A label-setting algorithm for calculating shortest path trees in sparse networks. *

L. Marton†

HU ISSN 1418-7108: HEJ Manuscript no.: ANM-030323-A

Abstract

Solving practical network problems by computers, one of the main tools is to find the tree of shortest paths starting from a single point as starting point. In different applications one frequently has such problems, e.g. in traffic organization, where the corresponding network is sparse, i.e., for each vertex, the number of edges directed from it, is bounded, and one has to determine the trees of shortest paths starting from all or almost all vertices. In the present work, a tree building by label-setting algorithm is presented for solving such problems, and the efficiency of the algorithm is investigated. The latter one is supported by computer demonstrations. The algorithm considered is such a version of the well-known label-setting method, which is based on the length preordering of edges directed from a given vertex.

1 Introduction

By a *network* we mean a simple, directed, connected graph, where each edge has a nonnegative value, which is called the *length* of the edge. The vertices of the network are denoted by positive integers. Let the vertex with the greatest index be N, furthermore, (i, j) denote the edge from vertex i to vertex j, and $h(i, j)$ the length of this edge. By a *path* from vertex i to vertex j we mean a sequence $P(i, j) = (i_1, j_1), \ldots, (i_k, j_k)$ of edges, with $i_1 = i$, $j_k = j$ and $j_l = i_{l+1}$, $1 \leq l < k$. Vertices i, j are called the *starting point* and *endpoint* of the path, respectively. By the *length* of a path we mean the sum of the lengths of the edges contained in this path. If a path has the same starting and end point, then we call it a *cycle*. A network is called *symmetric* if the existence of edge (i, j) implies that of edge (j, i) and $h(i, j) = h(j, i)$. Our example (Figures 1., 2., 3.) is symmetric, therefore, we do not write directions for the edges on the figures, in fact, each edge on the figures represents two edges. In the present work, the symmetry has no importance.

A subnetwork is called a *directed tree* if it does not contain cycle, and there is a vertex in it, called *root* or *starting point* with the following property: for each vertex in the subnetwork distinct from the root, there is exactly one path from the root to it. By *minimal tree* we mean such a directed tree in which each path has the minimal length among the paths leading from its starting point to its endpoint in the original network (cf. Figure 1., vertex 1).

We can easily describe a directed tree with labels where for each vertex (except the root), the label denotes the preceding vertex on the path leading from the root to the vertex considered. In a directed tree, each vertex has a *distance* which is the length of the path from the root to it (see Figure 2., 3.). For vertex i, let $c(i)$ and $t(i)$ denote its label and its distance, respectively. The distance of the root according to the definition is 0, and if it is necessary, from technical point of view, we shall also give a label to the root. This label can be any distinguished label (e. g. the root itself or 0).

Algorithms for determining the minimal trees by some tree building technique are important from both theoretical and practical point of view. One reason is that there are not known more efficient algorithms to search the shortest path locally for two vertices. Furthermore, these algorithms have computer implementations which preserve the theoretical efficiency. The latter property is not valid

*Translation from Hungarian, Alkalmazott Mat. Lapok 19, No.2, 115-132 (1999).

†Széchenyi István University, Egyetem tér 1., H-9026 Győr, Hungary

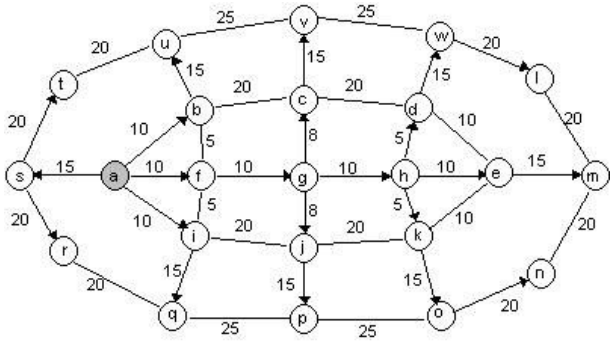

Figure 1. Minimal tree

for other algorithms known for finding the shortest paths between each pair of vertices (e.g. Warshall algorithm), thus, in practical problems with enormous networks it can be very difficult to use these procedures. A summary of this topic and further references can be found in the works [2] and [3].

2 Basic algorithm

One of the basic tree building algorithms is the general label-setting method or Dijkstra-type algorithm which has several versions. Here, one version of this algorithm is described. For this purpose, let us define two subsets K and A of vertices as follows.

- Let K be the set of *finished vertices* (*ready set*). A vertex is *finished* if it has the final (the shortest) distance and final label.
- Let A be the set of *active vertices* (*active set*). A vertex is *active* if it has a temporally label and a finite distance but, the investigation of the vertex have not yet been finished.

