Computing and Informatics, Vol. 31, 2012, 531–549

GENERALIZED MANEUVERS IN ROUTE PLANNING FOR COMPUTING AND INFORMATICS

Petr Hliněný, Ondrej Moriš

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
e-mail: {hlineny, xmoris}@fi.muni.cz

Abstract. We study an important practical aspect of the route planning problem in real-world road networks – *maneuvers*. Informally, maneuvers represent various irregularities of the road network graph such as turn-prohibitions, traffic light delays, round-abouts, forbidden passages and so on. We propose a generalized model which can handle arbitrarily complex (and even negative) maneuvers and extend traditional Dijkstra's Algorithm in order to solve route planning queries in this model without prior adjustments of the underlying road network graph. Finally, we also briefly evaluate practical performance of our approach (as compared to ordinary Dijkstra on an amplified network graph).

Keywords: Route planning, road network, maneuver, restriction, prohibition

Mathematics Subject Classification 2010: 68, 68R10; 05C85

1 INTRODUCTION

Since mass introduction of GPS navigation devices, the *route planning problem*, has received considerable attention. This problem is in fact an instance of the well-known single pair shortest path (SPSP) problem in graphs representing real-world road networks. However, it involves many challenging difficulties compared to ordinary SPSP. Firstly, classical algorithms such as Dijkstra's [5], A* [7] or their bidirectional variants [12] are not well suited for route planning despite their optimality in wide theoretical sense. It is mainly because graphs representing real-world road networks are so huge that even an algorithm with linear time and space complexity cannot be feasibly run on typical mobile devices.

Secondly, these classical approaches disregard certain important aspects of real-world road networks, namely route restrictions, traffic regulations, or actual traffic info. Hence a route found by such classical algorithms might not be optimal or not even feasible. Additional attributes are needed in this regard.

The first difficulty has been intensively studied in the past, and complexity overheads of classical algorithms have been largely improved by using various preprocessing approaches. Two sorts of advanced techniques have emerged and become popular. The first one prunes the search of Dijkstra's or A* algorithms using preprocessed information. The second sort of techniques exploits a road network structure with levels of hierarchy. There are also many other techniques and combinations,and we refer a reader to Cherkassky et al. [2], Delling et al. [3, 4], and Schultes [13].

In this paper, we focus on the second mentioned sort of issues – route restrictions, traffic regulations, traffic info, etc. – as it is still receiving significantly less attention in the aforementioned mainstream research.

1.1 Related Work

The common way to model required additional attributes of road networks is with so-called *maneuvers* (see Definition 1). Unfortunately, maneuvers do not seem to be in the center of interest of route planning research papers: they are often assumed to be encoded into the underlying graph of a road network, or they are addressed only partially with rather simple types of restriction attributes such as turn-penalties and path prohibitions, or they are considered from a very specific point of view.

Basically, there are two research directions for modeling maneuvers – either maneuvers are encoded into the underlying road network graph using artificial vertices and edges, or a query algorithm is adjusted in order to handle maneuvers during queries. The first research direction, seemingly simplest and commonly used, can be applied without respect to used query algorithm since it makes a road network graph maneuver-free and therefore there is no need to adjust the queries in any way [9, 11, 16, 17, 15], and there is not much to improve nowadays. The problem is that it can significantly increase the size of the resulting graph [15]. For instance, replacing a single turn-prohibition can add up to eight new vertices in place of one original [6]. A solution like this one thus conflicts with the aforementioned graph-size issue. Another noticeable approach [1] uses so-called dual graph representation instead of the original one, where allowed turns are modeled by dual edges. We remark that although this solution works much better for prohibited turns modeling, it is impossible to model the other types of restrictions, for instance traffic lights delays, in dual graphs. On the other hand, the second research direction – adjustment of a query algorithm – is not used very often and there is a lot of space for possible improvements. Although there are a few known approaches [10, 14, 18], they solve only simple types of maneuvers such as, for instance, turn penalties and turn prohibitions [10, 6] or forbidden paths [14, 18].

To summarize, a sufficiently general approach for arbitrarily complex maneuvers seems to be missing in the literature despite its practical importance.

1.2 Our Contribution

Firstly, we introduce a formal model of a generic maneuver – from a single vertex to a long self-intersecting walk – with either positive or negative effects (penalties); being enforced, recommended, not recommended or even prohibited. Our model can capture virtually any route restriction, most traffic regulations and even some dynamic properties of real-world road networks.

Secondly, we integrate this model into Dijkstra's algorithm, rising its worst-case time complexity only slightly (depending on a structure of maneuvers). The underlying graph is not modified at all and no preprocessing is needed. Even though our idea is fairly simple and relatively easy to understand, it is novel in the respect that no comparable solution has been published to date. Furthermore, some important added benefits of our algorithm are as follows:

- It can be directly used bidirectionally with any alternation strategy using an appropriate termination condition; it can be extended also to the A* algorithm by applying a "potential function to maneuver effects".
- Many route planning approaches use Dijkstra or A* in the core of their query algorithms, and hence our solution can be incorporated into many of them (for example, those based on a reach, landmarks or various types of separators) quite naturally under additional assumptions.
- Our algorithm tackles maneuvers "on-line" – that is no maneuver is processed before it is reached. And since the underlying graph of a road network is not changed (no vertices or edges are removed or added), it is possible to add or remove maneuvers dynamically even during queries to some extent.

2 MANEUVERS: BASIC TERMS

A *(directed) graph* $G = (V, E)$ is a pair of a finite set V of vertices and a finite multiset $E \subseteq V \times V$ of edges (self-loops and parallel edges are allowed). The vertex set of G is referred to as $V(G)$, its edge multiset as $E(G)$. *A subgraph* H of a graph G is denoted by $H \subseteq G$.

A walk $P \in G$ is an alternating sequence of vertices and edges $(u_0, e_1, u_1, \ldots, e_k, u_k) \subseteq G$ such that $e_i = (u_{i-1}, u_i)$ for $i = 1, \ldots, k$, the multiset of all edges of a walk P is denoted by $E(P)$. A *subwalk* $Q \subseteq P$ is $Q = (u_i, e_{i+1}, \ldots, e_j, u_j)$ for any $0 \leq i \leq j \leq k$. *A concatenation* $P_1.P_2$ of walks $P_1 = (u_0, e_1, u_1, \ldots, e_k, u_k)$ and $P_2 = (u_k, e_{k+1}, u_{k+1}, \ldots, e_l, u_l)$ is the walk $(u_0, e_1, u_1, \ldots, e_k, u_k, e_{k+1}, \ldots, e_l, u_l)$. If $P_2 = (u, f, v)$ represents a single edge, we write $P_1.f$. If edges are clear from the graph, then we write a walk simply as $(u_0, u_1, \ldots, u_k)$.

