The ε-approximation of the Label Correcting Modification of the Dijkstra's Algorithm

František Kolovský[1], Jan Ježek[1] and Ivana Kolingerová[2]
[1]*Depatment of Geomatics, University of West Bohemia, Univerzitní 2732/8, Plzeň, Czech Republic*
[2]*Department of Computer Science, University of West Bohemia, Univerzitní 2732/8, Plzeň, Czech Republic*

Keywords: Time-dependent Shortest Path Problem, Approximation, Travel Time Function, Road Network.

Abstract: This paper is focused on searching the shortest paths for all departure times (profile search). This problem is called a *time-dependent shortest path problem* (TDSP) and is important for optimization in transportation. Particularly this paper deals with the ε-approximation of TDSP. The proposed algorithm is based on a label correcting modification of Dijkstra's algorithm (LCA). The main idea of the algorithm is to simplify the arrival function after every relaxation step so that the maximum relative error is maintained. When the maximum relative error is 0.001, the proposed solution saves more than 95% of breakpoints and 80% time compared to the exact version of LCA. A more efficient precomputation step for another time-dependent routing algorithms can be built using the developed algorithm.

1 INTRODUCTION

Computing the arrival function from a source node to all other nodes is important for a lot of transportation applications. More formally, given a directed graph $G = (V,E)$, a source node $s \in V$, we want to know the travel time between the source node s and all other nodes for every departure time (in some literature called a *travel time profile*). This problem is generally called the *time-dependent shortest path problem* (TDSP). The main principle is that the arrival time t_u at the node u is used as the argument of the arrival time function f corresponding with the edge that origins at u.

The common approach is to use a piecewise linear function as a realization of the arrival function. Let us have two consecutive edges (e.g, the edges (s,u) and (u,d) in Figure 1a) then the arrival time at the node d is the value of the arrival function f_{ud} in the arrival time $f_{su}(t_d)$ at the node u, where t_d is the departure time at the node s. In the exact case, the combination $f_2(f_1(t))$ of two piecewise linear functions f_1, f_2 with $|f_1|$, $|f_2|$ linear pieces is also a piecewise linear function with up to $|f_1| + |f_2|$ linear pieces (Foschini et al., 2014). It means that the arrival function at the end of the path with n edges can have up to $\sum_{i=1}^{n} |f_i|$ linear pieces. For example, the path across the Pilsen city has around 100 edges. If every arrival function on the path has 24 linear pieces, the resulting arrival function has 2400 liner pieces. It can be seen that the computational time and memory requirements strongly increases with the length of the paths.

The problem with an increase in the number of linear pieces can be solved using the ε-approximation of the resulting arrival function. This approach reduces the number of linear pieces and thus reduces memory requirements as well as computation time. Our proposed algorithm is based on the so-called label correcting modification of Dijkstra's algorithm (Orda and Rom, 1990). The main idea is to perform a simplification of the arrival functions during the computation with a suitable maximum absolute error such that the relative error ε is maintained.

2 DEFINITIONS AND PRELIMINARIES

2.1 Road Network

Let $G = (V,E)$ be a directed graph that represents a road network, where V is a set of nodes and E is a set of edges. Each edge $(u,v) \in E$ has an *arrival function* (AF) $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ that for the given departure time at u returns the arrival time at v. Alternatively, we can define a *travel time function* (TTF) that returns the time needed to cross the edge. A relationship between

Kolovský, F., Ježek, J. and Kolingerová, I.
The e-approximation of the Label Correcting Modification of the Dijkstra's Algorithm.
DOI: 10.5220/0007658200260032
In *Proceedings of the 5th International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM 2019)*, pages 26-32
ISBN: 978-989-758-371-1

the TTF g and the corresponding AF f is defined as $g(t) = f(t) - t$.

It is assumed that every AF f fulfill the FIFO property: $\forall t_1 < t_2 : f(t_1) \leq f(t_2)$ and the departure time t_d must be smaller then the arrival time t_a (the travel time must be positive). AFs are implemented as piecewise linear functions.

