TRANSPORT PROBLEMS 2010
PROBLEMY TRANSPORTU Volume 5 Issue 2

route planning, road network, dynamic graph

Gábor SZŰCS
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
2nd Magyar Tudósok Krt. H-1117 Bld. I.B. Room 222, Budapest, Hungary
**Corresponding author*. E-mail: szucs@tmit.bme.hu

ROUTE PLANNING IN DYNAMIC GRAPHS WITH LINEAR CHANGING AND PREPROCESSING FOR SPEED-UP

Summary. The goal of this paper is to work out a concept for route planning in a road network, where the costs of roads are not constant, but changing in a linear way. The solution developed is based on the classical Dijkstra's algorithm, which helps to find the route with minimal cost. The new algorithm takes the varying into account in order to find out the best route. This search refers not only to a moment of the departure but to the whole duration of the travel. A speed-up technique has been developed for preprocessing before run time. This preprocessing phase helps to give back the route with minimal cost for the user quickly in run time query. A numerical example has been presented to show the detailed steps of the algorithm and the speed-up technique.

PLANOWANIE PODRÓŻY W DYNAMICZNYCH GRAFACH Z UWZGLĘDNIENIEM WSTĘPNIE PRZETWORZONEGO I ZMIENIAJĄCEGO SIĘ LINIOWO PRZYSPIESZENIA

Streszczenie. Celem artykułu jest wypracowanie koncepcji planowania tras (trasowania) w sieci drogowej, w której koszty połączeń nie są stałe, lecz zmieniają się w sposób liniowy. Zastosowane rozwiązanie opiera się na klasycznym algorytmie Dijkstra, który umożliwia znajdowanie tras po koszcie minimalnym. Proponowany algorytm uwzględnia dynamiczną różnorodność tras, w celu generowania najkorzystniejszej trasy. Jej poszukiwanie uwzględnia nie tylko momenty rozpoczęcia podróży, ale także czas trwania całej podróży. Technikę przyspieszania (speed-up) rozwinięto, w celu wstępnego przetwarzania przed fazą wykonania. Faza wstępnego przetwarzania pozwala szybciej pozyskać trasę, po koszcie minimalnym dla użytkownika. W artykule zostały zaprezentowane liczne przykłady, w których przedstawiono kolejne kroki algorytmu i techniki przyspieszania.

1. INTRODUCTION

A dynamic graph algorithm maintains a given property on a graph subject to dynamic changes, such as edge *insertions*, edge *deletions* and edge *weight updates*. A dynamic graph algorithm should process queries on this property quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: we say that it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

This paper deals with only weight updates, because our focus is transport area, and in the graph, as representation of road network, there is no sense of the edge insertions and deletions. In the road network the cancelling of the roads and construction of new roads are rare, otherwise these kinds of changes do not occur during the travel time of an individual vehicle, so there is no need to insert and delete.

2. GRAPHS WITH LINEAR CHANGING COSTS

In this paper the investigated graphs are special dynamic graphs: the costs of the edges change linear way (and there are no edge insertions or edge deletions). The goal of the investigation is to find the path with minimal cost. There are some solutions for this at the fully dynamic graphs (Cicerone et. al. 2003, Frigioni et. al. 2003). One of them is based on A* algorithm (Likhachev et. al. 2008), other works deal with shortest path for all node pairs (Demetrescu and Italiano 2006a, 2006b). But these solutions exploit all types of changes (insertions, deletions and weight updates), so in our case are not applicable because of considering only weight updates.

We define *graph with linear changing costs* by the following way: A graph is given with an ordered pair G: = (V, E) comprising a set V of vertices or nodes together with a set E of edges (roads), which connect two nodes. Besides the cost of each edge (from node *i* to j) is given by a function: c(*i*, *j*) = max (a•t+b, c_{min}), where t is time, *a*, *b* ∈ R and -1 < *a* < 1, furthermore c_{min} is the minimal cost (the cost should not decrease below this value). The c_{min} plays the roll at case of only decreasing cost function, where *a* is negative. There is a restriction for all cost function, that the value at zero time (and any time) should be equal or greater than zero.

