_{d_1}\). The point \(e_1\) is connected to point \(b_1\), \(c_1\), \(d_1\). The length of these arcs are the time go through point \(a\).

For a traffic planning project we transform the main traffic network of Budapest and we got a large network with 100.000 points. This network is sparse because one point is connected to average 3 other points.

The shortest path on this network has to be computed several times, that’s why it is important to develop a powerful, fast SP algorithm and a convenient storing method for sparse matrices. In the next section this storing method and SP algorithm is presented.

### III. Storing Method

As it was mentioned earlier the road network usually large and in the same time sparse. If the number of point is denoted by \(n\), then the \(n^2\) storing places is needed by using the traditional technique. The civil engineering uses several measures for characterize a network. The most simple measure is the completeness index

\[
b = \frac{m}{n}
\]

where \(m\) is the number of arcs and \(n\) is the number of points. Denote the set of points by \(N\) and the set of arcs by \(E\). There is a similar index for characterize the sparseness:

\[
b = \frac{m}{n(n-1)/2}
\]

The other index is the diameter of a graph which is the maximal value of SP in a graph:

\[
d = \max_{i,j \in N} t(P_{ij})
\]

where \(t(P_{ij})\) is the length of the minimal path from the point \(i\) to point \(j\). Several other measures could be used. Our suggestion is to compare the number of the arcs of the actual graph to the complete graph which contains all arcs.

If we want to store the input data of a traffic network, generally the matrix form is used. The network is given by matrix \(A\), where the elements are the following

\[
a_{ij} = \begin{cases} t_{ij} & \text{if arc}(i, j) \text{ exist} \\ \infty & \text{otherwise} \end{cases}
\]

where \(t_{ij}\) is the biggest number of the computer

Using the matrix \(A\), \(n^2\) storing place is needed. In the case of Budapest public network the number of the points in the transformed network is \(100.000^2=10,000,000,000\), which is 10 billion (2 or 4 bytes are needed for storing one element depending on the length of the arcs). This is a large number, and it is difficult to use. To avoid this problem some other storing methods were developed. The number of arcs approximately is 300,000. The number of all possible arcs is approximately \(100,000*100,000=10,000,000,000\). If we divide this number by the actual number of arcs, we get the density (sparseness) of the network which is low.

In the developed storing method only the nonzero elements of the matrix \(A\) are stored, that is only the length of the existing arcs.

The first method is quite simple. Two vectors are needed, \(A\) and \(B\) for storing the network. This means to
store the structure of the graph and the length of the arcs. In vector A, two elements belong to each point. The 2i−1-th element of vector A shows the number of arcs going out of point i, the next element of it is a pointer which shows the places where the specifications of these arcs are stored in vector B. Let a(2i) = q, then b(q) contains the end of the first arc going out of i, and b(q + 1) contains the length of this arc, the corresponding \( r(i, b(q)) \) value. Let us suppose, that p arcs go out of point i. Then endpoints of the arcs going out of point i are stored of the lost element is \( b(q + 2i) \), \( l = 0,1, \ldots, p - 1 \) and the corresponding length of the arcs are stored in \( b(q + 2i + 1) \), \( l = 0,1, \ldots, p - 1 \).

This storing method needs \( 2n + 2m \) storing places when \( n \) is the number of points and \( m \) is the number of arcs. In order to demonstrate this method a sample network is given in Figure 4.

![Sample Network](image)

Figure 4. The sample network

The correspondent vectors A and B are given above:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>31</td>
<td>11</td>
<td>22</td>
<td>19</td>
<td>21</td>
<td>22</td>
<td>25</td>
<td>1</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

An other pointer type storing method is presented now. The data of the network consist of 3 vectors. The size of vector A is equal to the number of points, the length of vector B and C are equal to the number of arcs.

The i-th element of vector A points to endpoint of the arcs \( (i, b(i)) \) in B and the length of the arc \( (i, b(i)) \) which \( t_{b(i)} \) is stored in c(i). If q arcs go out of point i, then the endpoints and the travelling times are stored the corresponding vector element of \( B \) and C. The endpoint of these arcs are stored in B in the following places \( b(i), b(i + 1), \ldots, b(i + q - 1) \) and the length of these arcs are stored in vector C in the following elements \( c(i), c(i + 1), \ldots, c(i + q - 1) \). The size of these vectors are the following: \( |A| = n \), \( |B| = |C| = m \), that is \( n + 2m \) elements are needed for storing the graph instead of \( n^2 \).

This storing method is demonstrated in connection the given graph (Figure 4).