Algorithm

- **Step a.** Assign distance 0 to the root and the infinite distance to the further vertices. Let the label of the root be itself. Furthermore, let K be the empty set, and A be the singleton set containing the root. Proceed to Step b.
- **Step b.** Choose a vertex from A with the minimal distance and denote it by i. (If there are more vertices with minimal distance, then choose the smallest one.) Delete i from A, and equip K with it. Proceed to Step c.
- **Step c.** For each edge (i, j) with $t(i) + h(i, j) < t(j)$:
  - **c1.** If $j \notin A$, then put j into A, and modify its label and distance as follows:
    $$c(j) := i, \quad t(j) := t(i) + h(i, j).$$
  - **c2.** If $j \in A$, then modify the label and distance of j by
    $$c(j) := i, \quad t(j) := t(i) + h(i, j).$$
- **Step d.** Terminate if A is empty. Then, the labels determine the minimal tree. Otherwise, proceed to Step b.

The first few steps of the algorithm are demonstrated on Figures 2a-2b.

The proof of correctness of the algorithm is well-known (see e. g. [1]). Regarding this proof, we remark that more minimal trees can exist under a fixed root. The distances of vertices in two different minimal trees have to be the same but the labels can be different.

As far as the efficiency of the algorithm above is concerned, it is obvious that the time complexity is $O(N^2)$. Indeed, at Step b, we pass one vertex from set A into set K, consequently, the number of iterations is N. Furthermore, in each iteration, we have to find one element with minimal distance

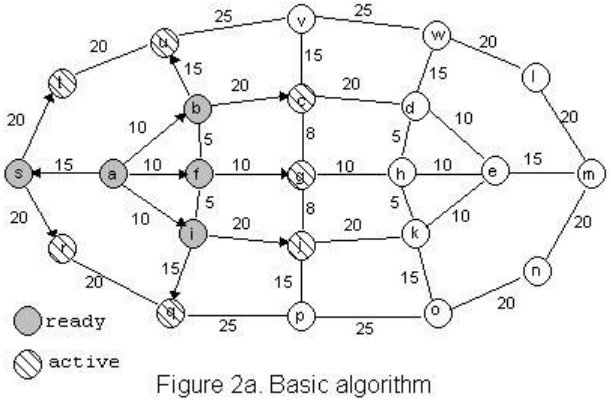

Figure 2a. Basic algorithm

1	K	a																				
	A	b	f	i	s																	
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
	C	a	a				a			a										a				
	T	0	10	∞	∞	∞	10	∞	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	15	∞	∞	∞	∞
2	K	a	b																			
	A	f	i	s	u	c																
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
	C	a	a	b			a			a										a		b		
	T	0	10	30	∞	∞	10	∞	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	15	∞	25	∞	∞
3	K	a	b	f																		
	A	i	s	g	u	c																
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
	C	a	a	b			a	f		a										a		b		
	T	0	10	30	∞	∞	10	20	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	15	∞	25	∞	∞
4	K	a	b	f	i																	
	A	s	g	q	u	j	c															
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
	C	a	a	b			a	f		a	i							i		a		b		
	T	0	10	30	∞	∞	10	20	∞	10	30	∞	∞	∞	∞	∞	∞	25	∞	15	∞	25	∞	∞
5	K	a	b	f	i	s																
	A	g	q	u	j	c	r	t														
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
	C	a	a	b			a	f		a	i							i	s	a	s	b		
	T	0	10	30	∞	∞	10	20	∞	10	30	∞	∞	∞	∞	∞	∞	25	35	15	35	25	∞	∞

Figure 2b. Basic algorithm

among at most $N-1$ elements (set A), and we have to investigate the inequality, $t(i)+h(i,j)<t(j)$, at most $N-1$ times. It is worth noting that set K has no importance, it only makes easy the description of the procedure (and the proof of correctness), thus, we can omit it in computer realizations.

It is easy to prove that any tree building algorithm has to investigate each edge in the network at least one times. We can see that the Dijkstra algorithm introduced above investigates each edge exactly one times. Hence, the efficiency of each procedure which is derived from this algorithm and preserves the basic ideas (one times edge examination, *final* set, *active* set, choosing the element with minimal distance) depends on the following factors:

- *The number of elements in the active set.*
- *The data structure which stores and handles the active set.*

Regarding the second factor, many computer implementations are known, they are summarized below. We can classify the computer implementations known for the active set as follows:

- α. *Disordered* (set, array, list)
- β. *Linearly ordered* (array, list)
- γ. *More-level or partially ordered* (linked lists, trees, heap,.etc)

The ordering always means the increasing (nondecreasing) ordering of the vertices on the distances.

Analyzing the steps concerning the active set, we can conclude the following observations regarding the possible implementations of the active set.