In the rest of the paper we call the *graph with linear changing costs* briefly as dynamic graph. The defined graph may be directed or undirected, the undirected one can be traced back to such directed one, where each edge is replaced into 2 edges with opposite directions. Hereafter we deal with only directed graph.

3. EXTENDED DIJKSTRA ALGORITHM FOR DYNAMIC GRAPH

3.1. Algorithm for Graph with Linear Changing Cost

At route planning the task is to reach a node (end node) from another node (start node) in the graph at smallest cost. Using the original Dijkstra algorithm (Dijkstra 1959) we can construct an extended procedure for graphs with linear changing costs, which differs from algorithms developed in the literature (Cicerone et. al. 2003, Frigioni et. al. 2003, Demetrescu and Italiano 2006a, 2006b) because of different graph types.

At the extended Dijkstra algorithm the graph and the start node are given, and the algorithm will calculate the smallest costs for every other node and will give that neighbor node (see '`previous`' at the pseudo-code of the algorithm), via which the investigated node can be accessed as the smallest cost path.

Thus at end node we can get the smallest cost, furthermore we can go back node by node via 'previous' nodes: this will be the smallest cost path. Let us denote the time, when the search begin at the start node by t_0 and the cost to end node by s(v|t_0) provided the search has began at the start node at t_0.

The cost between two neighbor nodes is c(u, v | t_e), where t_e is the starting time from node *u* (this is equal to the arrival time via the previous edge to node *u*, because the waiting at nodes is not allowed). The detailed steps of the algorithm can be seen at the next pseudo-code.

1. function extended_alg(graph,start,t_0):
2. do for every v node //begin

3. $s(v|t_0)$:= infinite
4. previous (v) := not defined //end
5. s(start) := 0
6. Q := all node
7. while Q not empty //BEGIN
8. u := node with smallest cost in Q
9. remove u from Q
10. for every v neighbor of u: //begin
11. $q = s(u|t_0) + c(u,v|t_e=s(u|t_0))$
12. if $q < s(v|t_0)$
13. $s(v|t_0) := q$
14. previous (v) := u //end,END
15. return previous ()

The algorithm terminates after investigation of all nodes and guarantees the finding of smallest cost if there were no negative cost edges.

In general the cost may be every type of costs, but in our application area the cost is the time, because in the transport the most important is the smallest travel time.

In this survey the assumptions and declarations of the model are the following:

1. The costs are varying by linear way.
2. The travel time is considered as the cost.

Costs are changing during the crossing via the edges of the graph with changing cost. Because of assumption 2 the final cost (travel time) is different from the predefined one (travel time influences on oneself). Final cost of an edge should be investigated and determined. Let us denote the departure time at a node of the examined edge by t_e. Let us denote the unknown arrival time at the other node of the edge after the crossing through the examined edge by t_x. Finally let us denote the transfer time of the crossing by t_T. The cost is the function of the time: $c(u, v \mid t_e)$. According to assumption 2 the transfer time is equal to time average of the varying cost.

$$t_T = E(c(t)) = \frac{\int_{t_e}^{t_x} c(t)}{t_x - t_e} = \frac{\int_{t_e}^{t_x} c(t)}{t_T} \tag{1}$$

In order to express the t_T, we should calculate the result of the integration. If the t_T does not reach the minimal cost (c_{min}), then the change is simple linear. The integration of the linear function can be calculated.

$$\int_{t_e}^{t_x} c(t) = \left[\frac{a \cdot t^2}{2} + b \cdot t\right]_{t_e}^{t_x} = \frac{a \cdot t_x^{\,2}}{2} + b \cdot t_x - \frac{a \cdot t_e^{\,2}}{2} - b \cdot t_e = \frac{a}{2}\left(t_x + t_e\right) \cdot t_T + b \cdot t_T \tag{2}$$

Based on this and previous equation a new equation can be written:

$$t_T^2 = \frac{a}{2}\left(t_x + t_e\right) \cdot t_T + b \cdot t_T \tag{3}$$

$$t_T = \frac{a}{2}\left(t_e + t_T + t_e\right) + b \tag{4}$$

$$t_T\left(1 - \frac{a}{2}\right) = a \cdot t_e + b \tag{5}$$

$$t_T = \frac{a \cdot t_e + b}{\left(1 - \frac{a}{2}\right)} \tag{6}$$

As can be seen from this equation the parameter *a* should be less, than 2, otherwise the transfer time (t_T) will be infinite. If the parameter *a* is positive, then the transfer time is greater than the intercept parameter *b*. If *a* is negative, then the transfer time is less than parameter *b;* at case of zero the transfer time is equal to the constant parameter *b*.

