Preprocessing Speed-Up Techniques Is Hard*

Reinhard Bauer, Tobias Columbus, Bastian Katz,
Marcus Krug, and Dorothea Wagner

Faculty of Informatics, Karlsruhe Institute of Technology (KIT)
{Reinhard.Bauer,Bastian.Katz,Marcus.Krug,Dorothea.Wagner}@kit.edu,
columbus@informatik.uni-karlsruhe.de

Abstract. During the last years, preprocessing-based techniques have been developed to compute shortest paths between two given points in a road network. These *speed-up techniques* make the computation a matter of microseconds even on huge networks. While there is a vast amount of experimental work in the field, there is still large demand on theoretical foundations. The preprocessing phases of most speed-up techniques leave open some degree of freedom which, in practice, is filled in a heuristical fashion. Thus, for a given speed-up technique, the problem arises of how to fill the according degree of freedom optimally. Until now, the complexity status of these problems was unknown. In this work, we answer this question by showing NP-hardness for the recent techniques.

1 Introduction

Computing shortest paths in graphs is used in many real-world applications like route-planning in road networks or for finding good connections in railway timetable information systems. In general, DIJKSTRA's algorithm computes a shortest path between a given source and a given target. Unfortunately, the algorithm is slow on huge datasets. Therefore, it cannot be directly used for applications like car navigation systems or online working route-planners that require an instant answer of a source-target query.

Often, this problem is coped with by dividing the computation of the shortest paths into two stages. In the *offline stage*, some data is precomputed that is used in the *online stage* to answer a source-target query heuristically faster than DIJKSTRA's algorithm. Such an approach is called a *speed-up technique* (for Dijkstra's algorithm). During the last years, speed-up techniques have been developed for road networks (see [1] for an overview), that make the shortest path computation a matter of microseconds even on huge road networks consisting of millions of nodes and edges.

Usually, the offline stage leaves open some degree of freedom, like the choice of how to partition a graph or of how to order a set of nodes. The decision taken to fill the respective degree of freedom has direct impact on the search space of the online stage and therefore on the runtime of a query. Currently,

* Partially supported by the DFG (project WAG54/16-1).

T. Calamoneri and J. Diaz (Eds.): CIAC 2010, LNCS 6078, pp. 359–370, 2010.

these decisions are made in a purely heuristical fashion. A common trade-off is between preprocessing time/space and query time/search space. In this paper we show the NP-hardness of preprocessing the offline stage such that the average search space of the query becomes optimal. For each technique, we demand that the size of the preprocessed data should be bounded by a given parameter. This model is used because practitioners in the field usually compare their results by absolute query times, size of the search space, size of preprocessing and time needed for the preprocessing. In practice, the basic technique can be enriched by various heuristic improvements. We will not consider such improvements and stick to the basic core of each technique. This implies that, for the sake of simplicity, some techniques are slightly altered.

The techniques considered are ALT [2], Arc-Flags [3,4], SHARC [5], Highway Node Routing / Multilevel-Overlay Graph [6,7,8] and Contraction Hierarchies [9]. We left out Geometric Containers [10,11], Highway Hierarchies [12,13] and Reach-Based Pruning [14,15,16] as their offline stage only contains tuning parameters but no real degree of freedom. However, two interesting aspects of Reach-Based Pruning are included. We further did not work on Transit Node Routing [17,18] as this is a framework for which also parts of the query-algorithm are to be specified.

Related Work. There is a huge amount of work on speed-up techniques. An overview on experimental work can be found in [1]. There is large demand on a theoretical foundation for the field and there exists only few theoretical work: In [19] results are given for a problem that is related to the technique of inserting *shortcuts* to the underlying graph. Recently, a graph-generator for road networks was given [20] and it is shown that graphs evolving from that generator possess a property called *low highway dimension*. For graphs with this property, a special preprocessing technique is proposed and runtime guarantees for Reach-Based Routing, Contraction Hierarchies, Transit Node Routing and Sharc using that preprocessing technique are given.

2 Preliminaries

Some proofs have been omitted due to space restrictions. These proofs can be found in [21].

Throughout the work $G = (V, E, len)$ denotes a directed weighted graph with n nodes, m edges and positive length function $len : E \to \mathbb{R}^+$. Given an edge (u, v) we call u the *source node* and v the *target node* of (u, v). Further, (u, v) is an *incoming edge* of v and an *outgoing edge* of u. With $\overline{G}$ we denote the reverse graph, i.e. the graph $(V, \{(v, u) \mid (u, v) \in E\})$. A path P from x_1 to x_k in G is a finite sequence $x_1, x_2, \ldots, x_k$ of nodes such that $(x_i, x_{i+1}) \in E$, $i = 1, \ldots, k-1$. The *length* of a path P in G is the sum of the length of all edges in P. A shortest path between nodes s and t is a path from s to t with minimum length.