Informally, a walk $Q = (u_0, e_1, \ldots, e_k, u_k)$ is a *prefix* of another walk $P = (u_0, e_1, \ldots, e_k, u_k, \ldots, e_l, u_l)$ if Q is a subwalk of P starting with the same sequence of vertices and edges, and analogically with *suffix*. The *prefix set* of a walk $P = (u_0, e_1, \ldots, e_k, u_k)$ is $Prefix(P) = \{(u_0, e_1, \ldots, e_i, u_i) |\ 0 \leq i \leq k\}$, and analogically

$\mathit{Suffix}(P) = \{(u_i, e_{i+1}, \ldots, e_k, u_k) |\ 0 \le i \le k\}$. So, a prefix (suffix) of a walk P is defined as a member of $\mathit{Prefix}(P)$ ($\mathit{Suffix}(P)$), and it is *nontrivial* if $i \ge 1$ ($i < k$, resp.).

The weight of a walk $P \subseteq G$ with respect to a weighting $w : E(G) \mapsto \mathbb{R}$ of G is defined as $\sum_{e \in E(P)} w(e)$ and denoted by $|P|_w$. *A distance* from u to v in G, $\delta_w(u, v)$ is the minimum weight of a walk $P = (u, \ldots, v) \subseteq G$ over all such walks and P is then called *optimal* (with respect to weighting w). If there is no such walk then $\delta_w(u, v) = \infty$. *A path* is a walk without repeating vertices and edges. However, we remark in advance that in the presence of maneuvers an optimal walk need not be a path (Figure 1).

Virtually any route restriction or traffic regulation in a road network, such as turn-prohibitions, traffic lights delays, forbidden passages, turn-out lanes, suggested directions or car accidents by contrast, can be modeled by *maneuvers* – walks having extra (either positive or negative) "cost effects". Formally:

Definition 1 (Maneuvers)**.** *A maneuver* M of G is a walk in G that is assigned a penalty $\Delta(M) \in \mathbb{R} \cup \infty$. A set of all maneuvers of G is denoted by $\mathcal{M}$.

Remark 1. A maneuver with a negative or positive penalty is called *negative* or *positive*, respectively. Furthermore, there are two special kinds of maneuvers – the *restricted* ones of penalty 0 and the *prohibited* ones of penalty ∞.

Obviously, if a route contains some maneuvers, its cost is influenced by their penalties. This cost effect of a maneuver is formalized as follows:

Definition 2 (Penalized weight)**.** Let G be a graph with a weighting w and a set of maneuvers $\mathcal{M}$. The *penalized weight* of a walk $P \subseteq G$ containing the maneuvers $M_1, \ldots, M_r \in \mathcal{M}$ as subwalks is defined as $|P|_w^{\mathcal{M}} = |P|_w + \sum_{i=1}^{r} \Delta(M_i)$.

In Remark 1 we are mentioning several different types of maneuvers depending on their penalty. The intended meaning of these types in route planning is as follows.

- If a driver enters a restricted maneuver, s/he must pass it completely (cf. Definition 3); s/he must obey the given direction(s) regardless of the cost effect. Examples are headings to be followed or specific round-abouts.
- By contrast, if a driver enters a prohibited maneuver, s/he must not pass it completely. S/he must get off it before reaching its end, otherwise it makes his/her route infinitely bad. Examples are forbidden passages or temporal closures.
- Finally, if a driver enters a positive or negative maneuver, s/he is not required to pass it completely; but if s/he does, then this will increase or decrease the cost of his/her route accordingly. Negative maneuvers make his/her route better (more desirable) and positive ones make it worse. Examples of positive maneuvers are, for instance, traffic lights delays, lane changes, or left-turns. Examples of negative ones are turn-out lanes, shortcuts, or implicit routes.

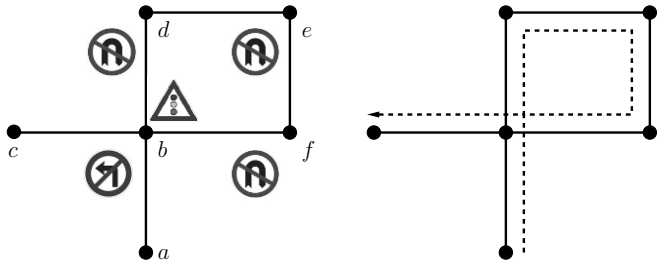
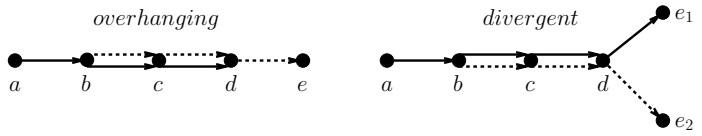


Fig. 1. A road network containing five maneuvers: $M_1 = (a, ab, b, bc, c)$ with $\Delta(M_1) = \infty$ (prohibited left turn), $M_2 = (a, ab, b, bf, f)$ with $\Delta(M_2) = 1$ (right turn traffic lights delay), $M_3 = (b, bd, d, db, b)$, $M_4 = (d, de, e, ed, d)$ and $M_5 = (e, ef, f, fe, e)$ with $\Delta(M_3) = \Delta(M_4) = \Delta(M_5) = \infty$ (forbidden U-turns). All edges have weight 1. The penalized weight of the walk (a, ab, b, bc, c) is $2 + \infty$, the penalized weight of the walk $(a, ab, b, bf, f, fe, e, ed, d, db, b, bd, c)$ is $6+1$. Therefore the optimal walk (with respect to the penalized weight) from a to c is $(a, ab, b, bd, d, de, e, ef, f, fb, b, bc, c)$ with the penalized weight $6 + 0$.

Consider two walks Q_1, Q_2. We say that Q_2 *overhangs* Q_1 if a nontrivial prefix of Q_2 is a suffix of Q_1 (particularly, $E(Q_1) \cap E(Q_2) \neq \emptyset$). Furthermore, Q_1, Q_2 are *divergent* if, up to symmetry between Q_1, Q_2, a nontrivial prefix of Q_2 is contained in Q_1 but the whole Q_2 is not a subwalk of Q_1 and Q_2 does not overhang Q_1.



Definition 3 (Valid walks)**.** Let $G, w, \mathcal{M}$ be as in Definition 2. A walk P in G is *valid* if and only if $|P|_w^{\mathcal{M}} < \infty$ and, for any restricted maneuver $M \in \mathcal{M}$, it holds that if a nontrivial prefix of M is a subwalk of P, then whole M is a subwalk of P or M overhangs P (in other words, M is not divergent with P).