When the t_T reaches the minimal cost, the transfer time will remain the same: c_{min}. Based on these the unknown arrival time can be written:

$$t_x = t_T + t_e = \max\left(\frac{a \cdot t_e + b}{\left(1 - \frac{a}{2}\right)},\ c_{\min}\right) + t_e \tag{7}$$

3.2. Example for the Algorithm

Fig. 1 shows an example for graph with linear changing cost, where the cost functions are given by [a;b] form.

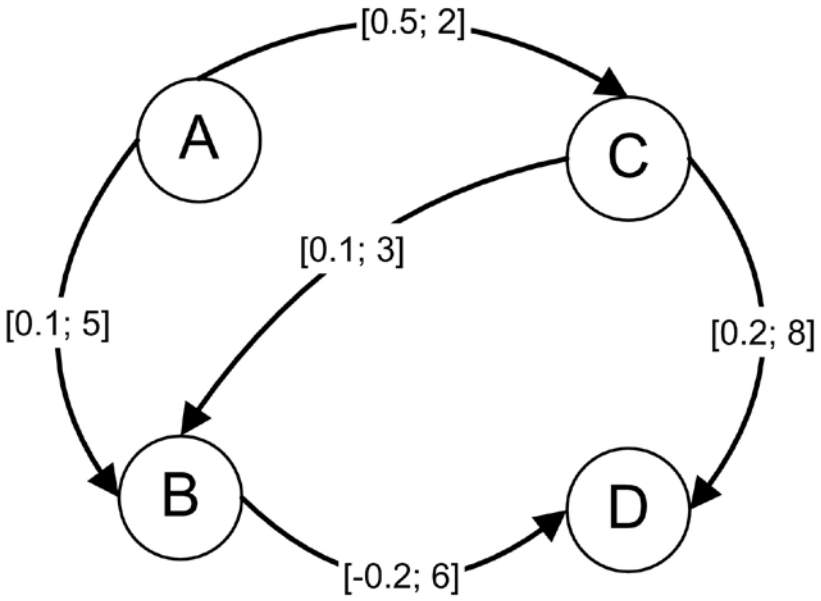

Fig. 1. An example for graph with linear changing cost
Rys. 1. Przykład grafu liniowej zmiany kosztów

Let us follow the extended Dijkstra algorithm step by step from node *A* starting at t_0=0. The aim is the node *D*, but the extended procedure will calculate the smallest costs for all node. The c_{min}=2 for all edges are given, but it can play a role only at edge between *B* and *D*, because this is the only one, where the cost is decreasing. Let us denote the calculated smallest cost of the whole path from start node to a node *u* by *s(u)*. This is the abbreviated notation of $s(u \mid t_0=0)$.

1. step. Explication of start node, i.e. node A:

s(A) = 0
c(A, B | t_e=0) = 5.26316
c(A, C | t_e=0) = 2.66667
s(B) = s(A)+ c(A, B | t_e=0) = 5.26316
s(C) = s(A)+ c(A, C | t_e=0) = 2.66667

The next possible nodes for explication are the node *B* and node *C*, and node *C* has the smallest cost, so in the next step this node will be explicated.

2. step. Explication of node C:

$c(C, D \mid t_e=2.66667) = 9.48148$
$c(C, B \mid t_e=2.66667) = 3.43860$
$s(D) = s(C)+ c(C, D \mid t_e=2.66667) = 12.14815$

Furthermore we should examine, that the earlier s(B) value or new value is smaller:

$s(B) \; ? \; s(C)+ c(C, B \mid t_e=2.66667) = 6.10527$

The earlier s(B) value: 5.26316 is less than 6.10527, so the s(B) will remain the same. The unexplicated node with smallest cost is the node *B*.