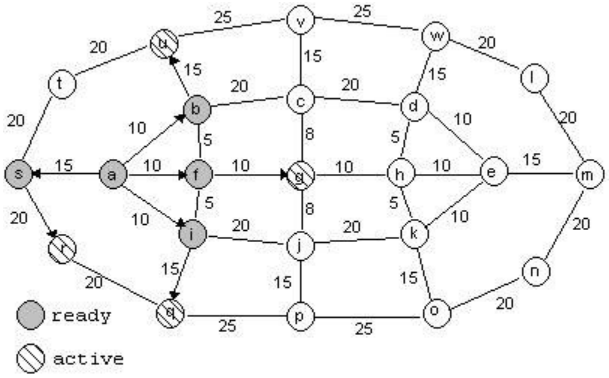

Figure 3a. Modified algorithm

- Step b: To find an element with the minimal distance needs $O(|A|)$ time for α type implementation. For β and γ type implementations we get the required element immediately (first element, top element). Deleting can be executed immediately for α and β type implementations and it usually has the same time complexity as inserting for γ type implementation.
- Step c1: This step is immediate for α type implementation but for β and γ type implementations, the time complexity of finding the place of a new element is a function of $|A|$ (in the case of β, this function is linear, in the case of γ, it is logarithmic).
- Step c2: The time complexity is similar as in Step c1. The difference is that for β and γ type implementations, we need to find the new place of one element in A and we have to remove it.

3 Modified procedure

The basic ideas of the modified procedure can be sketched as given below.

From the above analysis, we can conclude that the decrease of the number of active elements in general results in a more efficient algorithm. Investigating problem instances (see Figure 2), it seems that the active set is too large, there are a lot of vertices, which have to wait for more steps to become minimal. Hence, it would be very practical to put less vertices into the active set in each step, mainly such vertices which has a greater chance to become minimal.

We can carry out this idea if for each vertex, we order the edges directed from it on increasing (nondecreasing) lengths. For special type of networks, this *preordering* does not cause essentially larger time complexity. For example, if

- *for each point of the network, the number of edges directed from it is bounded by a constant* $k << N$, *and*
- *we need to determine all (or relatively a lot) minimal trees,*

then the preordering yields approximately k^2 operations for each tree which is a negligible small constant regarding N^2.

The preordering makes us possible to put less vertices into the active set during the iterations, but of course we have to be careful, we always must put the vertex which will have minimal distance during the next iteration into the active set.

The idea of the preordering as an improving factor was occurred in an algorithm presented in [5], and one can also find a tree building algorithm in [2] which is based on a similar preordering.

In this paper, preserving the basic properties of the Dijkstra procedure, we present a new tree building technique with label-setting, and we prove its correctness and its theoretical efficiency.

Presentation of the modified algorithm

For the sake of unique representation, we note that if two edges have the same length, then we order forward that, which has the smaller endpoint. We use the preordering with the help of a function denoted

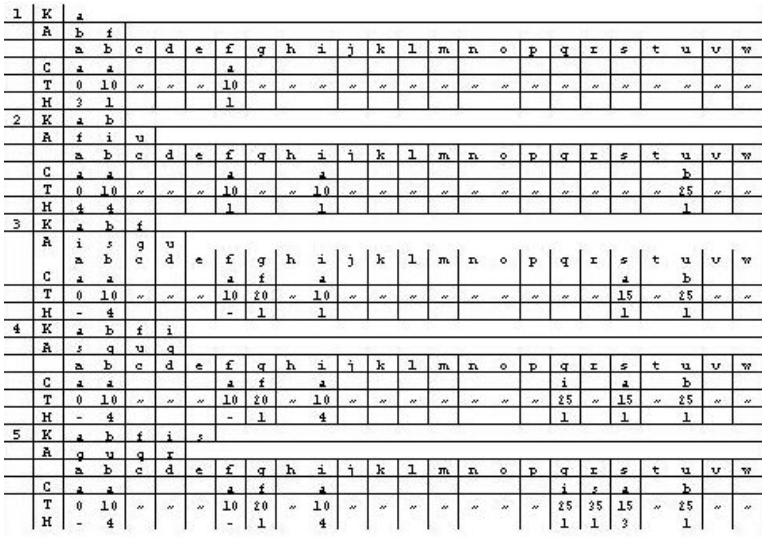

Figure 3b. Modified algorithm

by m. During the procedure $m(i)$ provides the smallest edge directed from vertex i which have not yet investigated. At the beginning, for each i, which is not sink, let $m(i) = 1$.

Modified algorithm

- **Step a'.** Let the distance and label of the root be 0 and itself, respectively, and let us assign infinite distance to the further vertices. Furthermore, let $K' = \emptyset$ and A' be the singleton set containing the root. For each vertex i, which is not sink, let $m(i) = 1$.
- **Step b'.** Choose one element from A' with the minimal distance, denote it by i'. (If there are more vertices which have minimal distance choose the smallest element among them.) Delete vertex i' from set A' and put it into set K'. Let $e = c'(i')$ be the label of vertex i'. Proceed to Step c'.
- **Step c'.** Proceed the following (c1'-c4') sub-procedure for both e and i'.
  - **Step c1'.** Denote the examined vertex by x.
  - **Step c2'.** For edge (x, y) determined by $m(x)$, examine property $t'(x) + h(x, y) < t'(y)$. If this property holds, then proceed to Step c3'. The sub-procedure is ended if this property does not hold and $m(x)$ has the maximal value. Otherwise, let $m(i) := m(i) + 1$ and proceed to Step c2'.
  - **Step c3'.** Proceed to Step c4', if $y \in A'$. Otherwise, put y into set A', and modify its label, its distance, and the value of $m(x)$ as follows:

    $$c'(y) := x, \quad t'(y) := t'(x) + h(x, y), m(x) := m(x) + 1.$$

    Then, the sub-procedure terminates.
  - **Step c4'.** Since $y \in A'$, it has a label $c'(y)$. Let $z = c'(y)$, furthermore, as in step c3' let:

    $$c'(y) := x, \quad t'(y) := t'(x) + h(x, y), m(x) := m(x) + 1.$$

    Finally, let $x := z$ and proceed to Step c2'.
- **Step d'.** Terminate, if $A' = \emptyset$. Then, the minimal tree is determined by the produced labels. Otherwise ($A' \neq \emptyset$), proceed to Step b'.

We show some starting steps of the procedure (see Figures 3.a-b).

We represent the algorithm by its flowchart (cf. Figure 4.), furthermore, demonstrate the work of the main sub-procedure separately (cf. Figure 5).

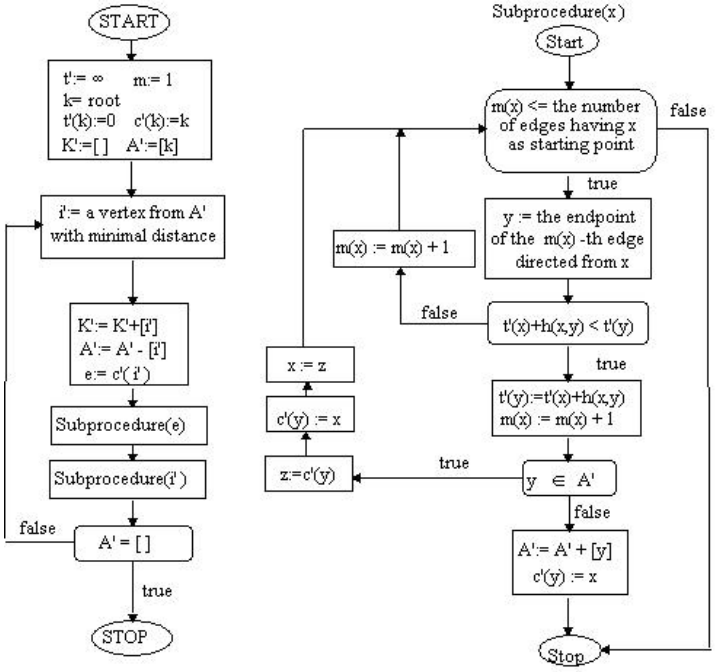

Figure 4. The flowchart of the modified algorithm

Correctness

Regarding the correctness of the modified procedure, we have to prove that it gives a minimal tree. This statement is proved in more steps. First, using the notations of Figure 5, we examine the sub-procedure which enlarge the active set with y_u and decrease the distance of active vertices y_l, $l = 1, \dots, u-1$.

Let $\underline{t'(z)}$, $\underline{c'(z)}$ denote the distance and label of vertex z before the sub-procedure and $\widetilde{t'(z)}, \widetilde{c'(z)}$ the distance and label of vertex z after the sub-procedure. The preordering on the vertices and the construction of the sub-procedure imply the following relations.

$$\underline{c'(y_1)} = x_2, \dots, \underline{c'(y_l)} = x_{l+1}, \dots, \underline{c'(y_{u-1})} = x_u,$$

$$\underline{t'(y_1)} = \underline{t'(x_2)} + h(x_2, y_1), \dots, \underline{t'(y_{u-1})} = \underline{t'(x_u)} + h(x_u, y_{u-1}),$$

$$(3.1) \quad h(x_2, y_1) \le h(x_2, y_2), \dots, h(x_u, y_{u-1}) \le h(x_u, y_u),$$

$$\widetilde{c'(y_1)} = x_1, \dots, \widetilde{c'(y_l)} = x_l, \dots, \widetilde{c'(y_u)} = x_u,$$

$$\widetilde{t'(y_1)} = \widetilde{t'(x_1)} + h(x_1, y_1), \dots, \widetilde{t'(y_u)} = \widetilde{t'(x_u)} + h(x_u, y_u).$$

Let us investigate the relationship between the new distances of y_l and y_{l+1}, $l = 1, \dots, u-1$. The sub-procedure selects y_l with the following property:

$$(3.2) \quad \widetilde{t'(y_l)} = \widetilde{t'(x_l)} + h(x_l, y_l) < \underline{t'(x_{l+1})} + h(x_{l+1}, y_l) = \underline{t'(y_l)}.$$

The distance of x_{l+1} does not change during the sub-procedure, therefore, $\underline{t'(x_l)} = \widetilde{t'(x_l)}$. This observation and inequalities 3.1, 3.2 together with $\widetilde{t'(y_{l+1})} = \widetilde{t'(x_{l+1})} + h(x_{l+1}, y_{l+1})$ imply the following inequality:

$$(3.3) \quad \widetilde{t'(y_l)} < \widetilde{t'(y_{l+1})}.$$

This yields that we have proved the following assertion.