We finally get to the summarizing definition. A structure of a road network is naturally represented by a graph G such that the junctions are represented by $V(G)$ and the roads by $E(G)$. The chosen cost function (for example travel time, distance, expenses) is represented by a *non-negative* weighting $w : E(G) \mapsto \mathbb{R}_0^+$ assigned to G, and the additional attributes such as traffic regulations are represented by maneuvers as above. The least natural requirements on a reasonable road network are then contained in the next definition:

Definition 4. Let G be a graph with a non-negative weighting w and $\mathcal{M}$ a set of maneuvers in G. Then *a road network* is defined as the triple $(G, w, \mathcal{M})$. Furthermore, it is called *proper* if the following three conditions are satisfied:

i. no two restricted maneuvers in $\mathcal{M}$ are divergent,

ii. no two negative maneuvers in $\mathcal{M}$ overhang one another, and

iii. for all $N \in \mathcal{M}$, $\Delta(N) \geq -|N|_w^{\mathcal{M}\setminus\{N\}}$ (that is, for every walk P in G, $|P|_w^{\mathcal{M}} \geq 0$).

Motivation for the required properties i.–iii. in Definition 4 is of both natural and practical character: as for i., it simply says that no two restricted maneuvers are in a conflict (that is no route planning deadlocks). Point ii. concerning only non-overhanging negative maneuvers is needed for a fast query algorithm, and it is indeed a natural requirement (to certain extent, overhanging maneuvers can be modeled without overhangs). We remark that other studies usually allow no negative maneuvers at all. Finally, iii. states that no negative maneuvers can result in a negative overall cost of any walk – another very natural property. In informal words, a negative penalty of a maneuver somehow "cannot influence" suitability of a route before entering and after exiting the maneuver.

Within a road network, only valid walks (Definition 3) are allowed further on, and the *distance* from u to v in a road network $(G, w, \mathcal{M})$, denoted by $\delta_w^{\mathcal{M}}(u, v)$, is the minimum penalized weight (Definition 2) of a valid walk $P = (u, \ldots, v) \subseteq G$. Such a walk P is then called *optimal with respect to the penalized weight*. If there is no such walk, then $\delta_w^{\mathcal{M}}(u, v) = \infty$ (see Figure 1).

2.1 Strongly Connected Road Network

The traditional graph theoretical notion of strong connectivity also needs to be refined – it must suit our road networks to dismiss possible route planning traps now imposed by maneuvers.

First, we need to define a notion of a *"context"* of a vertex (position) v in a road network; capturing all maneuvers one has started, but not yet finished, before reaching v. Precisely, it is a maximal walk in G ending at v such that it is a proper prefix of a maneuver in $\mathcal{M}$, or $\emptyset$ otherwise. The set of all such walks for the vertex v is denoted by $\mathcal{X}_{\mathcal{M}}(v)$. For example, on the road network depicted in Figure 1, $\mathcal{X}_{\mathcal{M}}(b) = \{(a, b), \emptyset\}$. More formally:

Definition 5 (Context)**.** Let $\mathcal{M}$ be a set of maneuvers over a graph G. We define

$$\mathcal{X}_{\mathcal{M}}(v) \stackrel{\text{def}}{=} \big\{X \in \mathit{Prefix}^{<}(\mathcal{M}) \,|(v) \in \mathit{Suffix}(X)\big\} \cup \{\emptyset\}, \text{ where}$$

$$\mathit{Prefix}^{<}(M) \stackrel{\text{def}}{=} \mathit{Prefix}(M) \setminus \{M\}, \quad \mathit{Prefix}^{<}(\mathcal{M}) \stackrel{\text{def}}{=} \bigcup\nolimits_{M\in\mathcal{M}} \mathit{Prefix}^{<}(M).$$

This $\mathcal{X}_{\mathcal{M}}(v)$ is the *maneuver-prefix set* at v, that is the set of all proper prefixes of walks from $\mathcal{M}$ that end right at v, including the mandatory empty walk. An element of $\mathcal{X}_{\mathcal{M}}(v)$ is called a *context* of the position v within the road network.

The reverse graph G^R of G is a graph on the same set of vertices with all of the edges reversed. Let $(G, w, \mathcal{M})$ be a road network, a *reverse road network* is defined as $(G^R, w^R, \mathcal{M}^R)$, where $w^R : E(G^R) \mapsto \mathbb{R}_0^+$, $\forall (u, v) \in E(G^R) : w^R(u, v) = w(v, u)$

and $\mathcal{M}^R = \{M^R | M \in \mathcal{M}\}$, $\forall M^R \in \mathcal{M}^R : \Delta(M^R) = \Delta(M)$. Traditional graph connectivity is then extended to our road networks as follows:

Definition 6. A road network $(G, w, \mathcal{M})$ is *strongly connected* if, for every pair of edges $e = (u', u)$, $f = (v, v') \in E(G)$ and for each possible context $X = X_1 \cdot e \in \mathcal{X}_{\mathcal{M}}(u)$ of u in G and each one of v in G^R, that is $Y^R = Y_1^R . f^R \in \mathcal{X}_{\mathcal{M}^R}(v)$, there exists a valid walk starting with X and ending with Y.

We remark that Definition 6 naturally corresponds to strong connectivity in an amplified road network modeling the maneuvers within underlying graph.

3 ROUTE PLANNING QUERIES WITH MANEUVERS

At first, let us recall classical Dijkstra's algorithm [5]. It solves the SPSP[1] problem, a graph G with a non-negative weighting w for a pair $s, t \in V(G)$ of vertices.

- The algorithm maintains, for all $v \in V(G)$, a *(temporary) distance estimate* of the shortest path from s to v found so far in $d[v]$, and a predecessor of v on that path in $\pi[v]$.
- The scanned vertices, that is those with $d[v] = \delta_w(s, v)$, are stored in the set T; and the reached but not yet scanned vertices, that is those with $\infty > d[v] \geq \delta_w(s, v)$, are stored in the set Q.
- The algorithm work as follows: it iteratively picks a vertex $u \in Q$ with minimum value $d[u]$ and relaxes all the edges (u, v) leaving u. Then u is removed from Q and added to T. *Relaxing* an edge (u, v) means to check if a shortest path estimate from s to v may be improved via u; if so, then $d[v]$ and $\pi[v]$ are updated. Such v is added into Q if is not there already.
- The algorithm terminates when t is scanned or when Q is empty.

Time complexity depends on the implementation of Q; such as it is $\mathcal{O}(|E(G)| + |V(G)| \log |V(G)|)$ with the Fibonacci heap.

3.1 $\mathcal{M}$-Dijkstra's Algorithm: Overview

In this section, we will briefly sketch the core ideas of our natural extension of Dijkstra's algorithm. We refer a reader to Algorithm 1 for a full-scale pseudocode of this $\mathcal{M}$-Dijkstra's algorithm.

1. Every vertex $v \in V(G)$ scanned during the algorithm, is considered together with its context $X \in \mathcal{X}_{\mathcal{M}}(v)$ (Definition 5); that is as a pair (v, X). The intention is for X to record how v has been reached in the algorithm, and the same v can obviously be reached and scanned more than once, with different contexts. For

[1] Single Pair Shortest Path: Given a graph and two vertices, find a shortest path from one to another.

instance, b can be reached with the empty or (a, b) contexts in the road network depicted in Figure 1.