3. step. Explication of node B:

$c(B, D \mid t_e=5.26316) = 4.49761$
4.49761 is greater, than c_{min}, so this will be the same.
$s(D) \; ? \; s(B)+ c(B, D \mid t_e=5.26316) = 9.76077$

The earlier s(D) value: 12.14815 is greater than the new one, so the s(D) will be equal to 9.76077. There are no unexplicated nodes, so the procedure terminates. The smallest cost is 9.76077, and corresponding path is *A-B-D*.

4. SPEED-UP TECHNIQUE

4.1. Speed-up Techniques for Static Graph

Although the standard algorithm is fast for finding the smallest cost path in the theory, the corresponding algorithms are often not fast enough for applications in the real networks that require a huge number of the shortest path computations. A speed-up technique is needed, and therefore some algorithms like bidirectional Dijkstra, A* algorithm (Goldberg & Harrelson 2005, Hart et. al. 1968), Arc-Flags, Highway Hierarchies, Reach-based and other algorithms (Goldberg & Werneck 2005, Delling & Wagner 2007) have been developed to solve this problem. The speed-up techniques use preprocessing phase (Lauther 1997, 2004) with some calculations to take the route search in real time more quickly.

The most simple speed-up technique of the search is the bidirectional Dijkstra algorithm. In this procedure, an additional search is started from the target node and the query stops as soon as both searches meet. Applying this approach, the paths of forward and backward search are alternated. This bidirectional Dijkstra algorithm can not be used in the dynamic graph defined in this paper, because we do not know the arrival time at aim node.

The other important technique is the Highway Hierarchies (Sanders & Schultes 2005, 2006), which uses the concept of local search. This approach is a purely hierarchical method, i.e. an approach trying to exploit the hierarchy of a graph. Therefore, the network is contracted and then ‘important’ edges (the so called highway edges) are identified. By rerunning those two steps, a natural hierarchy of the network is obtained. The contraction phase builds the core of a level and adds shortcuts to the graph. The identification of highway edges is done by local Dijkstra executions.

A known speed-up technique of the search is the Arc-Flags (Möhring et. al. 2005, Köhler et. al. 2006) method, which is a modified Dijkstra algorithm in order to avoid exploring unnecessary paths. This means the procedure checks the flag entry of the corresponding target region (the region where

the target node t belongs to) every time before the Dijkstra algorithm wants to traverse an arc. This is the only modification to the standard Dijsktra algorithm. More precisely, the Arc-Flag approach partitions the graph into cells and attaches a label to each edge. A label contains a flag for each cell indicating whether the shortest path to the corresponding cell exists that starts with this edge. As a result, Arc-Flag Dijkstra often only visits those edges which lie on the shortest path of a long-range query.

There is another speed-up technique, Reach-based (Gutman 2004), whose essentials, that the edges have got scores at the preprocessing phase. These scores depend from the location of a long path. Close to two endpoints (the problem is symmetric for the start node and aim node) the scores are low, in the middle of the long path the score are high. In the real-time search the algorithm takes into these scores account. The middle of a long path belongs to global problem (this is a common part of many paths), this is the reason of high scores. The dynamic graph is not symmetric from view of the start node and aim node, therefore this speed-up technique (as the previous ones) is not applicable in our case.

4.2. Speed-up Technique for Dynamic Graph

Our dynamic graph contains the influencing of the time, therefore the speed-up techniques described above can not be used. In our solution we use a long preprocessing before user asks. In this preprocessing phase the speed-up algorithm calculates the minimal costs for every end node at any time. After this parametric calculation the user can give the starting time, desired end node; the procedure will give back the minimal cost and the corresponding path at once by substation of the parameters. The next example will show the parametric calculations of the speed-up algorithm.