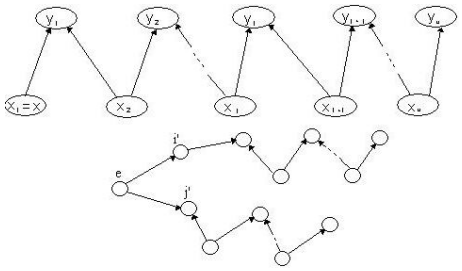

Figure 5. Subprocedure

Lemma 3.1. *During the sub-procedure, the sequence of the modified distances is monotone increasing.*

We perform the sub-procedure in the Step **c** for the vertex i which has the minimal distance in the active set and for the vertex e which is the label of i. Furthermore, by the preordering, we have $h(e,i') \leq h(e,j')$ for each edge (e,j') examined during the performance of the sub-procedure for e (see Figure 5.). These observations and Lemma 3.1 imply the following statement.

Lemma 3.2. *After the execution of Steps b, and c in the algorithm the distance of any vertex in the active set is not smaller than the distance of the vertex placed into the ready set during the steps considered.*

On the basis of the above lemmas we can prove the following statement.

Lemma 3.3. *The distance of a vertex placed into the ready set is not smaller then the distance of other vertices from the ready set. If two vertices have the same distances, then firstly the smaller vertex is placed into the ready set.*

Proof. The first part of the lemma is an obvious consequence of Lemma 3.2. To prove the second part, contrary, let us suppose that $p \in K'$, $q \in K'$, $t'(p) = t'(q)$, $p < q$ and q was placed into the set K' earlier than p. By Lemma 3.2 and $t'(p) = t'(q)$, we can assume that q was placed into the set K' during the l-th iteration part and q was placed into K' during the $l+1$-th iteration part. By the definition of Step b., the above assumption yields that at the beginning of the l-th iteration we had $t'(q) < t'(p)$ and we decreased the value of $t'(p)$ during the iteration part. On the other hand, by (3.3) and Lemma 3.2, the decrease of $t'(p)$ to $t'(q)$ is only possible in the case when (see Figure 5.) $q = i'$, $p = j'$ and $h(e,i') = h(e,j')$, which contradicts the preordering defined on the edges.

Lemma 3.3 immediately implies the following observation.

Corollary 3.4. *Each vertex placed into the ready set has a final distance and label. These vertices do not become active again during the procedure.*

Furthermore, we have to prove the following statement.

Theorem 3.5. *If the active set becomes empty, then each vertex for which there is a path leading from the root to it has a label and a final distance.*

Proof. For verification by contradiction, let us suppose that there is a vertex with a path leading from the root to it which has no label and finite distance. Let us call these vertices *bad vertices*. If there is a bad vertex, then there is such a one which is an endpoint of an edge having its starting point from the ready set. Let p denote the starting point of this edge. Then, investigating the edges directed from p, some of these edges have bad endpoints. Consider the smallest such edge with respect to the preordering. Denote its endpoint by r (cf Figure 6.). Then, (p,r) is not the first edge in the preordering from p since the endpoint of the first edge has a label and a finite distance when p is placed into the ready set. On the other hand, the endpoints of edges smaller then (p,r) have a label and finite distance by the definition of r. During the iteration, when p was placed into the ready set some of these endpoints received label p. (Otherwise, we would consider vertex r during the sub-procedure of this iteration.) Consequently, there are vertices which are the endpoints of edges smaller then (p,r) and which had p as a label during the procedure. Let denote that one among these vertices which is the endpoint of the greatest edge by u. Now, consider two cases. First, let us suppose that u also has p as the final label. Then, during the iteration part, when u was placed into the ready set we would consider r and assign a label to it or we would assign p as a label to such a vertex which is the endpoint of an edge directed from p which is greater than (p,u) and smaller than (p,r). Therefore, we get a contradiction in this case. Let us suppose that the final label of u is not p. Then, investigating the iteration part in which the label of u was modified,

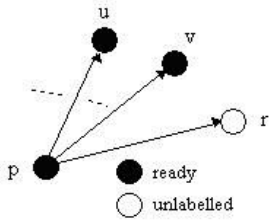

Figure 6. For Theorem 3.5

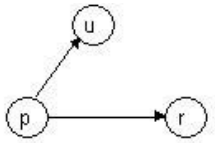

Figure 7. For Theorem 3.6

we get a contradiction in the same way as above. Consequently, we get a contradiction in both case, thus, we proved the statement of the theorem.