2. Temporary distance estimates are stored in the algorithm as $d[v, X]$ for such vertex-context pairs (v, X). At each step the algorithm selects a next pair (u, Y) such that it is minimal with respect to the following partial order $\leq_{\mathcal{M}}$.

   Remark 2. Partial order $\leq_{\mathcal{M}}$:

   $$(v_1, X_1) \leq_{\mathcal{M}} (v_2, X_2) \overset{\text{def}}{\Longleftrightarrow} \begin{array}{ll} \big(d[v_1, X_1] < d[v_2, X_2] & \vee \\ (d[v_1, X_1] = d[v_2, X_2] & \wedge\, X_1 \in \mathit{Suffix}(X_2)\,)\big). \end{array}$$

3. Edge relaxation from a selected vertex-context pair (u, Y) respects all maneuvers related to the context Y (there can be more such maneuvers). If one of them is restricted, then only its unique (cf. Definition 4, i.) subsequent edge is taken, cf. Algorithm 1, RestrictedDirection. Otherwise, every edge $f = (u, v)$ is relaxed such that the distance estimate at v – together with its context as derived from the concatenation $(Y.\, f)$ – is (possibly) updated with the weight $w(f)$ plus the sum of penalties of all the maneuvers in $(Y.\, f)$ ending at v, cf. Algorithm 1, Relax.
4. If an edge being relaxed is the first one of a negative maneuver, a specific process is executed before scanning the next vertex-context pair. See below.

3.2 Processing Negative Maneuvers

Note that the presence of a maneuver of negative penalty *may violate* the basic assumption of ordinary Dijkstra's algorithm; that relaxing an edge never decreases the nearest temporary distance estimate in the graph. An example of such a violation can be seen in Figure 2, for instance, at vertex v_5 which would not be processed in its correct place by ordinary Dijkstra's algorithm. That is why a negative maneuver M must be processed by $\mathcal{M}$-Dijkstra's algorithm at once – whenever its starting edge is relaxed, cf. Algorithm 1, ProcessNegative.

Suppose that an edge $f = (u, v)$ is relaxed from a selected vertex-context pair (u, X) and there is a negative maneuver $M = (v_0, f_1, v_1, \ldots, v_{n-1}, f_n, v_n)$, $u = v_0$, $v = v_1$ starting with f (that is $f = f_1$), processing M works as follow:

1. Vertex-context pairs $(v_i, X_i), 0 \leq i \leq n$ along M are scanned one by one towards the end of M. The other vertices leaving these v_i are ignored.
2. Scanned vertex-context pairs are added to Q and their distance estimates are updated, but none of them is added into T. They must be properly scanned during the main loop of the algorithm.
3. This process terminates when the end (v_n, X_n) is reached or the distance estimate of some (v_i, X_i) bounces to ∞ (that is there is a prohibited maneuver ending at v_i) or when some restricted maneuver forces us to get off M (and thus M cannot be completed).

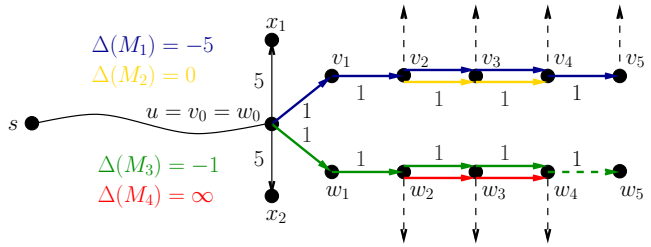


Fig. 2. A road network containing two negative maneuvers, $M_1 = (v_0, \ldots, v_5)$ and $M_3 = (w_0, \ldots, w_5)$, a restricted maneuver $M_2 = (v_2, v_3, v_4)$, and a prohibited maneuver $M_4 = (w_2, w_3, w_4)$. When u is being processed (with its implicit context), x_1, x_2 and v_1, w_1 are relaxed normally. Furthermore, negative maneuver processing is executed for both M_1 and M_3. As a result, v_5 will be immediately reached and inserted to Q with distance estimate equal to that of u which is less than those of x_1, x_2 (5 from u) and of v_1, w_1 (1 from u). On the other hand, w_5 will not be reached in the process because the distance estimate of w_4 bounces to ∞ while handling M_4.

Remark 3. Let us now get back to the condition ii. in Definition 4. Why do we need to forbid negative maneuvers overhanging each other to achieve faster queries algorithm?

Consider negative overhanging maneuvers M_1 and M_2. When ProcessNegative is executed on M_1 (while relaxing its first edge) and it reaches M_2, another ProcessNegative must be started for M_2. The problem occurs when the first edge of M_2 is reached later again, for instance, with a different context or a better distance estimate. In such case ProcessNegative must be executed again for M_2. If there are many long chains of overhanging maneuvers, this might heavily decrease the overall performance of the algorithm.

We believe that this problem can be efficiently solved by taking advantage of a suitable utilization of relative distance estimates in a combination with more advanced queues – pairing heaps [19]. However, further implementation details are out of scope of this paper and they are left for future research. Notice also that any overhanging could be encoded into the road network by introducing auxiliary vertices and edges. We also remark that negative maneuvers overhanging is indeed very rare in practice – recall that negative maneuvers usually represent turn-out lanes, recommended turns or shortcuts. These kinds of maneuvers are usually short in practice and it is almost useless to tie them together. It would be very surprising for us to see such overhanging for more than two maneuvers in practice.

3.3 $\mathcal{M}$-Dijkstra's Algorithm: Pseudocode and Analysis

As sketched in the previous parts, we now present a full formal pseudocode of our variant of Dijkstra's Algorithm respecting maneuvers from a set $\mathcal{M}$; see Algorithm 1.

Algorithm 1 $\mathcal{M}$-Dijkstra's Algorithm

Input: A proper road network $(G, w, \mathcal{M})$ and vertices $s, t \in V(G)$.
Output: A valid walk from s to t in G optimal with respect to the penalized weight.