Given two nodes s and t, the distance $dist(s,t)$ from s to t is the length of a shortest path between s and t and infinity if no s-t-path exists.

Dijkstra's algorithm. Given a graph $G = (V, E)$ with length function $len : E \to \mathbb{R}^+$ and a root $s \in V$, Dijkstra's algorithm finds the distances from s to all nodes in the graph. For each node v in the graph, the algorithm maintains a status marker, indicating exactly one of the states *unvisited, visited* or *settled* and a distance label $d(v)$. A min-based priority queue Q is provided that contains all visited nodes, keyed by the distance label.

Each node v is initialized to be unvisited and $d(v)$ is set to be infinity. Then, $d(s)$ is set to be 0, s is set to be visited and inserted into Q. While there are visited nodes, the algorithm extracts one node v with minimal distance label from Q, marks v as finished and *relaxes* all of its outgoing edges (v, w). An edge (v, w) is relaxed as follows: If $d(w) \leq d(v) + len(v, w)$ nothing is to be done. Otherwise $d(w)$ is set to be $d(v) + len(v, w)$. If w is already visited, its priority in the queue is updated, otherwise it is marked as visited and inserted into the queue.

There are many possibilities to break ties when extracting nodes from the queue. Throughout this work, we additionally identify every node uniquely with an integer between 1 and $|V|$. Among all nodes with minimal distance in the queue, the smallest integer gets extracted first.

In this work we focus on s-t-queries, i.e. queries for which only a shortest s-t is of interest. Hence, the algorithm can break after the node t has been marked as settled. The *search-space* of the query is the set of nodes, settled up to that point.

Bidirectional Search. This approach starts a Dijkstra's search rooted at s on G (the *forward search*) and one rooted at t on $\overline{G}$ (the *backward search*). Whenever a node has been settled it is to be decided if the algorithm changes to the opposite search. A simple approach is to swap the direction every time a node is settled. The *distance balanced* bidirectional search changes to the other direction iff the minimal distance label of nodes in the queue is greater than the minimal distance label of nodes in the contrary queue. There are different possible stopping criteria for bidirectional search which get specified for the particular speed-up technique applied. During the run of the algorithm, the *tentative distance* is $\min\{\text{dist}(s,u)+\text{dist}(u,t)|\ u \text{ has been settled by both searches}\}$. Finally, $\text{dist}(s,t) = \min\{\text{dist}(s,v)+\text{dist}(v,t)\}$ over all nodes v, that get settled from both directions. The search-space of a bidirectional search is the union of the search-spaces of forward and backward search. We consider the search space to be a multi-set, i.e. when computing the size of the search space we count nodes that get settled in both directions twice.

Speed-up techniques. The query of each speed-up technique we consider is either a modified Dijkstra's algorithm or a modified bidirectional search. The output of an s-t-query is $\text{dist}(s,t)$. We do not consider extra techniques like path-unpacking (see [1] for a description). For a given technique, we write $V_{\mathcal{F}}(s,t)$ for the size of the search space of an s-t-query when choosing $\mathcal{F}$ to fill the particular degree of freedom.

3 Reach-Based Pruning

Reach is a centrality measure indicating whether a node lies in the middle of a long shortest path. More formally, the reach $\mathcal{R}_P(v_i)$ of a node v_i with respect to a path $P = (v_1, \ldots, v_k)$ is $\min\{len(v_1, \ldots, v_i), len(v_i, \ldots, v_k)\}$. The reach $\mathcal{R}(v)$ of a node with respect to a graph G is $\max_{\{P \in \mathrm{SP} \mid v \in P\}} \mathcal{R}_P(v)$ where SP denotes the set of all shortest paths in G. For ease of notation, we consider a single vertex to be a path of length 0.

There exist different variants of how to use reach for pruning the search-space of a bidirectional Dijkstra's search, all of them sharing the same main idea. We use the *self-bounding* query explained later. In practice the approach is mixed with other ingredients like ALT, contraction and the computation of upper bounds for reach-values which we do not consider here. Further, inspired by Contraction Hierarchies we relax the stopping criterion. This has been shown to be reasonable by experimental tests.