Lemma 3.6. *During the performance of the modified procedure, we assign to each vertex a ready vertex as a label.*

Proof. The statement is obviously valid for the root. To prove by contradiction, let us suppose that there is a vertex not satisfying the above statement. Denote by r (see Figure 7.) the vertex firstly investigated during the procedure which had an active label p. Let us examine how the procedure could assign this label to vertex r. There are two possibilities. One possibility is that p had the minimal distance in A'. In this case, at the beginning of the iteration, p became ready and, by Lemma 3.4, p remained ready till the end of the procedure. Hence, this case leads to a contradiction. In the other case the procedure assigned the label p to r, when a vertex u which had the label p was investigated. In this case, if p was not ready, we found a vertex u having an active label which was investigated earlier then r. Hence, this case contradicts the definition of r. Both cases led to a contradiction, thus, we proved the statement.

Lemma 3.7. *If $p \in K'$ and u is the endpoint of an edge directed from p, which is smaller regarding the preordering than the edge determined by $m(p)$, then $u \in K'$ or $u \in A'$, furthermore, $t'(u) \leq t'(p) + h(p,u)$ is valid for the current distances.*

Proof. Firstly, let us observe that during the procedure the values of $t'(x)$ decrease for every x. Since the value of $m(i)$ determines a greater edge than (p,u), the inequality $t'(p) + h(p,u) < t'(u)$ was investigated by the algorithm. If this inequality was valid, then the procedure had changed the distance of u by $t'(u) := t'(p) + h(p,u)$. The investigation of the above inequality happened at the iteration part in which p was placed into the ready set or later, thus, after this investigation the value of $t'(p)$ have not changed. Hence, by the first observation, we proved that the required inequality is indeed hold. On the other hand, $t'(u)$ is finite which yields that $u \in K'$ or $u \in A'$.

Theorem 3.8. *If $i \in K'$, then there is no path in the network leading from the root to i which has smaller length than $t'(i)$.*

Proof. By Lemma 3.4, it is sufficient to prove that this property is hold when a vertex is placed into the ready set. To verify by contradiction, let us suppose that there are vertices not satisfying the above property, when they are placed into K'. Denote by p (see Figure 8.) the first vertex during the performance of the procedure which does not satisfy the statement, furthermore, by r its label. This yields that $t'(p) = t'(r) + h(r,p)$ is a minimal distance among the distances of the active vertices, furthermore, there is a path $(\ldots, v, s, \ldots, p)$ leading from the root to p which has a length $w < t'(p)$. Since p is the first vertex having this property during the procedure, the path does not contain r. Let v denote the last ready vertex preceding p from the path, furthermore, s denote the vertex which follows after v in the path. Let us consider two cases depending on $m(v)$. First, suppose that $m(v)$ determines an edge which is greater regarding the preordering than (v,s). In this case, by Lemma 3.7, $s \in A'$ and $t'(s) \leq t'(v) + h(v,s)$. On the other hand, $v \in k'$, thus, by the definition of p, the length of any path leading from the root to v is

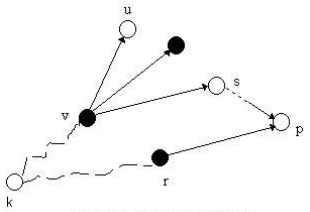

Figure 8. For Theorem 3.8

at least $t'(v)$. Hence, $t'(p) > w \geq t'(v) + h(v,s) \geq t'(s)$, which contradicts the minimal property of $t'(p)$. Now, suppose that $m(v)$ determines an edge which is not greater with respect to the preordering than (v,s). In this case, there is an active vertex u (cf. Figure 8.) for which the edge (v,u) is greater than the edge determined by $m(v)$ and smaller than (v,s). Then, by $t'(v) + h(v,s) \geq t'(v) + h(v,u) \geq t'(u)$, we get a contradiction in a similar way as above. Consequently, we got a contradiction in both cases, thus the statement is proved.

By Theorem 3.5 and 3.8, we proved that the modified algorithm is correct, it produces a minimal tree.

4 The efficiency of the modified algorithm

During the performance of the procedure, each edge is investigated one times (cf. Step c'). This observation follows from the definition of function m. We have seen that under such algorithms the time complexity is determined by the size of the active set. Let us investigate the sizes of the active sets which belong to the basic and to the modified algorithm, respectively. We prove that the active set belonging to the modified algorithm is a subset of the corresponding active set belonging to the basic algorithm in each iteration. For the sake of brevity, we denote the basic algorithm by D and the modified algorithm by D'.

Theorem 4.1. *For every iteration steps of the algorithm,*l

- *the two algorithms choose and place the same vertex with the same distance, into the active set*
- *the active set belonging to algorithm D' is a subset of the active set belonging to algorithm D.*

Proof. We prove the statement by induction. It is obvious that the statement is valid after the first iteration. Let $n \leq N$ and suppose that the statement is valid for each iteration part performed before the n-th iteration part. We prove that then it is also valid after the n-th iteration part. Let us observe that by Lemma 3.5 and the induction hypothesis, $A' \neq \emptyset$ and $A \neq \emptyset$ are valid. Now, we verify that the two algorithms put the same vertices with the same labels and distances into the ready set .