$\mathcal{M}$-DIJKSTRA$(G, w, \mathcal{M}, s, t)$
1: **for all** $v \in V(G)$, $X \in \mathcal{X}_{\mathcal{M}}(v)$ **do** /* *Initialization.* */
2: $d[v, X] \leftarrow \infty$; $\pi[v, X] \leftarrow \perp$
3: **done**
4: $d[s, \emptyset] \leftarrow 0$; $Q \leftarrow \{(s, \emptyset)\}$; $T \leftarrow \emptyset$
5: **if** $(s) \in \mathcal{M}$ **then** $d[s, \emptyset] \leftarrow d[s, \emptyset] + \Delta(s)$ **fi**

/* *The main loop starts at $(s, \emptyset)$ and terminates when either all reachable vertex-context pairs have been scanned or when t is reached with some of its contexts.* */
6: **while** $Q \neq \emptyset \wedge [\nexists X \in \mathcal{X}_{\mathcal{M}}(t)$ s.t. $(t, X) \in T]$ **do**
7: $(u, X) \leftarrow \min_{\leq_{\mathcal{M}}}(Q)$; $Q \leftarrow Q \setminus \{(u, X)\}$ /* *Recall $\leq_{\mathcal{M}}$ (Remark 2)* */
8: $F \leftarrow$ RESTRICTEDDIRECTION(u, X) /* *Possible restricted dir. from u.* */
9: **if** $F = \emptyset$ **then** $F \leftarrow \{(u, v) \in E(G) \mid v \in V(G)\}$ **fi**
10: **for all** $f = (u, v) \in F$ **do**
11: RELAX(u, X, f, v)
12: **for all** $M = (u, f, v, \ldots) \in \mathcal{M}$ s.t. $\Delta(M) < 0 \wedge |E(M)| > 1$ **do**
13: PROCESSNEGATIVE(X, M)
14: **done** /* *Negative man. starting with f are processed separately.* */
15: **done**
16: $T \leftarrow T \cup \{(u, X)\}$
17: **done**
18: CONSTRUCTWALK(G, d, π) /* *Use "access" information stored in $\pi[v, X]$.* */

RELAX(u, X, f, v) /* *Relaxing an edge f from vertex u with context X.* */
1: $\delta \leftarrow w(f) + \sum_{N \in \mathcal{N}} \Delta(N)$ where $\mathcal{N} = \mathcal{M} \cap \mathit{Suffix}(X.f)$
2: $X' \leftarrow$ LONGESTPREFIX$(X.f)$ /* *see Algorithm 2* */
3: **if** $d[u, X] + \delta < d[v, X']$ **then**
4: $Q \leftarrow Q \cup \{(v, X')\}$; $d[v, X'] \leftarrow d[u, X] + \delta$; $\pi[v, X'] \leftarrow (u, X)$
5: **fi**

PROCESSNEGATIVE$(X, M = (v_0, e_1, \ldots, e_n, v_n))$
1: $i \leftarrow 1$; $X_0 \leftarrow X$; $F \leftarrow \emptyset$ /* *Relaxing sequentially all the edges of M.* */
2: **while** $i \leq n \wedge d[v_{i-1}, X_i] < \infty \wedge F = \emptyset$ **do**
3: RELAX$(v_{i-1}, X_{i-1}, e_i, v_i)$
4: $X_i \leftarrow$ LONGESTPREFIX$(X_{i-1}.e_i)$ /* *see Algorithm 2* */
5: $F \leftarrow$ RESTRICTEDDIRECTION$(v_i, X_i) \setminus \{e_{i+1}\}$
6: $i \leftarrow i + 1$
7: **done**

Algorithm 2 Supplementary routines for Algorithm 1

LongestPrefix (P) : a walk $P' \subseteq G$

/* *The longest (proper) prefix of some maneuver contained as a suffix of* P: */

1: $P' \leftarrow \max_{\subseteq} \left[(\mathit{Suffix}(P) \cap \mathit{Prefix}^{<}(\mathcal{M})) \cup \{\emptyset\} \right]$

2: **return** P'

RestrictedDirection(u, X) : $F \subseteq E(G)$

/* *Looking for edge* f *leaving* u *that follows in a restrict. man. in context* X.*/

1: $F \leftarrow \{f = (u, v) \in E(G) \mid \exists \text{ restricted } R \in \mathcal{M} :$
$E(X) \cap E(R) \neq \emptyset \wedge \mathit{Suffix}(X.\, f) \cap \mathit{Prefix}(R) \neq \emptyset\}$

2: **return** F

After that, it remains to argue about correctness of our algorithm and its time complexity. Assuming validity of crucial Definition 4 ii. in a proper road network, correctness of $\mathcal{M}$-Dijkstra's Algorithm 1 can be argued analogously to a traditional proof of Dijkstra's algorithm. Hereafter, the time complexity overhead of our algorithm depends solely on the number of vertex-context pairs, or better expressed, on the number of maneuvers per vertex.

Theorem 1. Let a proper road network $(G, w, \mathcal{M})$ and vertices $s, t \in V(G)$ be given. $\mathcal{M}$-Dijkstra's algorithm (Algorithm 1) computes a valid walk from s to t in G optimal with respect to the penalized weight, in time $\mathcal{O}\big(c_{\mathcal{M}}^2|E(G)| + c_{\mathcal{M}}|V(G)| \log(c_{\mathcal{M}}|V(G)|)\big)$ where $c_{\mathcal{M}} = \max_{v \in V(G)} |\{M \in \mathcal{M} \mid v \in V(M)\}|$ is the maximum number of maneuvers per vertex.

Proof. We follow a traditional proof of ordinary Dijkstra's algorithm with a simple modification – instead of vertices we consider vertex-context pairs as in Definition 6 and in Algorithm 1.

For a walk P let $\chi(P) = \max_{\subseteq} \left[(\mathit{Suffix}(P) \cap \mathit{Prefix}^{<}(\mathcal{M})) \cup \{\emptyset\} \right]$ denote the context of the endvertex of P with respect to maneuvers $\mathcal{M}$. Let P_x stand for the prefix of P up to a vertex $x \in V(P)$. The following invariant holds at every iteration of the algorithm:

I. For every $(u, X) \in T$, the final distance estimate $d[u, X]$ equals the smallest penalized weight of a valid walk P from s to u such that $X = \chi(P)$. Every vertex-context pair directly accessible from a member of T belongs to Q.

II. For every $(v, X') \in Q$, the temporary distance estimate $d[v, X']$ equals the smallest penalized weight of a walk R from s to v such that $X' = \chi(R)$ and, moreover, $(x, \chi(R_x)) \in T$ for each internal vertex $x \in V(R)$ (except vertices reached during ProcessNegative, if any).

This invariant is trivially true after the initialization. By induction we assume it is true at the beginning of the while loop on line 6, and line 7 is now being executed – selecting the pair $(u, X) \in Q$. Then, by minimality of this selection,

(u, X) is such that the distance estimate $d[u, X]$ gives the optimal penalized weight of a walk P from s to u such that $X = \chi(P)$. Hence the first part I. of the invariant (concerning T, line 16) will be true also after finishing this iteration.