The reach-query is bidirectional Dijkstra's algorithm with the following two modifications: First, no stopping criterion is used, hence all vertices reachable from the source get settled. Second, a node v is not settled if $\mathcal{R}(v)$ is smaller than its priority in the queue. Note that v can still get visited. We denote by $d^+(v)$ and $d^-(v)$ the length of the shortest path from s and to t respectively, found by visiting or settling node v. Finally, $\mathrm{dist}(s,t)$ is given by $\min(d^+(v) + d^-(v))$ over all nodes v that are visited or settled from both directions.

Search-Space Minimal Reach. In case shortest paths are not unique the technique still computes correct distances even if only considering one shortest path for each source-target pair. The Problem MinReach is that of choosing these shortest paths such that the resulting average search-space becomes minimal. More formally, we choose a set $\mathcal{P}$ of shortest paths and compute $\mathcal{R}(v)$ by $\max_{\{P \in \mathcal{P} \mid v \in P\}} \mathcal{R}_P(v)$. We denote by $V_{\mathcal{P}}(s,t)$ the search space of the s-t-reach-query using this reach-values for pruning.

*Problem (*MinReach*).* Given a graph $G = (V, E, len)$, choose $\mathcal{P} \subseteq \mathrm{SP}$ such that $\mathcal{P}$ contains at least one shortest s-t path for each pair of nodes $s, t \in V$ for which there is an s-t-path and such that $\sum_{s,t \in V} V_{\mathcal{P}}(s,t)$ is minimal.

Theorem 1. *Problem* MinReach *is NP-hard(even for directed acyclic graphs).*

Proof. We make a reduction from Exact Cover by 3-Sets (X3C). W.l.o.g we may assume $\bigcup_{c \in C} = U$. Given an X3C-instance (U, C) with $|U| = 3q$ we construct a MinReach-instance $G = (V, E)$ as follows: $V = \{a\} \cup C \cup U$ where a is an additional vertex. There is an edge (a, c) for each $c \in C$. There is an edge $(c, u) \in C \times U$ if, and only if $u \in c$. All edge lengths are 1. The construction is polynomial, see Figure 1 for a visualization.

It is $\mathcal{R}(u) = 0$ for $u \in \{a\} \cup U$. Hence, these nodes only get settled as start nodes from the forward search or as target nodes from the backward search. Given the set $\mathcal{P}$, we denote the search space starting at node z by $V^+_{\mathcal{P}}(z)$ and $V^-_{\mathcal{P}}(z)$ for forward and backward search, respectively. We decompose

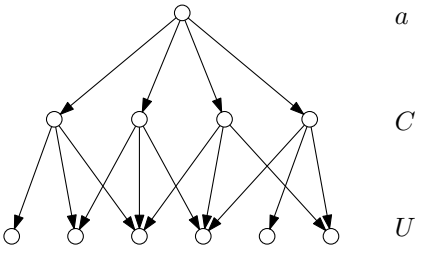

Fig. 1. Graph G constructed from the X3C-instance $\{1,2,3\},\{2,3,4\},\{3,4,6\},\{4,5,6\}$

$$\sum_{s,t\in V} |V_{\mathcal{P}}(s,t)| = |V| \left(|V^+(a)| + \underbrace{\sum_{s\in U\cup C} |V^+(s)|}_{=|U\cup C|} + \underbrace{\sum_{t\in\{a\}\cup C} |V^-(t)|}_{=|C|+1} + \sum_{t\in U} |V^-(t)| \right)$$

Claim. There is a set $\mathcal{P}$ such that $|V|\,[(1+q)+|U\cup C|+|C|+1+2|U|] \geq \sum_{s,t\in V} V_{\mathcal{P}}(s,t)$ if and only if there is an exact cover for (U,C).