4.3. Example for Speed-up Technique

First of all the algorithm examines the decreasing cost parameters. The example graph with costs is the same, as in the previous session, so only one edge has been investigated: the edge between *B* and *D*. The algorithm searches a moment (breakpoint), when the linear cost switches over to fix. At time t_m, the c(B, D | t_e) reaches the minimum, so based on the following equations the corresponding t_m value can be calculated.

$$c(A, B \mid t_e = t_m) = \frac{0.1 \cdot t_m + 5}{0.95} \tag{8}$$

At later calculations it can be seen, that this time interval is the smallest value to the node *B*.

$$c(B, D \mid t_e = t_m + \frac{0.1 \cdot t_m + 5}{0.95}) = \frac{-0.2 \cdot \left(t_m + \frac{0.1 \cdot t_m + 5}{0.95} \right) + 6}{\left(1 - \frac{-0.2}{2} \right)} = -0.20096\, t_m + 4.49761 = c_{min} = 2 \tag{9}$$

From this equation the t_m can be calculated: $t_m = 12.42839$.

At edge of node *B* and *D* before starting time 12.42839 (at start node) the decrease linear cost function is valid, after that time the cost function will be constant. In the following the detailed steps of the preprocessing can be seen.

1. step. Explication of node A

Let us denote the departure time at node *A* by t_Z ($t_Z \geq 0$). Furthermore let us denote the calculated smallest cost of the whole path from start node to a node *u* by *s(u)* as the abbreviation of *s(u | t_Z)*.

$$c(A, C \mid t_e = t_Z) = \frac{a \cdot t_e + b}{\left(1 - \frac{a}{2} \right)} = \frac{0.5 \cdot t_Z + 2}{0.75} \tag{10}$$

$$c(A, B \mid t_e = t_Z) = \frac{0.1 \cdot t_Z + 5}{0.95} \tag{11}$$

$s(B) = c(A, B \mid t_e=t_Z)$? $s(C) = c(A, C \mid t_e=t_Z)$

$(0.1 \cdot t_Z + 5) \cdot 0.75$? $(0.5 \cdot t_Z + 2) \cdot 0.95$

1.85 ? $0.4 \cdot t_Z$

if $t_Z < 4.625$, then $c(A, C \mid t_e=t_Z)$ is smaller. In this point the algorithm bifurcates into two threads. In the next step the node *C* will be explicated if $t_Z < 4.625$, otherwise the node *B* will be explicated.

2. step. Explication of node C if $t_Z < 4.625$:

The node *B* and node *D* should be explicated in this step. The departure time at node *A* is t_Z, the arrival time at node *C* is $s(C) + t_Z$.

$$c(C, D \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = \frac{0.2 \cdot \left(t_Z + \frac{0.5 \cdot t_Z + 2}{0.75} \right) + 8}{\left(1 - \frac{0.2}{2} \right)} \tag{12}$$

$$s(D) = s(C) + c(C, D \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) \; = 1.03704\, t_Z + 12.14815 \tag{13}$$

$$c(C, B \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = \frac{0.1 \cdot \left(t_Z + \frac{0.5 \cdot t_Z + 2}{0.75} \right) + 3}{\left(1 - \frac{0.1}{2} \right)} \tag{14}$$

$$s(C) + c(C, B \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = 0.84211\, t_Z + 6.10526 \tag{15}$$

Furthermore we should examine, that the earlier s(B) value or new value is smaller:

$s(B)$? $s(C) + c(C, B \mid t_e)$

$0.10526 \cdot t_Z + 5.26316$? $0.84211 \cdot t_Z + 6.10526$

if $-1.14284 < t_Z$ then s(B) is smaller. This is always true, because t_Z is equal or greater than zero.

3. step. Explication of node B if $t_Z < 4.625$:

Only the node *D* should be explicated in this step. The departure time at node *A* is t_Z, the arrival time at node *B* is $s(B) + t_Z$.

$$c(B, D \mid t_e = t_Z + \frac{0.1 \cdot t_Z + 5}{0.95}) = \frac{-0.2 \cdot \left(t_Z + \frac{0.1 \cdot t_Z + 5}{0.95} \right) + 6}{\left(1 - \frac{-0.2}{2} \right)} \tag{16}$$

$$s(B) + c(B, D \mid t_e = t_Z + \frac{0.1 \cdot t_Z + 5}{0.95}) = \tag{17}$$

$$= \frac{0.1 \cdot t_Z + 5}{0.95} + \frac{-0.2 \cdot \left(t_Z + \frac{0.1 \cdot t_Z + 5}{0.95} \right) + 6}{1.1} = \text{-0.09569 } t_Z + 9.76077 \quad (18)$$

We should compare this to cost of route to node D: s(D) = 1.03704 t_Z + 12.14815. If t_Z > -2,10763, then new route is smaller. This will be always true because of zero or positive t_Z.