Let i denote the vertex chosen by algorithm D, furthermore, let $e = c(i)$. Since $e \in K$, by the induction hypothesis, we have that $e \in K'$ and $t(e) = t'(e)$ are the final distances of e. Consider the value of $m(e)$ in the case of algorithm D'. If $m(e)$ determines an edge which is not greater according to the preordering than (e,i), then there is a vertex $j \in A'$ for which (e,j) is smaller than (e,i) under the preordering. (Otherwise, $e \in K'$ would implies $i \in K'$ contradicting the induction hypothesis.) On the other hand, in this case, $j \in A$ is also valid. Furthermore, since $e \in K$, the algorithm D investigated each edge directed from e, thus $t(j) \leq t(e) + h(e,i)$ is valid. Consequently, by $h(e,j) \geq h(e,i)$, we have that $t(j) \leq t(e) + h(e,j) \leq t(e) + h(e,i) = t(i)$. Since algorithm D selected i to put into the ready set, the above inequality is possible if and only if $h(e,j) = h(e,i)$ and $i < j$. On the other hand, we know that (e,j) is smaller than (e,i) regarding the preordering which is a contradiction. Hence, we get a contradiction, and therefore, we proved that $m(e)$ determines an edge which is greater with respect to the preordering than (e,i). Then, Lemma 3.7 implies $i \in A'$ and $t'(i) \leq t'(e) + h(e,i) = t(e) + h(e,i) = t(i)$. On the other hand, by the correctness of procedure D and the definition of i, we can conclude that $t(i) \leq t'(i)$. These inequalities imply that $t(i) = t'(i)$.

Let i' denote the vertex chosen by algorithm D', furthermore, let $e' = c'(i')$. By the induction hypothesis, it follows that $i' \in A$. Furthermore, since $e' \in K'$, we have that $e' \in K$. This yields that during the iteration part when e' was placed into set K, algorithm D investigated each edge directed from e'. Consequently, the inequality $t(i') \leq t(e') + h(e',i') = t'(e') + h(e',i') = t'(i')$ is valid. On the other hand, by Theorem 3.8, one can conclude that $t'(i') \leq t(i)$. This gives that $t'(i') = t(i)$.

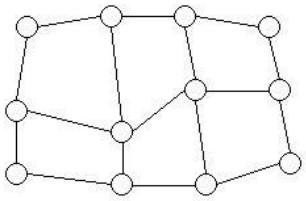

Figure 9. Grid

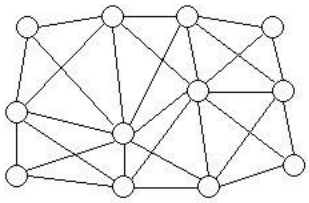

Figure 10. Diagonal grid

We have proved that vertices i and i' are active for algorithms D and D'. Both algorithms choose an active vertex which has the minimal distance, hence $t(i) \leq t(i') = t'(i) \leq t'(i') = t(i)$ is valid and $t(i) = t(i')$, $t'(i) = t'(i')$ follow. On the other hand, if there are more vertices with the minimal distance, then both algorithm choose the smallest one. This yields that $i = i'$, which is the first part of the statement.

In order to prove the second part of the statement, we have to verify that $A' \subset A$ is valid after the performance of the iteration part. We delete $i = i'$ from A and A', and thus, $A' \subseteq A$ is also valid after this operation . Set A contains all vertices, which are not ready vertices and they are the endpoints of edges having ready starting points. The construction of the sub-procedure of D' and Lemma 3.6 imply that A' contains only such vertices which are not ready and they are the endpoints of edges having ready starting points. Furthermore, by the induction hypothesis, the two ready sets are the same. These observations show that $A' \subseteq A$ is also valid after the performance of the iteration part.

The decrease of the size of the active sets, which we can achieve by the modified algorithm, depends on the considered network and root, we do not give a general estimation for it. However, we can conclude a connection between the average number of edges and the decrease of the active sets by the following statement.

Theorem 4.2. *During algorithm D', each ready vertex is assigned to at most one active vertex as a label.*

Proof. For each $p \in K'$ let us denote by $a(p)$ the number of active vertices with label p. Investigate the behavior of $a(p)$ during the procedure. Let us observe that during the sub-procedure (cf. Figure 5.) on the basis of Lemmas 3.4 and 3.6, $x_1, \ldots, x_u$ are ready vertices and $y_1, \ldots, y_u$ are active vertices, and therefore, the values of $a(x_2), \ldots, a(x_u)$ do not change. If $x_1 = x = i'$, then $a(x)$ is at most 1 since i' as a new ready vertex has not been assigned to any vertex earlier. Finally, if $x_1 = x = e$, then the increase of $a(x)$ is at most 1 during the sub-procedure, but the algorithm decreases this value before the performance of the sub-procedure putting i' into the ready set. Consequently, for each $p \in K'$, the value of $a(p)$ being at most 1 when the vertex becomes ready can not increase during the procedure.