"if": When computing the reach-values of nodes in C we only have to consider paths that start with a and end in U because paths consisting of only one edge do not contribute to reach-values greater than 0. Let $C' \subseteq C$ be an exact cover of (U,C). Further let $C'(u)$ denote the $c' \in C'$ with $u \in c'$. We set $\mathcal{P} = \{(a,C'(u),u)|\ u \in U\}$. Then, for each $c \in C$ we have $\mathcal{R}(c) = 1$ if there is a path (a,c,u) in $\mathcal{P}$ and $\mathcal{R}(c) = 0$ otherwise. Hence, $|V^+(a)| = 1+q$ and $|V^-(u)| = 2$ for each $u \in U$. This yields the claimed bound.
"only if": Let $\mathcal{P} \subseteq$ SP be such that $|V|\,[(1+q)+|U\cup C|+|C|+1+2|U|] \geq \sum_{s,t\in V} V_{\mathcal{P}}(s,t)$. We show that $C' = \{c \in C|\ (a,c,u) \in \mathcal{P}\}$ is an exact cover of (U,C). As $\mathcal{P}$ has to include one shortest a-u path for each $u \in U$ we know C' covers of U. With the above decomposition of the search-space we know that $|V^+(a)| + \sum_{t\in U} |V^-(t)| \leq 1+q+2|U|$. It is $V^+(a) = \{a\} \cup \{c \in C|\ \mathcal{R}(c) \geq 1\} = \{a\} \cup C'$ and, for $u \in U$, $V^-(u) = \{u\} \cup \{c \in C|\ \mathcal{R}(c) \geq 1, u \in c\} = \{u\} \cup \{c \in C|\ (a,c,u) \in \mathcal{P}\} \geq 2$. Hence $|C'| \leq q$.

External Shortcuts for Reach-Based Pruning. This is an enhancement for reach-based pruning similar to problem ExtShortcutsArcFlags. We assume that, given the input graph G and a parameter k, we are allowed to insert a set $\mathcal{S}$ of k shortcuts to G. The resulting graph G' is the input of the search technique MinReach and we denote the resulting search-space of an s-t-query by $V_{\mathcal{S}}(s,t)$. One can solve the MinReach-part of the preprocessing-phase by a heuristic approach. In that case, one can show the NP-hardness of inserting shortcuts for a wide range of strategies. We show that it is NP-hard to insert the shortcuts even if we are given an oracle that optimally solves problem MinReach in constant time.

*Problem (*ExtShortcutsReach*).* Given a graph $G = (V, E, len)$ and a positive integer k, insert a set $\mathcal{S}$ of k shortcuts to G, such that $\sum_{s,t \in V} V_{\mathcal{S}}(s,t)$ is minimal.

Theorem 2. *Problem* ExtShortcutsReach *is NP-hard (even for directed acyclic graphs and even if there is an oracle that solves Problem* MinReach *in constant time).*

Proof. We make a reduction from X3C. Let (U, C) be an instance of X3C with $|U| = 3q$. W.l.o.g we may assume $\bigcup_{c \in C} = U$. We construct an instance $(G = (V, E, len), k = q)$ of ExtShortcutsReach as follows: the set V consists of two nodes c^- and c^+ for each $c \in C$, one node u for each $u \in U$, one additional node a and an additional set M with $|M|$ to be specified later. There is an edge (c^+, u) with length 2 iff $u \in c$. Further, there are edges (a, c^-) with length 2 and (c^-, c^+) with length 1 for each $c \in C$. Moverover, there is an edge (m, a) with length 1 for each $m \in M$. We set $k = q$. The transformation is polynomial as $|M|$ will be polynomial in the input size, see Figure 2 for a visualization.

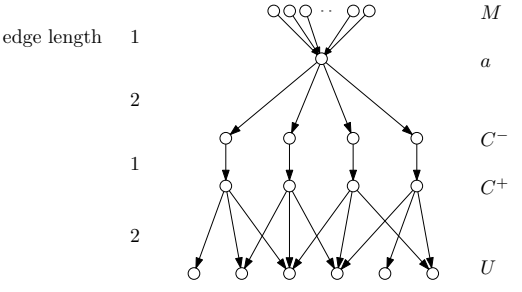

Fig. 2. Graph G constructed from the X3C-instance $\{1,2,3\}, \{2,3,4\}, \{3,4,6\}, \{4,5,6\}$

Given the set $\mathcal{S}$, we denote the search space starting at node z by $V^+_{\mathcal{S}}(z)$ and $V^-_{\mathcal{S}}(z)$ for forward and backward search, respectively. It is $\mathcal{R}(a) \leq 1$ and $\mathcal{R}(m) = 0$ for $m \in M$. Hence

$$\sum_{s,t \in V} V_{\mathcal{S}}(s,t) = |V| \underbrace{\sum_{z \in \{a\} \cup C^- \cup C^+ \cup U} \big(V^-(z) + V^+(z)\big)}_{\leq 2|\{a\} \cup C^- \cup C^+ \cup U|^2} + |V| \sum_{z \in M} (\underbrace{V^-(z)}_{=|1|} + V^+(z))$$

We call a shortcut assignment $\mathcal{S}$ *set covering* if $\mathcal{S}$ contains, for each $u \in U$, a shortcut (a, c^+) such that $u \in c$.