Concerning the second claim II. of the invariant, we have to examine the effect of lines 8–15 of the algorithm. Consider an edge $f = (u, v) \in E(G)$ starting in u, and any walk R from s to v such that $\chi(R_u) = X$. Since $\chi(R)$ must be contained in $X.f$ by definition, it is $\chi(R) = X'$ as in Relax, line 2,. Furthermore, every maneuver contained in R and not in R_u must be a suffix of $X.f$ by definition. So the penalized weight increase δ is correctly computed in Relax, line 1. Therefore, Relax correctly updates the temporary distance estimate $d[v, X']$ for every such f. Finally, any negative maneuver starting from u along f is correctly reached towards its end w on line 13, its distance estimate is updated by successive relaxation of its edges and, by Definition 4, ii. and iii., this distance estimate of w and its context is not smaller than $d[u, X]$; thus the second part of claimed invariant remains true.

Validity of a walk is given by line 8 – RestrictedDirection, that is enforcing entered restricted maneuvers; and line 1 in Relax – δ grows to infinity when completing prohibited maneuvers, "if" condition on line 3 in Relax is then false and therefore prohibited maneuver cannot be contained in an optimal walk.

Lastly, we examine the worst-case time complexity of this algorithm. We assume G is efficiently implemented using neighborhood lists, the maneuvers in $\mathcal{M}$ are directly indexed from all their vertices and their number is polynomial in the graph size (and hence $\log(c_{\mathcal{M}}|V(G)|) = \mathcal{O}\big(\log|V(G)|\big)$), and that Q is implemented as Fibonacci heap.

- The maximal number of vertex-context pairs that may enter Q is

$$m = |V(G)| + \sum_{M \in \mathcal{M}} (|M| - 1) \leq c_{\mathcal{M}} \cdot |V(G)|,$$

  and time complexity of all the Fibonacci heap operations is $\mathcal{O}(m \log m) = \mathcal{O}(c_{\mathcal{M}}|V(G)| \log|V(G)|)$.
- Every edge of G starting in u is relaxed at most those many times as there are contexts in $\mathcal{X}_{\mathcal{M}}(u)$, and edges of negative maneuvers are relaxed one more time during ProcessNegative. Thus the maximal overall number of relaxations is

$$r = \sum_{u \in V(G)} |\mathcal{X}_{\mathcal{M}}(u)| \cdot \mathit{out\text{-}deg}(u) + q \leq (c_{\mathcal{M}} + 1) \cdot |E(G)|$$

  where q is the number of edges belonging to negative maneuvers.
- The operations in Relax on line 1, LongestPrefix as well as RestrictedDirection can be implemented in time $O(c_{\mathcal{M}})$.

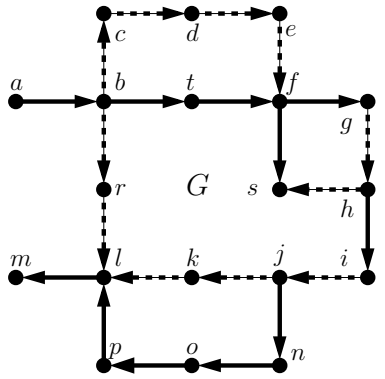
The claimed runtime bound follows. □

Notice that, in real-world road networks, the number $c_{\mathcal{M}}$ of maneuvers per vertex is usually quite small and independent of the road network size, and thus it can be

bounded by a reasonable minor constant. Although road networks in practice may have huge maneuver sets, particular maneuvers do not cross or interlap too much there. For example, $c_{\mathcal{M}} = 5$ in the current OpenStreetMaps of Prague.

3.4 Route Planning Example

In this section, we will demonstrate $\mathcal{M}$-Dijkstra's algorithm on a road network containing maneuvers. Consider the road network depicted below with a weighting representing travel times. There are five illustrative maneuvers (their edges are depicted by dotted lines): a traffic jam detour (M_1), a forbidden passage (M_2), a traffic light left turn delay (M_3), a traffic light delay (M_4) and a direct to be followed (M_5).



Road network $(G, w, \mathcal{M})$, where

- G is depicted on the left,
- $\forall e \in E(G) : w(e) = 1$,
- $\mathcal{M} = \{M_1, M_2, M_3, M_4, M_5\}$

$$\begin{aligned} M_1 &= (b, c, d, e, f) \\ \Delta(M_1) &= -3 \\ M_2 &= (b, r, l) \\ \Delta(M_2) &= \infty \\ M_3 &= (g, h, s) \\ \Delta(M_3) &= 5 \\ M_4 &= (s) \\ \Delta(M_4) &= 9 \\ M_5 &= (i, j, k, l) \\ \Delta(M_5) &= 0 \end{aligned}$$

The goal of out driver is to get from a to m as fast as possible. Classical Dijkstra's algorithm finds $P_1 = (a, b, r, l, m)$ with $|P_1|_w = 4$, but unfortunately $|P_1|_w^{\mathcal{M}} = \infty$ and hence P_1 is unacceptable for our driver – it contains a forbidden passage (M_2). On the other hand, $\mathcal{M}$-Dijkstra's algorithm finds $P_2 = (a, b, c, d, e, f, g, h, i, j, k, l, m)$ with $|P_2|_w^{\mathcal{M}} = 9$ and P_2 is optimal w.r.t. the penalized weight. Its steps are outlined in subsequent Table 1 and Figure 3.

Step	(u, X) *(line 7)*	$d[u, X]$	Q *(line 16)*
1	$[a, \emptyset]$	0	$[b, \emptyset]$
2	$[b, \emptyset]$	1	$[c, (b, c)]$; $[t, \emptyset]$, $[r, (b, r)]$
3	$[c, (b, c)]$	2	$[t, \emptyset]$; $[r, (b, r)]$; $[d, (b, c, d)]$; $[e, (b, c, d, e)]$; $[f, \emptyset]$
4	$[t, \emptyset]$	2	$[(r, (b, r)]$; $[d, (b, c, d)]$; $[e, (b, c, d, e)]$; $[f, \emptyset]$
5	$[r, (b, r)]$	2	$[d, (b, c, d)]$; $[e, (b, c, d, e)]$; $[f, \emptyset]$; $[l, \emptyset]$
6	$[f, \emptyset]$	2	$[g, \emptyset]$; $[s, \emptyset]$; $[d, (b, c, d)]$; $[e, (b, c, d, e)]$; $[l, \emptyset]$
7	$[d, (b, c, d)]$	3	$[g, \emptyset]$; $[s, \emptyset]$; $[e, (b, c, d, e)]$; $[l, \emptyset]$
8	$[g, \emptyset]$	3	$[h, (g, h)]$; $[s, \emptyset]$; $[e, (b, c, d, e)]$; $[l, \emptyset]$
9	$[e, (b, c, d, e)]$	4	$[h, (g, h)]$; $[s, \emptyset]$; $[l, \emptyset]$
10	$[h, (g, h)]$	4	$[i, \emptyset]$; $[s, \emptyset]$; $[l, \emptyset]$
11	$[i, \emptyset]$	5	$[j, (i, j)]$; $[s, \emptyset]$; $[l, \emptyset]$
12	$[j, (i, j)]$	6	$[k, (i, j, k)]$; $[s, \emptyset]$; $[l, \emptyset]$
13	$[k, (i, j, k)]$	7	$[s, \emptyset]$; $[l, \emptyset]$
14	$[l, \emptyset]$	8	$[m, \emptyset]$; $[s, \emptyset]$
15	$[m, \emptyset]$	9	$[s, \emptyset]$

Table 1. State of selected data structures during the steps of Algorithm 1. The second column shows a vertex-context pair chosen at the beginning of the while-loop, i.e. $\min_{\leq_{\mathcal{M}}}(Q)$. The third column shows its final distance estimate, i.e. $d[u, X] = \delta_w^{\mathcal{M}}(a, u)$ and, finally, the last column depicts elements of the queue Q at the end of the while-loop.