In Figure 1a you can see AFs (f_{su}, f_{sv}, f_{ud} and f_{vd}) for every edge in a small example graph with four nodes $V = \{s,u,v,d\}$ and four edges $E = \{(s,u),(s,v),(u,d),(v,d)\}$.

The points of AFs are called *breakpoints*. The number of breakpoints of AF f can be written as $|f|$. The following operation must be defined for two AFs:

- There are two consecutive edges (s,u) and (u,d) with AFs f_{su}, f_{ud}. The operation *combination* $f_{ud} * f_{su} : t \mapsto f_{ud}(f_{su}(t))$ represents AF from s to d. In Figure 1b there are AFs as results of the combination along the paths (s,u,d) (solid red line) and (s,v,d) (solid blue line).
- There are two parallel paths p_1, p_2 from s to d with AFs f^1_{sd}, f^2_{sd}. The operation *minimum* $\min(f^1_{sd}, f^2_{sd}) : t \mapsto \min\{f^1_{sd}, f^2_{sd}\}$ represents the earliest AF from s to d. In Figure 1a $p_1 = (s,u,d)$ and $p_2 = (s,v,d)$. In Figure 1c you can see this earliest AF as a result of the operation minimum (green line).

2.2 Problem Definition

More precisely, TDSP can be defined as minimizing the travel time over the set $P_{s,d}$ of all paths in G from the source node s to the destination node d:

$$f_d = \min\{f_p(t) | p \in P_{s,d}\} \quad (1)$$

where f_d is the function of the earliest arrival time (minimal AF) from s to d and f_p is AF of the path $p \in P_{s,d}$.

This paper deals with *one-to-all* problem. The input data are the graph G, AF f_{uv} for every edge $(u,v) \in G$ and the source node s. The output is the set F of the earliest AFs from the source node s to all other nodes u: $F = \{f_u | u \in V \setminus \{s\}\}$.

2.3 Approximation

In this paper the ε-approximation of AF $f^{\updownarrow}$ is understood as the ε-approximation of TTF $f(t) - \varepsilon g(t) \leq f^{\updownarrow}(t) \leq f(t) + \varepsilon g(t)$. So the ε-approximation of the set F is $F^{\updownarrow} = \{f_u^{\updownarrow} | u \in V \setminus \{s\}\}$.

Let us present some useful theorems about the approximation that were derived for a use in the proposed algorithm.

Theorem 1. *Let $g^{\updownarrow}$ be an ε-approximation of TTF g. Then it holds that*

$$g^{\downarrow} = \frac{g^{\updownarrow}}{1+\varepsilon} \leq g \leq \frac{g^{\updownarrow}}{1-\varepsilon} = g^{\uparrow}$$

Proof. If the function $g^{\updownarrow}$ is substituted by its extreme values $(1-\varepsilon)g$, $(1+\varepsilon)g$, the expression is still valid. □

Theorem 2. *Let $f_u^{\updownarrow}$ be an ε-approximation of AF f_u and $\overline{f}_v^{\updownarrow} = f_{uv} * f_u^{\updownarrow}$. Let $\alpha \in [0, \frac{\overline{g}_v^{\downarrow}}{g_u^{\downarrow}}]$ be the maximum slope of AF f_{uv}, $approx(f,\delta)$ be a function that simplifies the AF f with the maximum absolute error $\delta \geq 0$. Then $f_v^{\updownarrow} = approx(\overline{f}_v^{\updownarrow}, \varepsilon\overline{g}_v^{\downarrow} - \alpha\varepsilon g_u^{\downarrow})$ is the ε-approximation of AF f_v.*

Proof. The maximum absolute error of f_v is εg_v and the maximum absolute error of the operation $f_{uv} * f_u^{\updownarrow}$ is $\alpha\varepsilon g_u$. Then the result of the combination can be simplified with the maximum absolute error:

$$\delta = \varepsilon g_v - \alpha\varepsilon g_u \geq \varepsilon\overline{g}_v^{\downarrow} - \alpha\varepsilon g_u^{\downarrow}$$

Then δ must be ≥ 0

$$\varepsilon\overline{g}_v^{\downarrow} - \alpha\varepsilon g_u^{\downarrow} \geq 0$$

$$\alpha \geq \frac{\overline{g}_v^{\downarrow}}{g_u^{\downarrow}}$$

□

Theorem 2 can be also formulated in a local form for a given departure time.