2. step. Explication of node B if 12.14815 > $t_Z \geq$ 4.625:

Only the node *D* should be explicated in this step.

$$s(D) = s(B) + c(B, D \mid t_e = t_Z + \frac{0.1 \cdot t_Z + 5}{0.95}) = -0.09569 \cdot t_Z + 9.76077 \quad (19)$$

3. step. Explication of node C if 12.14815 > $t_Z \geq$ 4.625:

The node *B* and node *D* should be explicated in this step. The cost of the old route to the node *B* is (based on the step 1.): s(B) = 0.10526 t_Z + 5.26316. We should compare this to cost of new route A-C-B.

$$s(C) + c(C, B \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = 0.84211 \; t_Z + 6.10526 \quad (20)$$

if -1.14284 < t_Z then s(B) is smaller. This is always true, because t_Z is equal or greater than zero. We should examine the node *D* as well. The cost of the old route to the node *D*: s(D) = -0.09569 t_Z + 9.76077. We should compare this to cost of new route A-C-D.

$$s(C) + c(C, D \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = 1.03704 \cdot t_Z + 12.14815 \quad (21)$$

if t_Z > -2,10763, then old route is smaller. This will be always true because of zero or positive t_Z.

2. step. Explication of node B if $t_Z \geq$ 12.42839:

Only the node *D* should be explicated in this step.

$$s(D) = s(B) + c(B, D \mid t_e = t_Z + \frac{0.1 \cdot t_Z + 5}{0.95}) \quad (22)$$

where $c(B, D \mid t_e) = 2$

$$s(D) = 0.10526 \; t_Z + 7.26316 \quad (23)$$

(With t_Z = 12.42839 we can take a simple check, where s(D) is equal to 8.5714 by both calculations.)

3. step. Explication of node C if $t_Z \geq$ 12.42839:

The node *B* and node *D* should be explicated in this step. The cost of the old route to the node *B* is: s(B) = 0.10526 t_Z + 5.26316. We should compare this to cost of new route A-C-B.

$$s(C) + c(C, B \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = \quad (24)$$

$$= 0.84211 \cdot t_Z + 6.10526 \quad (25)$$

if -1.14284 < t_Z then s(B) is smaller. This is always true, because t_Z is equal or greater than zero. We should examine the node *D* as well. The cost of the old route to the node *D*: s(D) = 0.10526 t_Z + 7.26316. We should compare this to cost of new route A-C-D.

$$s(C) + c(C, D \mid t_e = t_Z + \frac{0.5 \cdot t_Z + 2}{0.75}) = 1.03704 \cdot t_Z + 12.14815 \quad (26)$$

if $t_Z > -5{,}24264$, then old route is smaller. This will be always true because of zero or positive t_Z.

Final numerical result

The best route from node *A* to *D* at any time is A-C-D. The minimal cost of this route is equal to -0.09569 t_Z + 9.76077 if the starting time $t_Z < 12.42839$; otherwise is equal to 0.10526 t_Z + 7.26316. Based on these preprocessing calculations the best route and the corresponding cost can be given back quickly.

5. CONCLUSIONS

The route planning is important in many area of the transport: logistics (Kiisler 2008), urban traffic (Matis 2008), city transport (Daunoras et. al. 2008). In this paper a solution has been worked out for route planning in a road network, where the costs of roads are changing in linear way. The algorithm developed is based on the classical Dijkstra's algorithm, which helps to find the route with minimal cost.

The new algorithm works on dynamic graph and it takes the varying into account in order to find out the best route. This search refers not only to a moment of the departure but to the whole duration of the travel, in the future as well. A speed-up technique has been developed for preprocessing of the calculations before run time. This preprocessing phase helps to give back the route with minimal cost for the user quickly in run time query. This concept helps in route planning and extends the information with trends in the future. This is useful for such traffic, which can be forecasted in linear way.

6. ACKNOWLEDGEMENT

This work was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Science.