3.5 Experimental Results

In this section, we briefly examine the performance of the proposed algorithm. The prototype of the $\mathcal{M}$-Dijkstra's algorithm is written in C. Publicly available road networks are taken from TIGER/Line 2010[2] and from 9th DIMACS Implementation Challenge[3]. We are using directed edges, i.e. every traffic lane is represented by one edge. The compilation was done by gcc 4.5.1 with -O2, and the queries were executed on 1GHz AMD Athlon(tm) 64 X2 Dual Core Processor 4 800+ with 2 GB of memory running GNU/Linux kernel 2.6.35.14. Notice that the data used contain a considerable amount of errors, are planarized and *contain no maneuvers data*. For this reason, we have generated various types of maneuver by ourselves.

[2] `http://www.census.gov/geo/www/tiger/tgrshp2010/tgrshp2010.html`

[3] `http://www.dis.uniroma1.it/~challenge9/`

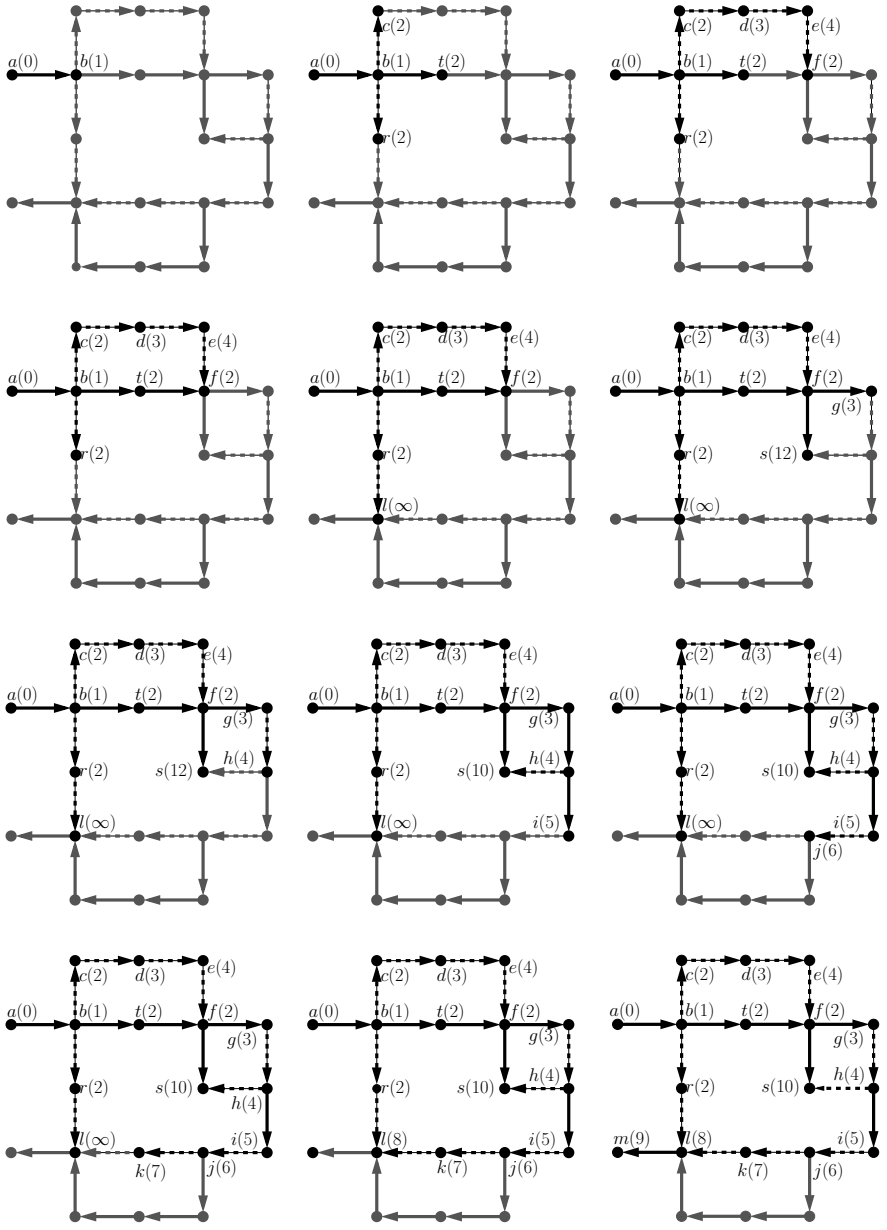


Fig. 3. A computation of an optimal walk with respect to the penalized weight from a to m in G. Numbers represent the distance from the start a. Black vertices are reached or scanned and black edges were relaxed. Dotted edges represent maneuver edges. Step 7 has no figure since it has the same figure as 6, analogously for 8.

First, we did the basic evaluation by comparing our $\mathcal{M}$-Dijkstra's algorithm to the other two modification of Dijkstra's algorithm mentioned in Section 1.1 – one by Kirby and Potts [10] and another by Gutierrez and Medaglia [6]. We were using only turn prohibitions as these two algorithms used in our comparison cannot handle more complex maneuvers. We performed 1 000 random queries and then took the average values. The same basic implementation of Dijkstra's algorithm was used in all three algorithm. *Results were very similar, the average numbers of scanned elements and running times were the same in average.* Moreover, with rather small number of generated turn prohibitions, results were even comparable to the maneuver-free version of Dijkstra's algorithm. All these results were expected because turn prohibitions are the simplest types of maneuvers and cannot influence the performance of evaluated algorithms too much.

Although it would be more interesting to compare our algorithm with different approaches on road network with more complex route restrictions, there are no comparable algorithms unfortunately. Therefore we compared our algorithm to ordinary maneuver-free Dijkstra's algorithm executed on a road network with already encoded maneuvers. Results of this basic evaluation indicate a very good performance of our algorithm.