In Figure 1b there are dotted lines that represent the approximation of AFs. In Figure 1d you can see the ε-approximation $f_d^{\updownarrow}$ of the earliest AF f_d from s to d (solid green line) and its upper bound $f_d^{\uparrow}$ and lower bound $f_d^{\downarrow}$ (green dashed lines).

2.4 Related Work

There are two groups of methods that compute an approximation of the AF. The first methods use *forward* and *backward probes*. The forward probe computes the arrival time at the node d with the given departure time at the node s. The backward probe solves the inverse problem. The arrival time at d is given and we want to know the departure time at s. These probes can be computed using the well-known Dijkstra's algorithm (Dehne et al., 2012).

These methods recognize two types of breakpoints. The **V** points represent points that are created as images of the breakpoints that lie on the edge arrival

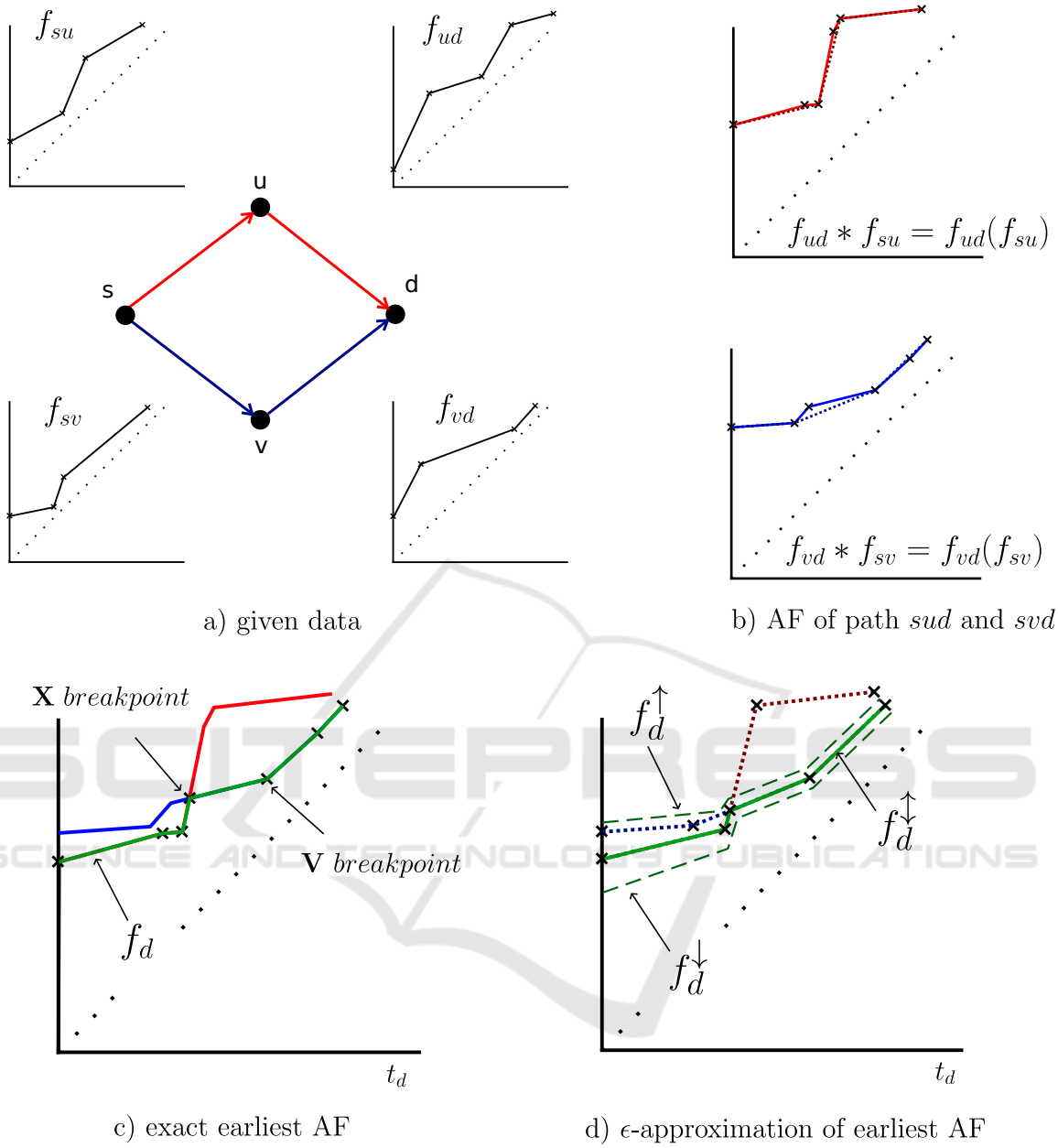

Figure 1: Example of calculation of the arrival function from node s to d.

functions $\{f_{uv}|(u,v) \in E\}$. The **X** points are created as an intersection of two AF in the *minimum* operation. It can be proved that the AF between two consecutive **V** points is concave or a line segment (Foschini et al., 2014) (see Example in Fig. 1c). The algorithms described in (Foschini et al., 2014), (Omran and Sack, 2014) use this concavity. First the **V** points are computed using one backward probe and two forward probes (more in (Dehne et al., 2012)) and then the approximation of AF between the **V** points is determined. The main problem of this approach is that the computation of **V** points requires $3\sum_{(u,v)\in E}|f_{uv}|$ probes (Dehne et al., 2012).

The second group of methods uses a label correcting modification of the Dijkstra's algorithm (LCA) (Algorithm 1). The modifications of the Dijkstra's algorithm are:

- The node labels are AFs from s.
- The key of the priority queue is the minimum of AF ($\min f$).
- The relaxation of the edge (u,v) is performed using

$f_v = \min(f_v, f_{uv} * f_u)$