References

1. Cicerone S., Stefano G. D., Frigioni D., Nanni U.: *A fully dynamic algorithm for distributed shortest paths*, Theoretical Computer Science, Volume 297, Issues 1-3, 17 March, 2003, pp. 83–102.
2. Daunoras J., Bagdonas V., Gargasas V.: *City transport monitoring and routes optimal management system*, Transport. Vol. 23, No. 2, pp. 144–149.
3. Delling D., Wagner D.: *Landmark-Based Routing in Dynamic Graphs*. In: 6th Workshop on Experimental Algorithms (WEA), 2007, pp. 52–65.
4. Demetrescu C. Italiano G. F.: *Dynamic shortest paths and transitive closure: Algorithmic techniques and data structures*, Journal of Discrete Algorithms, Volume 4, Issue 3, September, 2006, pp. 353–383.
5. Demetrescu C., Italiano G. F.: *Fully dynamic all pairs shortest paths with real edge weights*, Journal of Computer and System Sciences, Volume 72, Issue 5, August, 2006, pp. 813–837.
6. Dijkstra E. W.: *A note on two problems in connexion with graphs*. In Numerische Mathematik, Volume 1, Amsterdam, The Netherlands: Mathematisch Centrum. 1959, pp. 269–271.
7. Frigioni D., Marchetti-Spaccamela A., Nanni U.: *Fully dynamic shortest paths in digraphs with arbitrary arc weights*, Journal of Algorithms, Volume 49, Issue 1, October, 2003, pp. 86-113.
8. Goldberg A.V., Harrelson C.: *Computing the shortest path: A* meets graph theory*. In: 16th ACM-SIAM Symposium on Discrete Algorithms, 2005, pp. 156–165.

9. Goldberg A.V., Werneck R.F.: *An efficient external memory shortest path algorithm.* In: Algorithm Engineering and Experimentation (ALENEX), 2005, pp. 26–40.
10. Gutman R. J.: *Reach-based routing: A new approach to shortest path algorithms optimized for road networks.* In Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX) and the First Workshop on Analytic Algorithmics and Combinatorics (ANALCO), New Orleans, LA, USA, Philadelphia, PA, USA, SIAM, 2004, pp. 100–111.
11. Hart P.J., Nilsson N.J., Raphael, B.: *A formal basis for the heuristic determination of minimum cost paths.* IEEE Transactions on Systems Science and Cybernetics 4. 1968, pp. 100–107.
12. Kiisler. A.: *Logistics in Estonian business companies,* Transport. 2008, Vol. 23, No. 4, pp. 356–362.
13. Köhler E., Möhring R.H., Schilling H., Hilger, M.: *Fast Point-to-Point Shortest Path Computation with Arc-Flags.* In: 9th DIMACS Challenge on Shortest Paths, 2006.
14. Lauther, U.: *Slow preprocessing of graphs for extremely fast shortest path calculations,* Lecture at the Workshop on Computational Integer Programming at ZIB, 1997.
15. Lauther, U.: *An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background.* In: Geoinformation und Mobilitat – von der Forschung zur praktischen Anwendung, Volume 22., IfGI prints, Institut für Geoinformatik, Münster, 2004, pp. 219–230.
16. Likhachev M., Ferguson D., Gordon G., Stentz A., Thrun S.: *Anytime search in dynamic graphs,* Artificial Intelligence, Volume 172, Issue 14, September, 2008, pp. 1613–1643.
17. Matis P.: *Decision support system for solving the street routing problem,* Transport, 2008. Vol. 23, No. 3, pp. 230–235.
18. Möhring R. H., Schilling H., Schütz B., Wagner D., Willhalm T.: *Partitioning graphs to speed up Dijkstra's algorithm.* In: 4th International Workshop on Efficient and Experimental Algorithms, 2005, pp. 189–202.
19. Sanders P., Schultes D.: *Highway hierarchies hasten exact shortest path queries.* In 13th European Symposium, on Algorithms (ESA), Volume 3669 of LNCS, Springer, 2005, pp 568–579.
20. Sanders P, Schultes D.: *Engineering highway hierarchies.* In: 14th European Symposium on Algorithms (ESA), Volume 4168 of LNCS, Springer, 2006. 804–816

Received 18.06.2009; accepted in revised form 20.06.2010