![Graph](image)

IV. SHORTEST PATH IN TRANSPORTATION NETWORK

Our method is especially useful to find the shortest path in Transportation Network which is usually large and sparse. This method needs only n addition and n comparison instead of \( \Theta(n^2) \) which is the number of steps of other SP methods. Beside the addition and comparison we have to order increasing by the length of the arcs going out of each point. Because the network is large, and it could be supposed that average 3 arcs go out of each point this task is simple and need only two or three comparison in the case of each point. This little modification is presented below. The vector A is the same. Only the vector B and C would be modified.

![Graph](image)

It is easy to see, that in each step only one addition and comparison is needed. Each step the potentially candidate arc are marked, the initial points of that are stored in vector \( H_1 \) and the correspondent endpoint in vector \( H_2 \) respectively.

The algorithm is a tree building potential algorithm. In this method the set of points N is divided into two sets S and T which fulfill

\[ S \cap T = \emptyset \quad S \cup T = N \]  

(5)

The potentially candidate set the following

\[ E_{ST} = \{ (x,y) \mid x \in S, y \in T, (x,y) \in E, t_y = \min_{z \in T} t_z \} \]  

(6)

One more vector R contains the potential of the arc \( (x,y) \), where \( x \in H_1 \), \( y \in H_2 \), \( (x,y) \in E \). This potential is the shortest path from the initial point to y. The vector R is ordered in increasing order. We do not order in each step, because one or two element will be computed, and it is easy to put them in a vector, which is already ordered. Maximum 2 arcs can be put in the set \( E_{ST} \) in one step.
The initial point of the path \( s \) belongs to the set \( S \). In each step one point is added into the set \( S \). The \( x \in S \) point has the shortest path from the initial points. The length of the shortest path from \( s \) to \( x \) is so called potential. \( \mathbf{D} \) vector contains its values after the algorithm. The potential of \( y \in T \) point is equal to \( \infty \). During the algorithm in each step one point leaves the set \( T \) and goes to set \( S \). The algorithm is finished in \( n \) step, when \( S = N \) and \( T = \emptyset \), that is each point of \( S \) has the potential – the length of the minimal path from the initial point \( s \) to that point.

The new algorithm \( S \) is given above:

S0: Initial step:
\[
S = s, \ T = N - s, \ r = \mathbf{B}(\mathbf{A}(s)) \\
E_{ST} = (s, r), \ \mathbf{D}(s) = 0
\]

H1(1) = s, H2(1) = r, \( \mathbf{R}(l) = t_{uv} \)

Let \( u = s, \ v = r \).

S1: Compute \( E_{ST} \)
\[
E_{ST} = E_{ST} - \{(u, v)\} + \{(v, p), (u, q)\}
\]

where
\[
t_{vp} = \min_{p \neq q} t_{pq}, \ t_{uv} = \min_{uv \iota} t_{uv}
\]

S2: Compute the temporary potential belongs to point \( p \) respectively \( q \) and denote it by \( \overline{p} \) respectively \( \overline{q} \):
\[
\overline{p} = d_v + t_{vp}, \ \overline{q} = d_u + t_{uv}
\]

S3: If \( p \neq q \), then put the \( \overline{p} \) and \( \overline{q} \) values into the ordered vector \( \mathbf{R} \) and the \( u, v \) and the \( p, q \) points into the vector \( \mathbf{H1} \) and \( \mathbf{H2} \).

If \( p = q \), then the two candidate arcs endpoints are the same. Then \( \min(\overline{p}, \overline{q}) \) value would be put into the proper place of vector \( \mathbf{R} \).

Similarly modify the vectors \( \mathbf{H1} \) and \( \mathbf{H2} \) with the result of the above mentioned minimization.

S4: Actually the minimal length tree is grown by the arc \( (\mathbf{H1}(1), \mathbf{H2}(1)) \). Delete the first element of vectors \( \mathbf{H1}, \mathbf{H2} \) and \( \mathbf{R} \). Then
\[
S = S + \mathbf{H2}(1), \ T = T - \mathbf{H2}(1)
\]

and put \( \mathbf{R}(1) \) into the potential vector \( \mathbf{D} \).
\[
\mathbf{D}(\mathbf{H2}(1)) = \mathbf{R}(1), \ \text{and} \ \mathbf{H1}, \mathbf{H2} \ \text{and} \ \mathbf{R} \ \text{is modified by one step.}
\]

S5: If \( T = \emptyset \), then STOP otherwise GO TO S1.