LCA has time complexity $O(|V||E|)$ (Orda and Rom, 1990), but a real road network is far from the worst case. This technique is widely used, see e.g, (Geisberger and Sanders, 2010) (Batz et al., 2013) (Geisberger, 2010). First LCA is performed in the exact form and after that the resulting arrival functions F are simplified and used for further computation (e.g, some query algorithm). Some guarantees about the error of AF are presented in (Geisberger and Sanders, 2010), but these guarantees give only a maximum error dependent on the degree of approximation.

Algorithm 1: LCA in the exact form.

```
1  PQ = minimum priority queue where key is min f
2  ∀u ∈ V : f_u = ∞
3  g_s = 0
4  PQ.put(f_s)
5  while queue is not empty do
6      f_u = PQ.get()
7      foreach v : (u,v) ∈ E do
8          f̄_v = f_uv * f_u          // combination
9          if ∃t : f̄_v(t) < f_v(t) then // compare
10             f_v = min(f̄_v, f_v)        // min
11             PQ.put(f_v)
```

In Algorithm 1 the initialization is performed in the lines 1-4. All node labels (AFs) are set to infinity in the line 2. The travel time at s is set to zero (line 3) and the node label at s (f_s) is added to the priority queue (PQ) (line 4). In the line 6 the algorithm takes the node on the top of PQ and relaxes all edges that lead from this node. The relaxation is represented by the lines 8-11. The line 8 performs the combination of the node label at u (f_u) and the edge AF (f_{uv}). The condition in the line 9 checks for update. Updates of the label at the node v are performed in the line 10. The line 11 puts the node v to the PQ.

The main task is to develop an algorithm which solves TDSP with the given maximum relative error and is effective for a real road network. It follows that we focused on the ε-approximation of the LCA.

3 PROPOSED ALGORITHMS

This section describes two algorithms solving the ε-approximation of TDSP based on LCA (Algorithm 1).

3.1 ε-LCA Algorithm

The basic idea of the first proposed algorithm (ε-LCA) is that the simplification of AF is performed after every edge relaxation (the operation combination). The degree of the simplification is directed by Theorem 2.

The ε-LCA computes AF with the maximum relative error ε assuming that

$$\forall (u,v) \in E : \alpha_{uv} \in \left[0, \frac{\overline{g_v}^{\downarrow}}{g_u^{\downarrow}}\right] \tag{2}$$

where α_{uv} is the maximum slope of AF f_{uv}. The slope α_{uv} must be bounded because Theorem 2 is used in the ε-LCA and the theorem needs this assumption.