Algorithm	*Road Network Size*	*Scanned Elements*	*Time (ms)*
MD	651 543/1 710 124	321 287	2 776
D*	1 117 114/2 256 870	527 385	3 939

Table 2. A comparison of our $\mathcal{M}$-Dijkstra's algorithm (MD) with the classical maneuver-free Dijkstra's algorithm (D*) on the same road network but with all maneuvers encoded into it. There are 50 000 maneuvers generated with average length 5, out of them a quarter are negative maneuvers. Overhanging on more than one edge was avoided. We randomly chose 1 000 pairs of vertices (of the original road network) and computed optimal walks with respect to the (penalized) weight. Below we list average values for numbers of scanned vertices and running times of algorithms. We also remark that encoding maneuvers into the road network was algorithmically non-trivial as well as time consuming (approx. 27 minutes).

4 CONCLUSION

We have introduced a novel generic model of maneuvers that is able to capture almost arbitrarily complex route restrictions, traffic regulations and even some dynamic aspects of the route planning problem. It can model anything from single vertices to long self-intersecting walks as restricted, negative, positive or prohibited maneuvers. We have shown how to incorporate this model into Dijkstra's algorithm so that no adjustment of the underlying road network graph is needed. The running time of the proposed Algorithm 1 is only marginally larger than that of ordinary Dijkstra's algorithm (Theorem 1) in practical networks.

Our algorithm can be relatively straightforwardly extended to a bidirectional algorithm by running it simultaneously from the start vertex in the original network and from the target vertex in the reversed network. A termination condition must reflect the fact that chained contexts of vertex-context pairs scanned in both directions might contain maneuvers as subwalks. Furthermore, since the A* algorithm is just an ordinary Dijkstra's algorithm with edge weights adjusted by a potential function, our extension remains correct for A* if the road network is proper (Definition 4, namely iii.) even with respect to this potential function. Finally, we would like to highlight that, under reasonable assumptions, our model can be incorporated into many established route planning approaches.

Acknowledgments

We want to thank reviewers of both this article and our original paper submitted to MEMICS '11 [20], who took the time to carefully read what we wrote and make corrections, additions, suggestions, and improvements to our original words.

This research has been supported by the grant of Czech Science Foundation No. P202/11/0196.

REFERENCES

[1] Anez, J.—De La Barra, T.—Perez, B.: Dual Graph Representation of Transport Networks. Transportation Research Part B: Methodological, Vol. 30, 1996, No. 3, pp. 209–216.

[2] Cherkassky, B.–Goldberg, A. W.—Radzik, T.: Shortest Paths Algorithms: Theory and Experimental Evaluation. Mathematical Programming, Vol. 73, 1996, No. 2, pp. 129–174.

[3] Delling, D.—Wagner, D.: Time-Dependent Route Planning. In: Robust and Online Large-Scale Optimization, LNCS, pp. 207–230, Springer 2009.

[4] Delling, D.—Sanders, P.—Schultes, D.—Wagner, D.: Engineering Route Planning Algorithms. In Algorithmics of Large and Complex Networks, Lecture Notes in Computer Science, pp. 117–139, Springer, Berlin, Heidelberg 2009.

[5] Dijkstra, E.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, Vol. 1, 1959, pp. 269–271.

[6] Gutierrez, E.—Medaglia, A.: Labeling Algorithm for the Shortest Path Problem with Turn Prohibitions with Application to Large-Scale Road Networks. Annals of Operations Research, 157:169–182, 2008, 10.1007/s10479-007-0198-9.

[7] Hart, P. E.—Nilsson, N. J.—Raphael, B.: Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". SIGART, Vol. 1, 1972, No. 37, pp. 28–29.

[8] Hliněný, P.—Moriš, O.: Scope-Based Route Planning. In ESA '11: Proceedings of the 19th Conference on Annual European Symposium, pp. 445–456, Springer Verlag, Berlin Heidelberg 2011, arXiv:1101.3182 (preprint).

[9] Jiang, J.—Han, G.—Chen, J.: Modeling Turning Restrictions in Traffic Network for Vehicle Navigation System. In Proceedings of the Symposium on Geospatial Theory, Processing, and Applications 2002.

[10] Kirby, R. F.—Potts, R. B.: The Minimum Route Problem for Networks with Turn Penalties and Prohibitions. Transportation Research, Vol. 3, 1969, pp. 397–408.

[11] Pallottino, S.—Scutellà, M. G.: Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. Technical report, Univ. of Pisa 1997.

[12] Pohl, I. S.: Bi-Directional and Heuristic Search in Path Problems. Ph. D. thesis, Stanford University, Stanford (CA), USA, 1969.

[13] Schultes, D.: Route Planning in Road Networks. Ph. D. thesis, Karlsruhe University, Karlsruhe (Germany), 2008.

[14] Villeneuve, D.—Desaulniers, G.: The Shortest Path Problem with Forbidden Paths. European Journal of Operational Research, Vol. 165, 2005, No. 1, pp. 97–107.

[15] Winter, S.: Modeling Costs of Turns in Route Planning. GeoInformatica, Vol. 6, 2002, pp. 345–361. 10.1023/A:1020853410145.

[16] Ziliaskopoulos, A. K.—Mahmassani, H. S.: A Note on Least Time Path Computation Considering Delays and Prohibitions for Intersection Movements. Transportation Research Part B: Methodological, Vol. 30, 1996, No. 5, pp. 359–367.

[17] Ahuja, R. K.—Orlin, J. B.—Pallottino, S.—Scutellà, M. G.: Minimum Time and Minimum Cost-Path Problems in Street Networks with Periodic Traffic Lights. Transportation Science, Vol. 36, 2002, No. 5, pp. 326–336.

[18] Ahmed, M.—Lubiw, A.: Shortest Paths Avoiding Forbidden Subpaths. In Proceedings of the 26$^{\text{th}}$ International Symposium on Theoretical Aspects of Computer Science (STACS), 2009, pp. 63–74.

[19] Elmasry, A.: Pairing Heaps with $O(\log \log N)$ Decrease Cost. In Proceedings of the 12$^{\text{th}}$ Annual ACM-SIAM Symposium on Discrete Algorithms – SODA '09, pp. 471–476.

[20] Hliněný, P.—Moriš, O.: Generalized Maneuvers in Route Planning. In MEMICS '11: Selected LNCS Proceedings of the 7th Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, pp. 155–166, Springer, Berlin Heidelberg 2012, ArXiv:1101.3182 (preprint).

Petr Hliněný received his Ph. D. degree in algorithms, combinatorics, and optimization from Georgia Institute of Technology, Atlanta, USA in 1999, and in discrete mathematics from Charles University, Prague, CZ in 2000. Currently he is an Associate Professor at Faculty of Informatics, Masaryk University, Brno, Czech Republic. His research interests are in and topological graph theory, and in parameterized complexity theory.

Ondrej Moriš received his Master degree in computer systems from the Faculty of Informatics, Masaryk University, Brno, Czech Republic in 2010. Since September 2010, he is a Ph. D. student of Petr Hliněný and his research is aimed at route planning and algorithm engineering. Apart from this, he is also interested in parameterized algorithms and structural graph theory.