Claim. Let $\mathcal{S}$ be set-covering. Then $\mathcal{P}$ can be chosen such that $\sum_{m \in M} V^+_{\mathcal{S}}(m) \leq 2|M|$.

Let m be in M. It is $\mathcal{R}(u) = 0$ for $u \in U$. Hence $V^+(m) \cap U = \emptyset$. Further, for $c^+ \in C^+$ we have $\mathcal{R}(c^+) \leq 2$ and $\text{dist}(m, c^+) = 4$. Hence $V^+(m) \cap C^+ = \emptyset$. As $\mathcal{S}$

is set-covering, we can choose $\mathcal{P}$ such that paths starting in $M \cup \{a\}$ and ending in U do not to contain a node in C^-. If we do so, $\mathcal{R}(c^-) \leq 2 < \text{dist}(m, c^-)$ for c^- in C^-. Hence $V^+(m) \subseteq \{m, a\}$.

Claim. Assume $|M| > k$. Let $\mathcal{S}$ and $u^* \in U$ be such that $(a, u^*) \notin \mathcal{S}$ and such that for all $c \in C$ with $u^* \in c$, we have $(a, c^+) \notin \mathcal{S}$. Then $\sum_{m \in M} V^+_{\mathcal{S}}(m) \geq 3|M|$.

As $|M| > k$ we have at least one node $m' \in M$ such that $(m', v) \notin \mathcal{S}$ for all $v \in V$. Hence, (m', a, c^-) is the only shortest path for $c^- \in C$ and $\mathcal{R}(a) \geq 1$. Therefore a gets settled from each $m \in M$. Further, a shortest m'-u^*-path starts with (m', a, c^-_*) for a $c^-_* \in C^-$. Hence, $\mathcal{R}(c^-_*) \geq 3$ and c^-_* gets settled from all $m \in M$.

Claim. We specify $|M| = \max\{k, 2|\{a \cup C^- \cup C^+ \cup U\}|^2\}+1$. Then $\sum_{s,t \in V} V_{\mathcal{S}}(s, t)$ $\leq |V|(2|\{a \cup C^- \cup C^+ \cup U\}|^2 + |M| + 2|M|)$ if and only if there is an exact cover for (U, C).

Let C' be an exact cover of (U, C). Then $\{(a, c^+) |\ c \in C'\}$ is set-covering and the bound on the search-space holds with the above claims. On the other hand let $\sum_{s,t \in V} V_{\mathcal{S}^*}(s, t) \leq |V|(2 \cdot |\{a\} \cup C^- \cup C^+ \cup U|^2 + |M| + 2|M|)$. With the last claim we know that for each $u \in U$ there must be either a shortcut $(a, u) \in \mathcal{S}^*$ or a shortcut (a, c^+) with $u \in c$. We gain a shortcut assignment $\mathcal{S}'$ out of $\mathcal{S}^*$ by copying all shortcuts of the form (a, c^+) in $\mathcal{S}$ and taking, for each shortcut of the form $(a, u) \in \mathcal{S}^*$, one arbitrary shortcut (a, c^+) with $u \in c$. The set $\{c \mid (a, c^+) \in \mathcal{S}'\}$ is a cover of (U, C) of size q.

4 Highway Node Routing (Min-Overlay Graph)

Given the input graph $G = (V, E)$ this technique chooses a sequence $V := V_0 \supseteq V_1 \supseteq \ldots \supseteq V_L$ of sets of nodes. Then, a sequence $(G_0, G_1, \ldots, G_L)$ of graphs is computed which is defined by $G_0 := G$ and for each $l > 0$ by $G_l = (V_l, E_l)$ with $E_l := \{(s, t) \in V_l \times V_l \mid \forall$shtst s-t-paths $(s, u_1, \ldots, u_k, t)$ in G_{l-1} is $u_1, \ldots, u_k \notin V_l\}$. The length of an edge (u, v) in G_{i+1} is the length of a shortest u-v-path in G_i. Note that, given nodes $u, v \in V_1$ for which the the edge (u, v) is the only shortest u-v path in G_0, (u, v) is contained in E_1. The *level* i of a node v, is the highest index i such that $v \in V_i$. The *multi-level overlay graph* $\mathcal{G}$ is given by $\mathcal{G} = (V, E_1 \cup \ldots \cup E_L)$.