The ε-LCA differs from the exact LCA only in the computation of $\overline{f_v}$. The AF $\overline{f_v}$ in the line 8 in Algorithm 1 is simplified with the maximum absolute error δ (according to Theorem 2). So the line 8 is replaced by 2 lines

$$\begin{aligned} \delta(t) &= \varepsilon((f_{uv} * f_u^{\updownarrow})^{\downarrow}(t) - t) - \alpha(t)\varepsilon g_u^{\downarrow}(t) \\ \overline{f_v} &= approx((f_{uv} * f_u^{\updownarrow}), \delta) \end{aligned} \tag{3}$$

where $\alpha(t)$ is the maximum slope of f_{uv} in the interval $[f_u^{\downarrow}(t), f_u^{\uparrow}(t)]$. The simplification was performed using Douglas-Peucker algorithm or Imai and Iri algorithm (Imai and Iri, 1986).

The main problem of ε-LCA is that if the assumption (2) is not complied, $\delta < 0$ and the algorithm cannot ensure the given relative error ε. This occurs when the maximum slope α_{uv} is too large. This issue is resolved using the second algorithm that is described in the upcoming section.

3.2 ε-LCA-BS Algorithm

The second proposed algorithm (ε-LCA-BS) is based on backsearch. It has no limitations for the slope α. The pseudo-code of the ε-LCA-BS is in Algorithm 2. The basic idea is that if the algorithm finds an edge (u,v) where α is too big in some departure time interval $[t_m, t_n]$ ($\delta < 0$), it determines f_v in $[t_m, t_n]$ again with a higher accuracy (lines 11-15 of the Algorithm 2). The algorithm returns back to the point such that the edge (u,v) can be reached from this point with sufficient precision using the exact LCA.

When the label at the node v (AF f_v) is updated (the condition at the line 16 is fulfilled), the edge (u,v) is added to the predecessor list $pred(v)$ of the node v (lines 17-20). The set of predecessors form a graph $R = (V_R, E_R)$ (red color in Fig. 2). We assume that the graph R is acyclic. In general, the graph R may not be acyclic, but in real case it is very unlikely.

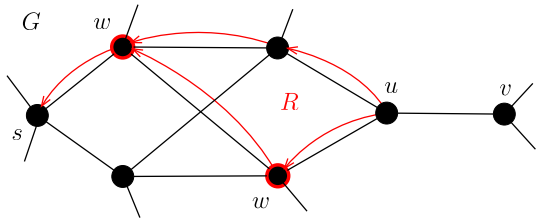

Figure 2: Example of backsearch procedure.

We want to find nodes w such that if the exact LCA is performed from these nodes w, the AF f_v is an ε-approximation. The nodes w have to satisfy the following inequalities in the interval $[t_m, t_n]$:

$$\max\left\{\prod_{e\in p}\alpha_e | p \in P_{vw}\right\} \leq \min\left(\frac{\overline{g}_v^{\downarrow}}{g_w^{\downarrow}}\right) \qquad (4)$$

Algorithm 2: ε-LCA-BS.

1 PQ = *minimum priority queue where key is* $\min f$
2 $\forall u \in V : f_u^{\updownarrow} = \infty$
3 $g_s^{\updownarrow} = 0$
4 PQ.put($f_s^{\updownarrow}$)
5 pred(s) = null
6 **while** *queue is not empty* **do**
7 $f_u^{\updownarrow}$ = PQ.get()
8 **foreach** $v : (u,v) \in E$ **do**
9 $\delta(t) =$ $\varepsilon((f_{uv} * f_u^{\updownarrow})^{\downarrow}(t) - t) - \alpha(t)\varepsilon g_u^{\downarrow}(t)$ `// according equation 3`
10 $\overline{f}_v = approx(f_{uv} * f_u^{\updownarrow}, \delta^+)$ `//` $\delta^+(t) = \max(\delta(t), 0)$
11 **if** $\exists t : \delta(t) < 0$ **then**
12 *find all intervals* $[t_m, t_n]$ *where* δ *is negative*
13 **foreach** $[t_m, t_n]$ **do**
14 $h = backSearch((u,v), [t_m, t_n])$
15 *substitute* $\overline{f}_v$ *by* h *in interval* $[t_m, t_n]$
16 **if** $\exists t : \overline{f}_v(t) < f_v^{\updownarrow}(t)$ **then**
17 **if** $\overline{f}_v < f_v^{\updownarrow}$ **then**
18 pred(v) = (u,v)
19 **else**
20 pred(v).add((u,v))
21 $f_v^{\updownarrow} = \min(\overline{f}_v, f_v^{\updownarrow})$
22 PQ.put($f_v^{\updownarrow}$)