The above statement is not valid for the basic algorithm. In the basic algorithm, if a vertex is placed into the ready set, then each not ready vertex which is endpoint of an edges directed from the vertex considered becomes active. Thus, by Lemma 4.2, we can expect that if the average number of vertices is greater, then the difference between the average size of A' and A increases. This statement can be illustrated by the following examples.

Networks considered

In the following networks, the lengths of the edges was generated randomly over $[1, 10]$ under uniform distribution.

- **R2015, R3030**: Grid type networks with sizes 20×15 and 30×30 (see Figure 9).
- **S2015, S3030**: Diagonal grid type networks with sizes 20×15 and 30×30 (cf. Figure 10).

Network	R2015	R3030	S2015	S3030	V 3 4	V 9 4	V 3 8	V 9 8
Vertex	300	900	300	900	300	900	300	900
Edge	1130	3480	2194	6844	1200	3600	2400	7200
Edge/Vertex	3.66	3.86	7.31	7.60	4.00	4.00	8.00	8.00
DA,DB,DC Set	22.39	36.35	36.48	57.03	75.43	224.96	113.54	337.38
D'A,D'B,D'C Set	18.26	28.93	24.21	37.37	54.60	163.92	70.68	210.41
Set %	**82**	**80**	**66**	**66**	**72**	**73**	**62**	**62**
DA Step	6719	32725	10944	51326	22629	202412	34063	303643
D'A Step	5478	26044	7263	33631	16380	147571	21204	189375
D'A Step %	**82**	**80**	**66**	**66**	**72**	**73**	**62**	**62**
DB Step	1939	9283	3704	17128	7311	67005	15046	136876
D'B Step	1711	8453	2736	12234	5437	50186	7498	66105
D'B Step %	**88**	**91**	**73**	**71**	**74**	**74**	**50**	**48**
DC Step	1727	5801	2053	6779	2184	8000	2491	8766
D'C Step	1620	5517	1803	5943	2033	7438	2175	7887
D'C Step %	**94**	**95**	**88**	**88**	**93**	**93**	**87**	**90**

Figure 11. Computational results

- **V-3-4, V-3-8**: Networks having 300 vertices where for each vertex, there are 4 (8) edges directed from it and 4 (8) edges directed to it.
- **V-3-4, V-3-8**: The same networks as above with 900 vertices.

Algorithms considered

- **DA** : The basic algorithm where the active set has α type computer implementation.
- **D'A** : The modified algorithm in which the active set has α type computer implementation.
- **DB** : The basic algorithm where the active set has β type computer implementation.
- **D'B** : The modified algorithm where the active set has β type computer implementation.
- **DC** : The basic algorithm in which the active set has γ type computer implementation.
- **D'C** : The modified algorithm where the active set has γ type computer implementation.

For each vertex of the networks we determined the minimal trees by using the above algorithms. The obtained average results are presented by the table of Figure 11. The algorithms were studied from the following points of view:

- **set**: The average size of the active set.
- **step**: The average number of the *minimal searching, inserting, removing* steps required to determine one minimal tree.
- % : The ratio of the attributes of the modified algorithm and basic algorithm per cent.

We have not investigated the computer run-time. The run-time highly depends on the programming language and the coding of the algorithm. A theoretically better algorithm with a less effective coding can have worst run-time than a theoretically worst algorithm with a more efficient coding.

We can measure the efficiency with the average size of the active set. Considering the investigated networks, the decrease of the average size of the active set (set % line in Figure 11) is about $18 - 38$ percent. Depending on the implementation of the active set, we can see this improvement considering the decrease of the required number of the inserting and removing steps.

The above presented algorithm with γ type implementation of the active set was used to solve practical problems in the area of traffic organization. Such applications can be found in [6], [7], [8], and [9].

References

[1] J. R. Evans, E. Minieka, *Optimization algorithms for networks and graphs* Marcel Dekker Inc., New York, 1992.

[2] A. Bakó, Traffic guiding by computer, *Alkalmazott matematikai lapok* **6** (1980), 351-392 (in Hungarian).

[3] N. Deo, C. Pang, Shortest Path Algorithms: Taxonomy and Annotation, *Networks* **14** (1984), 257-323.

[4] E. V. Denardo, B. L. Fox, Shortest Route Methods: Reaching, Pruning and Buckets, *Operations Research* **27** (1979), 161-186.

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

[6] *Ergänzende Verkehrsprognose für die Grenzüberschreitenden Strassen.* Ingenieurbüto Kriebernegg, Graz, 1992.

[7] *Traffic survey, forecast and sensitivity analysis of tolls for evaluating the tender of Motorway M5 in Hungary.* Bouygues (France) - Bauconsult (Hungary), Győr, 1993.

[8] *Traffic and charge returns survey on the Motorway M3.* (Object Budapest-Bauconsult), Győr, 1996, (in Hungarian).

[9] *The determination of the regularity and features of journeys taken place on the highway network in Hungary.* (Bauconsult) Győr, 1997, (in Hungarian).