The query is a modified distance-balanced bidirectional Dijkstra's algorithm in $\mathcal{G}$. From a node of level i, only edges in $E_i \cup \ldots \cup E_L$ are relaxed. The forward (backward) search is aborted when all keys in the forward (backward) priority queue are greater than or equal to the tentative distance. For the NP-completeness proof, we restrict to 2-level min-overlay graphs:

*Problem (*HighwayNodePreProcess*).* Given a graph $G = (V, E)$ and an integer $F \geq |E|$, choose $V_1 \subseteq V$, such that $|E \cup E_1| \leq F$ and $\sum_{s,t \in V} V_{V_1}(s, t)$ is minimal.

We observe that a feasible solution always must exist as we could choose $V_1 = V$.

Theorem 3. *Problem* HighwayNodePreProcess *is NP-hard.*

5 ALT

Goal-directed search is a variant of Dijkstra's algorithm which assigns a different priority to the nodes in the queue. For a node v, let $p(v)$ denote the priority of v in the queue when applying Dijkstra's algorithm (that means $p(v)$ is the tentative distance to v). Goal-directed search adds a potential $\Pi_t(v)$ depending on the target t to the nodes priority, i.e. the priority of v (when applying goal-directed search) is $\Pi_t(v) + p(v)$. An equivalent formulation (that implicitely substracts the constant $\Pi_t(s)$ from the priority) is as follows. Given the potential function $\Pi_t : V \to \mathbb{R}^+$, goal-directed search is Dijkstra's algorithm applied on G with altered edge lengths $\overline{len}(u,v) = len(u,v) - \Pi_t(u) + \Pi_t(v)$ for all edges $(u,v) \in E$. A potential is called *feasible* if $\overline{len}(u,v) \geq 0$ for every edge (u,v). The ALT-algorithm [2] is goal-directed search with a special potential function. Initially, a set $L \subset V$ of 'landmarks' is chosen. For a landmark $l \in L$ we define

$$\Pi_t^{l+}(v) := \mathrm{dist}(v,l) - \mathrm{dist}(t,l) \tag{1}$$

$$\Pi_t^{l-}(v) := \mathrm{dist}(l,t) - \mathrm{dist}(l,v) \tag{2}$$

We use the convention $\infty - \infty := 0$. Accordingly, for a set L of landmarks, the potential is

$$\Pi_t^L(v) := \max_{l \in L} \left\{ \Pi_t^{l+}(v), \Pi_t^{l-}(v), 0 \right\}.$$

Note that this potential is feasible and that $\Pi_t^L(t) = 0$. We denote by $V_L(s,t)$ and $V_\Pi(s,t)$ the search space of an s-t-ALT-query using landmarks L and potential Π, respectively.

The problem MinALT is that of assigning a given number of landmarks to a graph (and thus using only a given amount of preprocessing space), such that the expected number of settled nodes gets minimal.

*Problem (*MinALT*).* Given a directed graph $G = (V, E, len)$ and an integer r, find a set $L \subset V$ with $|L| = r$ such that $\sum_{s,t \in V} V_L(s,t)$ is minimal.

Theorem 4. *Problem* MinALT *is NP-hard.*

6 Arc-Flags

Main Technique. This approach partitions the set of nodes V into k cells $\mathcal{V} = (V_1, V_2, \ldots, V_k)$. For a node w, we write $\mathcal{V}(w) = V_i$ iff $w \in V_i$. To each edge (u,v) a k-bit vector $\mathcal{F}_{(u,v)}$ is attached, such that $\mathcal{F}_{(u,v)}(V_i)$ is true iff a shortest path starting with the edge (u,v) and ending at a node $t \in V_i$ exists. The Arc-Flags s-t-query is the variant of Dijkstra's algorithm which only relaxes edges (u,v) for which $\mathcal{F}_{(u,v)}(\mathcal{V}(t))$ is true.

*Problem (*ArcFlags*).* Given a graph $G = (V, E, len)$ and an integer k, find a k-partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of V such that $\sum_{s,t \in V} V_{\mathcal{V}}(s,t)$ is minimal.