where P_{vw} is the set of all paths from v to w in the graph R (the paths must contain the edge (u,v)) and α_e represents the maximum slope of AF f_e corresponding with the edge e. The set W is the set of all nodes w that meet the condition (4) and there is a path $p \in P_{vw}$ that does not contain any other node from the set W (W is the smallest possible).

These nodes w can be found using a topological ordering of V_R (Algorithm 3). The node labels α_b correspond to the left side of the inequalities (4). So the algorithm finds maximal paths in R. First the labels are set to negative infinity (the line 2) and the label at u is set to α_{uv} (the line 3). The lines 4-5 ensure a topological ordering. If the condition (4) in the line 6 is fulfilled, b is added to the W. The lines 9-12 ensure updating of the node labels. The part of $\overline{f}_v$ in the interval $[t_m, t_n]$ is substituted by a more accurate result of the exact LCA with the initial priority queue PQ that is created by adding all $f_w \in \{f_w | w \in W\}$ (the lines 13-16).

In Figure 2 there is an example of backsearch. The black color represents the original graph G and the red color represents the acyclic graph R. The edge (u,v) violates the condition 2. Then the algorithm starts *backSearch* procedure and finds the set W (red nodes) using the graph R.

Algorithm 3: backSearch.

1 $W = \{\}$ `// set of all w`
2 $\forall b \in V_R : \alpha_b = -\infty$
3 $\alpha_u = \alpha_{uv}$
4 **while** $\exists b : b \in V_R \setminus W \wedge deg^-(b) = 0$ **do**
 `// topological ordering`
5 b = *some node that meets the conditions above*
6 **if** $\alpha_b \leq \min\left(\frac{\overline{g}_v^{\downarrow}}{g_b^{\downarrow}}\right)$ **then**
7 $W = W \cup \{b\}$
8 **else**
9 **foreach** $a : (b,a) \in R$ **do**
10 **if** $\alpha_b \alpha_{ba} > \alpha_a$ **then**
11 $\alpha_a = \alpha_b \alpha_{ba}$
12 $V_R = V_R \setminus \{b\}$
13 **foreach** $w \in W$ **do**
14 PQ.put(f_w)
15 *run LCA on G with initial priority queue PQ on interval* $[t_m, t_n]$ `// algorithm 1`
16 **return** f_v

When the graph R is not acyclic, it is necessary to modify the algorithm for searching the set W.

If the condition (2) is fulfilled, the ε-LCA-BS is reduced to ε-LCA, because the algorithm then does not perform any *backSearch* procedure.

4 EXPERIMENTS

The real road network with real speed profiles that were computed from GPS tracks was used for testing. This data represent part of Paris in France (Figure 3).

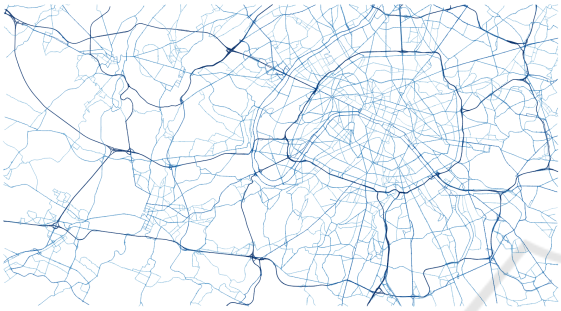

Figure 3: Route network for testing.