We well show that this algorithm is much faster for sparse matrices than the other tree building ones. The storing place of \( S \) is \( 4n + 2m + 3k \) where \( k \) is the maximal cardinality of the set \( E_{ST} \) \( (k < n) \).

V. CONCLUSION

The multiterminal minimal problem is to find the minimal path between each pair of points of the graph. The fastest algorithms are the matrix algorithms (Warshall [13] and Bellman [5] dynamic model) and the minimal spanning tree algorithms (Ford [10], D’Esopo [8]). The Table I contains the main features of these algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># additions</th>
<th># comparisons</th>
<th>Storage</th>
<th>( p \geq n )</th>
<th>( q \geq m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warshall</td>
<td>( n^3 )</td>
<td>( n^3 )</td>
<td>( n^3 )</td>
<td>( p^2 - n^2 )</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic</td>
<td>( n^4 )</td>
<td>( n^4 )</td>
<td>( n^2 )</td>
<td>( p^2 - n^2 )</td>
<td>-</td>
</tr>
<tr>
<td>Ford</td>
<td>( n^6 / 2 )</td>
<td>( n^6 / 2 )</td>
<td>( n^2 )</td>
<td>( 4(p-n) )</td>
<td>( 2(q-m) )</td>
</tr>
<tr>
<td>D’Esopo</td>
<td>-</td>
<td>-</td>
<td>( n^2 )</td>
<td>( 4(p-n) )</td>
<td>( 2(q-m) )</td>
</tr>
<tr>
<td>S</td>
<td>( n )</td>
<td>( 2n )</td>
<td>( n + 2m )</td>
<td>( 4(p-n) )</td>
<td>( 2(q-m) )</td>
</tr>
</tbody>
</table>

This table contains the main features of the fastest multiterminal minimal path algorithms. The first column shows the number of additions, the second the number of comparisons. The third column contains the number of storage places of the network.

In the fourth column we summarize how many extra memory places are needed if we grow up the number of points from \( n \) to \( p \). The last column contains these data in the case of growing the number of arcs from \( m \) to \( q \).

Next we run the algorithms in the case of complete graph \( n = 10, 40, 60, 80 \) and 100. The given functions were approximate by the functions \( a_1 x^2, a_2 x^3, a_3 x^4 \). As we think the fastest algorithm was in this case the Warshall and next our algorithm. The result of approximation is presented in Table II.

<table>
<thead>
<tr>
<th>Table II</th>
<th>CONSTANT OF RUNNING TIME FUNCTION APPROXIMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a_1 )</td>
</tr>
<tr>
<td>W</td>
<td>0.5653</td>
</tr>
<tr>
<td>D</td>
<td>0.4152</td>
</tr>
<tr>
<td>F</td>
<td>0.1199</td>
</tr>
<tr>
<td>D’Esopo</td>
<td>0.2045</td>
</tr>
<tr>
<td>S</td>
<td>0.0707</td>
</tr>
</tbody>
</table>

Finally we tested the methods in a road network which is usually sparse, and the sparseness is growing if the number of points is grown (see the last two rows of Table IV). In the simplest case it could be supposed that it is a square network which is demonstrated in Table III where the vertical and horizontal lines are the arcs, and the crossing points are the points of the network.

The test networks were \( 10 \times 10, 10 \times 20, 10 \times 30, 20 \times 30 \) and \( 20 \times 40 \). We run only the tree building
algorithms, because the matrix algorithms need much more running time in sparse matrices.

The result of the test is summarized in Table IV.

<table>
<thead>
<tr>
<th>Method</th>
<th>10×10</th>
<th>10×20</th>
<th>10×30</th>
<th>20×30</th>
<th>20×40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ford</td>
<td>496</td>
<td>3596</td>
<td>11836</td>
<td>92986</td>
<td>219281</td>
</tr>
<tr>
<td>D'Esopo</td>
<td>143</td>
<td>1021</td>
<td>3688</td>
<td>29735</td>
<td>84978</td>
</tr>
<tr>
<td>S</td>
<td>105</td>
<td>396</td>
<td>874</td>
<td>2843</td>
<td>4982</td>
</tr>
<tr>
<td>Number of arcs</td>
<td>180</td>
<td>370</td>
<td>560</td>
<td>1150</td>
<td>1540</td>
</tr>
<tr>
<td>Density(%)</td>
<td>3.6</td>
<td>1.8</td>
<td>1.2</td>
<td>0.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

The result was that our method is in each case much better than the best tree building algorithm and in the case of sparse matrices it is faster compared to all other algorithms.

REFERENCES