Theorem 5. *Problem* ArcFlags *is NP-hard.*

Search-Space Minimal Arc-Flags. This problem models a special aspect of the arc-flag technique. In case shortest paths are not unique, the situation may occur that one can improve the search-space by changing some flags from true to false while still guaranteeing the query to compute the correct distance. We consider the partition $\mathcal{V} = (V_1, V_2, \ldots, V_k)$ of the graph $G = (V, E)$ to be already given and change the rule of how to compute the vectors $\mathcal{F}_{(u,v)}$: For each pair of nodes s and t there shall be at least one shortest path $s = v_1, \ldots, v_\ell = t$ such that $\mathcal{F}_{(v_i,v_{i+1})}(\mathcal{V}(t))$ is true for $i = 1, \ldots, \ell - 1$. We may assume that for each edge (u, v) with $\mathcal{F}_{(u,v)}(V_i) = \mathit{true}$ there is at least one shortest path starting with (u, v) that leads to V_i. The problem MinFlags is that of how to assign values to the vectors $\mathcal{F}_{(u,v)}$ such that the resulting average search-space of an arc-flags query becomes minimal.

*Problem (*MinFlags*).* Given a graph $G = (V, E, len)$ and a partition $\mathcal{V} = (V_1, \ldots, V_k)$ of V, compute an arc-flag assignment $\mathcal{F} = (\mathcal{F}_{(u,v)})_{(u,v)}$ for G such that $\sum_{s,t \in V} V_{\mathcal{F}}(s, t)$ is minimal.

Theorem 6. *Problem* MinFlags *is NP-hard (even for directed acyclic graphs).*

External Shortcuts for Arc-Flags. This problem models an enhancement of the arc-flag technique that is used within the SHARC-algorithm. We are in the situation that the graph $G = (V, E, len)$, a h-partition $\mathcal{V} = (V_1, \ldots V_h)$ of G and an integer k are already given. A shortcut is an edge (u, v) that is added to the graph for which $len(u, v) = \mathrm{dist}(u, v)$. A shortest path $v_1, v_2, \ldots, v_\ell$ is called *canonical* if it is edge-minimal among all shortest v_1-v_ℓ-paths and if $(v_1, \ldots, v_\ell)$ is lexicographically minimal among all edge-minimal shortest v_1-v_ℓ-paths.

The query is similar to the arc-flag query but the preprocessing stage differs. We are allowed to add k shortcuts to the graph, afterwards the vectors $\mathcal{F}_{(u,v)}$ get computed as follows: $\mathcal{F}_{(u,v)}(V_i)$ is true iff a *canonical* shortest path starting with the edge (u, v) and ending at a node $t \in V_i$ exists. W.l.o.g we do not insert shortcuts that are already present in the graph.

*Problem (*ExtShortcutsArcFlags*).* Given a graph $G = (V, E, len)$, a partition $\mathcal{V} = (V_1, \ldots, V_h)$ of V and a positive integer k, insert a set $\mathcal{S}$ of k shortcuts to G, such that $\sum_{s,t \in V} V_{\mathcal{S}}(s, t)$ is minimal.

Theorem 7. *Problem* ExtShortcutsArcFlags *is NP-hard(even for directed acyclic graphs).*

7 Contraction Hierarchies

Throughout this section we work on undirected graphs. This is no restriction as the results also hold for directed graphs with edges always being symmetric. Given the input graph $G = (V, E)$, the preprocessing of Contraction Hierarchies (CH) consists of distuingishing a total order $\prec$ on V and iteratively *contracting*

the $\prec$-least node until G is empty. A node v is contracted as follows: For each pair of edges $\{u, v\}$, $\{v, w\}$ such that (u, v, w) is the only shortest u-w-path, a new edge (u, w) called a *shortcut* is introduced with length $len\{u, v\} + len\{v, w\}$ to G. Afterwards, v and all of its adjacent edges are removed from G. The output of the preprocessing is $\prec$ and the graph $H_{\prec}(G) = (V, E \cup E')$ where E' is the set of all edges that got inserted due to node contraction. We call $H_{\prec} := H_{\prec}(G)$ the *contraction hierarchy* of G and denote by $|H_{\prec}|$ the number of edges $|E'|$.

The CH-query is bidirectional Dijkstra's algorithm on $H_{\prec}(G)$ applying two changes. Firstly, no special stopping criterion exists, the whole reachable subgraph gets settled. Secondly, when settling node u, only edges $\{u, v\}$ with $u \prec v$ get relaxed .

*Problem (*CH Preprocessing*).* Given a graph $G = (V, E)$, a length function $len\colon E \to \mathbb{R}^+$ and a number $K \in \mathbb{Z}_{\geq 0}$, find an order $\prec$ on V, such that $|H_{\prec}(G)| \leq K$ and $\sum_{s,t \in V} V_{\prec}(s, t)$ is minimal?