The algorithms were implemented using Scala programming language (OpenJDK 1.8, Debian 10). The testing was performed on a computer with Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz and with 16 GB RAM. One thread was used only.

Table 1: Graph properties (#edges - number of edges, $|f_e|$ - number of linear pieces of AF for each edge).

dataset	#edges	$\lvert f_e \rvert$
G1	10 798	24
G2	33 354	24
G3	107 476	24
G4	160 092	24

Table 2: Absolute values of measured parameters for $\varepsilon = 0.001$.

	Imai and Iri		Douglas Peucker	
	time [s]	# bps	time [s]	# bps
G1	0.5	608 k	1.0	940 k
G2	1.9	2 087 k	3.9	3 136 k
G3	6.4	5 734 k	13.3	8 606 k
G4	9.6	7 876 k	20.2	11 892 k

The ε-LCA-BS was tested only because ε-LCA is a special case of ε-LCA-BS only. The maximum allowed relative error ε was set to $10^{-1}, 10^{-2}, 10^{-3}$ and 10^{-4}.

Four graphs (G1, G2, G3 and G4) were created to show the performance of the developed algorithms. Every edge in the graphs has AF with 24 linear pieces.

Table 3: Results of testing ε-LCA-BS (t_r - the relative time related to the exact version of LCA, *bps* - the relative number of breakpoints, ε - the maximum allowed relative error).

		Imai and Iri		Douglas Peucker	
	ε	t_r [%]	*bps* [%]	t_r [%]	*bps* [%]
G1	10^{-1}	3.9	0.1	7.0	0.2
	10^{-2}	6.0	0.8	6.4	0.9
	10^{-3}	16.4	3.1	32.3	4.9
	10^{-4}	45.2	10.6	104.7	15.3
G2	10^{-1}	1.7	0.1	1.5	0.1
	10^{-2}	4.1	0.6	4.7	0.7
	10^{-3}	14.0	3.0	27.1	4.6
	10^{-4}	42.0	10.6	97.3	15.4
G3	10^{-1}	1.5	0.1	1.5	0.1
	10^{-2}	3.5	0.5	3.3	0.5
	10^{-3}	11.1	2.5	21.4	3.8
	10^{-4}	37.5	9.4	86.5	14.2
G4	10^{-1}	1.4	0.1	1.0	0.1
	10^{-2}	3.3	0.5	3.3	0.5
	10^{-3}	10.5	2.4	20.3	3.6
	10^{-4}	35.3	8.8	82.8	13.7

Every graph represents different classes of roads. In Table 1 there are numbers of edges for each graph. In Table 3 there are performance results of ε-LCA-BS: the relative time t_r and the relative number of breakpoints *bps* related to the exact version of LCA. The same results you can see in Figure 4.

The results in Table 3 show that the maximum relative error 10^{-4} brings only a small improvement, but this accuracy is too big for a real use. The maximum relative error from 10^{-2} to 10^{-3} seems to be a good compromise between accuracy and performance.

In Table 2 there are absolute values of measured parameters for the maximal relative error 10^{-3}. The column *bps* represents the number of breakpoints in the resulting AFs. In all cases the graphs (G1-G4) do not violate the condition (2), thus the ε-LCA-BS was reduced to ε-LCA.

The results show that the breakpoints savings are significant. It means that the ε-LCA-BS saves a lot of memory. Let us assume a path that takes 1 hour, then the relative error 0.1 % implicates the absolute error 3.6 s. In this case the epsilon approximation saves more than 95% of memory and 80% of time. In case that ε is too small then the algorithm can run slower than the exact version, because the simplification takes too much time.