Theorem 8. *Problem* CH Preprocessing *is* NP*-hard.*

8 Conclusion

Speed-up techniques have been widely studied experimentally for the last years, especially for road-networks. There is large interest in a theoretical foundation of the techniques developed and some first work on the topics have been published [20,19]. In this work we focused on the preprocessing phases of the recent techniques. These usually incorporate a degree of freedom that, in practice, is filled in a heuristical manner. Until now, the complexity status of filling the according degree of freedom was unknown. We settled this question by showing that all variants considered are NP-hard to optimize.

There are numerous open questions for the topic. A reasonable next step to enhance this work is the development of approximation- or fixed parameter tractable algorithms for the preprocessing phase. Efficient algorithms for special graph classes would help to show the bounds of intractability.

When working with special graph classes (either for giving algorithms or showing the complexity status) modeling the two main applications for speed-up techniques, road-networks and public transportation networks would be helpful. Until now there is no experimentally verified model for these two applications (but some first work [20,22]). From a more theoretical point of view we have the question which of the problems can be solved efficiently on trees.

Another interesting question is the following: We assume preprocessing time and space is unbounded, how good can a speed-up technique actually get? Obviously, ALT and Arc-Flags can encode All-Pairs Shortest-Path in a way that a shortest s-t path and $\text{dist}(s, t)$ can be queried in time that is linear in the size of the shortest-path subgraph which yields optimal search-space for these techniques (but of course, the runtime of ALT would not be optimal in this case). The situation is not so clear for Highway-Node Routing, Reach, Highway-Hierarchies

and Contraction Hierarchies. One can show that the average CH-search space of an s-t-query can not guarantee to be better than $\Omega(n)$ for arbitrary graphs and $\Omega(\log n)$ for graphs with bounded degree [21]. Is the second bound tight?

References

1. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
2. Goldberg, A.V., Harrelson, C.: Computing the Shortest Path: A* Search Meets Graph Theory. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 156–165 (2005)
3. Lauther, U.: An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In: Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung. IfGI prints. vol. 22. pp. 219–230 (2004)
4. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) The Shortest Path Problem: Ninth DIMACS Implementation Challenge. DIMACS Book, vol. 74, pp. 41–72. American Mathematical Society, Providence (2009)
5. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. ACM Journal of Experimental Algorithmics 14 (May 2009); 2.4 Special Section on Selected Papers from ALENEX 2008
6. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. In: Vitter, J.S., Zaroliagis, C.D. (eds.) WAE 1999. LNCS, vol. 1668, pp. 110–123. Springer, Heidelberg (1999)
7. Schultes, D., Sanders, P.: Dynamic Highway-Node Routing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 66–79. Springer, Heidelberg (2007)
8. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006), pp. 156–170. SIAM, Philadelphia (2006)
9. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)
10. Wagner, D., Willhalm, T.: Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 776–787. Springer, Heidelberg (2003)
11. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric Containers for Efficient Shortest-Path Computation. ACM Journal of Experimental Algorithmics 10, 1.3 (2005)
12. Sanders, P., Schultes, D.: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 568–579. Springer, Heidelberg (2005)
13. Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 804–816. Springer, Heidelberg (2006)

14. Gutman, R.J.: Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), pp. 100–111. SIAM, Philadelphia (2004)
15. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006), pp. 129–143. SIAM, Philadelphia (2006)
16. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Better Landmarks Within Reach. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 38–51. Springer, Heidelberg (2007)
17. Sanders, P., Schultes, D.: Robust, Almost Constant Time Shortest-Path Queries in Road Networks. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) 9th DIMACS Implementation Challenge - Shortest Paths (November 2006)
18. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Shortest-Path Queries in Road Networks. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 46–59. SIAM, Philadelphia (2007)
19. Bauer, R., D'Angelo, G., Delling, D., Wagner, D.: The Shortcut Problem – Complexity and Approximation. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 105–116. Springer, Heidelberg (2009)
20. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In: Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 782–793 (2010)
21. Bauer, R., Columbus, T., Katz, B., Krug, M., Wagner, D.: Preprocessing Speed-Up Techniques is Hard. Technical Report 2010-04, ITI Wagner, Faculty of Informatics, Universität Karlsruhe, TH (2010), http://digbib.ubka.uni-karlsruhe.de/volltexte/1000016080
22. Eppstein, D., Goodrich, M.T.: Studying (non-planar) road networks through an algorithmic lens. In: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. ACM Press, New York (2008)