The main disadvantage of the ε-LCA-BS is that it is sensitive to values of the maximum slope of AFs. If AFs have too big slope then the algorithm performs too many calls of *backSearch* procedure and thereby makes the computation too slow. In practice, the func-

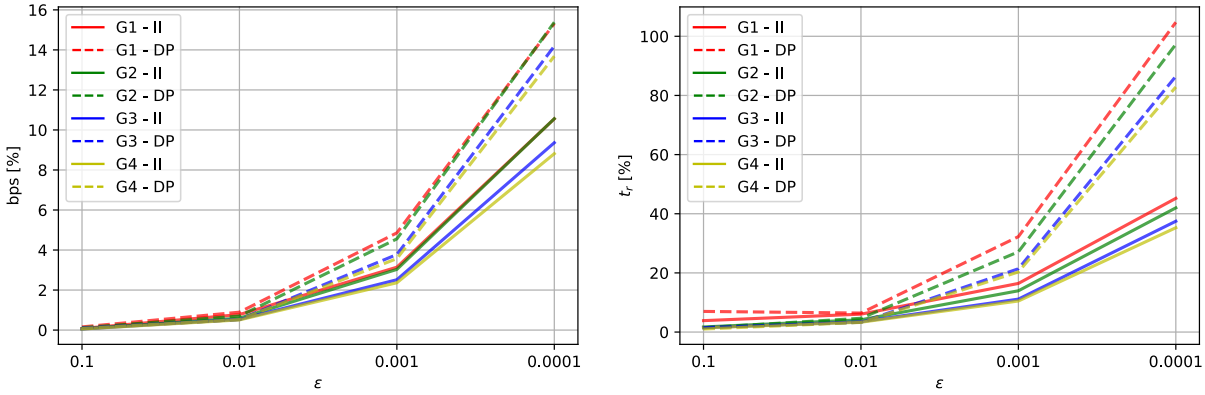

Figure 4: The relative number of breakpoints and the relative time related to exact LCA (II - Imai and Iri, DP - Douglas Peucker).

tions usually have small slopes. When it is certain that input data do not violate the condition (2), the algorithm is more suitable.

5 CONCLUSION

Two algorithms for ε-approximation of TDSP were presented. The algorithms significantly reduce the memory use. When the maximum relative error is a sufficiently large value (in our case 10^{-3}), the algorithms save the computational time too. From this point of view, the algorithms are suitable for precomputing the TTFs for the next use (e.g., time-dependent distance oracles, time-dependent contraction hierarchies).

In a real road network the maximum slopes of AFs are not too big (Strasser, 2017). So the main disadvantage (too many calls of back search procedure) is not a too big problem. In one-to-one problem case the developed algorithms can be combined with other speed-up techniques that reduce the graph (e.g., time-dependent-sampling (Strasser, 2017)).

In the future work it would be useful to use some heuristics for decision whether it is necessary to perform *backSearch*. The goal is to remove the cases when the difficult-to-calculated AF (using *backSearch*) is fully replaced by another AF from another node.

ACKNOWLEDGEMENTS

This work has been supported by the Project SGS-2019-015 (”Využití matematiky a informatiky v geomatice IV”) and by Ministry of Education, Youth and Sports of the Czech Republic, the project PUNTIS (LO1506) under the program NPU I

REFERENCES

Batz, G. V., Geisberger, R., Sanders, P., and Vetter, C. (2013). Minimum time-dependent travel times with contraction hierarchies. *Journal of Experimental Algorithmics*, 18:1.1–1.43.

Dehne, F., Omran, M. T., and Sack, J.-R. (2012). Shortest Paths in Time-Dependent FIFO Networks. *Algorithmica*, 62(1-2):416–435.

Foschini, L., Hershberger, J., and Suri, S. (2014). On the Complexity of Time-Dependent Shortest Paths. *Algorithmica*, 68(4):1075–1097.

Geisberger, R. (2010). Engineering Time-dependent One-To-All Computation. *arXiv:1010.0809 [cs]*. arXiv: 1010.0809.

Geisberger, R. and Sanders, P. (2010). Engineering time-dependent many-to-many shortest paths computation. In *OASIcs-OpenAccess Series in Informatics*, volume 14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Imai, H. and Iri, M. (1986). An optimal algorithm for approximating a piecewise linear function. *Journal of information processing*, 9(3):159–162.

Omran, M. and Sack, J.-R. (2014). Improved approximation for time-dependent shortest paths. In *International Computing and Combinatorics Conference*, pages 453–464. Springer.

Orda, A. and Rom, R. (1990). Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625.

Strasser, B. (2017). Dynamic Time-Dependent Routing in Road Networks Through Sampling. In *OASIcs-OpenAccess Series in Informatics*, volume 59. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.