Efficient Route Compression for Hybrid Route Planning*

Gernot Veit Batz, Robert Geisberger, Dennis Luxen,
Peter Sanders, and Roman Zubkov

Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany
{batz,luxen,sanders}@kit.edu

Abstract. We describe an algorithmic framework for lossless compression of route descriptions. This is useful for hybrid route planning where routes are computed by a server and then transmitted to a client device in a car using some mobile radio communication where bandwidth may be low. Compressed routes are represented by only a few via nodes which are the connection points when the route is decomposed into unique optimal segments. To reconstruct the route efficiently a client device needs basic but fast route planning capability. Contraction hierarchies make this approach fast enough for practice: Compressing takes only a few milliseconds. And previous experiments suggest that a client can decompress each route segment virtually instantaneously. So, as the segments can be decompressed successively while driving, it is not likely that the driver experiences any delay except for the time needed by the mobile communication.

1 Introduction

Today GPS-based car navigation is quite common. Routes can be computed either by a device located in the car or by a *server* system located in a computing center. The latter requires that routes are transmitted to the *client* device in the car using some *mobile radio communication* like the cellular phone network. We denote this server-based mobile setting as *hybrid route planning*.

Hybrid route planning is not only useful to take the current traffic situation or the latest changes of POI data into account. It also makes the benefits of several advanced route planning algorithms available for car drivers. Such algorithms, which compute high quality routes within milliseconds, are often quite sophisticated and adapting them to work well on mobile devices is usually not trivial – if at all possible. Examples are time-dependent route planning [1–7] where routes depend on the departure time, flexible route planning [8] where routes depend on a freely selectable parameter which models a tradeoff between energy consumption and travel time for example, multi-criteria route planning [9] where routes

* Partially supported by DFG project SA 933/5-1,2. This is a slightly extended version containing some details we omitted in the original version due to page limit. The original publication is available at www.springerlink.com.

are assessed with respect to multiple costs, customizable route planning [10] where cost functions can be altered rapidly, or the computation of alternative routes [11, 12]. Possible benefits are, for example,

- that routes are optimized with respect to the time of day which means that congestions can be avoided based on statistical data,
- that inconvenient roads can be avoided and that toll charges, energy consumption, or detours can be reduced even if travel time is the main objective,
- or that the driver can dynamically choose or fine-tune the cost function.

To make hybrid route planning convenient to use, the latency experienced by the driver should be as small as possible. However, the descriptions of the routes that have to be transmitted over the mobile communication can be quite complex and bandwidths can be low (as in the countryside for example). So, good compression rates are desirable. Also, the time needed for compressing and decompressing the routes has to be small.

In this work we present an algorithmic framework for lossless and efficient compression of route descriptions in the context of hybrid route planning. It provides good compression rates and the running times needed for compressing and decompressing are small. As a result, the user should not notice any latency except for the latency of the mobile communication. Our approach requires that

- the client has basic but fast route planning capability,
- client and server use the same road network topology, and
- the cost function used by the client changes rarely and is known to the server.

A route provided by a sophisticated algorithm running on the server is most probably not an optimal route with respect to a simple cost function that can be handled by the mobile client device. We observe, however, that the route can be composed of a few *unique* route *segments* which are optimal with respect to the client. Our compression exploits this in a simple but effective way: We represent the route by only giving the few locations where this unique optimal route segments meet. We call these locations the *via nodes*. The client can reconstruct the route from the few via nodes by simply computing the optimal routes between them. The uniqueness of the segments guarantees that the reconstructed route is exactly the route originally provided by the server.

To provide efficient decompression, the client device must be able to to perform fast and exact[1] computation of optimal routes. There, it is enough to decompress the *first* route segment fast. All other segments can be decompressed successively when driving. *Contraction hierarchies (CH)* [13, 14] is a fast and exact method for route planning which has also been adapted to run efficiently on mobile devices. Using these mobile CH the client should be able to decompress each segment of the route within less than 0.1 s [15]. From the drivers point of view this is as good as instantaneous. Note that Dijkstra's well known algorithm is *not* an alternative. Though it computes optimal routes, it has running times of more than a second even on server systems which is far to slow.

[1] We speak of *exact* route planning to indicate that the computed routes are optimal with respect to the underlying cost function and no heuristic is used.

Our Contributions. With CH the client is already able to decompress each segment of the route fast enough. So, we focus on the server-side algorithmic methods of computing the compressed representation of a path P as a sequence Q of via nodes. All these algorithmic methods are instances of a generic frame algorithm (Sect. 2). The first instantiation of the frame algorithm is based on Dijkstra's well known algorithm and yields the *minimum* possible number of via nodes. In practice, however, the Dijkstra-based compression is to slow. The second instantiation uses an algorithmic scheme inspired by binary search. It also yields minimal sequences of via nodes. Like the frame algorithm the binary scheme is a generic algorithm itself. More precisely, it requires a subroutine that decides whether a route is the unique optimal route between two nodes. This subroutine is invoked $O(|Q| \log |P|)$ times (Sect. 3).

Realizing the subroutine using the aforementioned CH (Sect. 4) yields a very fast compression technique, fast enough for practice. Interestingly, this approach needs sometimes even less via nodes than the Dijkstra-based approach (Sect. 5). This is because the definition of a unique optimal route is different in the context of CH. Our experiments indicate that via nodes provide good compression rates in practice. Also the running time needed by the compression is quite low if CH are used to realize the subroutine in the generic binary scheme (Sect. 6).

Related Work. We described some of the ideas presented in this work previously in a technical report [16]. To our knowledge, there is no other publication directly covering the efficient representation of routes beyond traditional data compression and error-correction. Tao et al. [17] show how to efficiently compute a representation that includes at least one out of every k consecutive nodes. While this can be seen as a compact representation of a shortest path it is not clear how to conduct a loss-less reconstruction of the represented path. Via nodes have been applied to CH in a different way [11, 12] to provide *reasonable* alternatives to the optimal route. Here, reasonable means that the alternative is not much longer, has not too much in common with the optimal route and is locally optimal.

2 Via Nodes and a Generic Frame Algorithm

We model road networks as directed graphs $G = (V, E)$. As the mobile device has limited main memory and computing power, it uses a very simple cost function: Every edge (u, v) has a constant weight $c(u, v) \in \mathbb{R}_{>0}$ assigned. Routes are modeled as paths in G. A path $\langle u_1, \ldots, u_k \rangle$ is a *shortest* path if it has minimal cost $c(\langle u_1, \ldots, u_k \rangle) := c(u_1, u_2) + \cdots + c(u_{k-1}, u_k)$ among all paths from u_1 to u_k. A shortest path $\langle u, \ldots, v \rangle$ is called a *unique* if it is the only shortest path from u to v. Subpaths of unique shortest paths are unique shortest paths too. By $G^\top$ we denote the *transpose* graph of G where all edges are reversed.

Consider a path $P := \langle u_1, \ldots, u_n \rangle$ which is not necessarily a shortest path. Let $Q := \langle\!\langle u_{i_1}, \ldots, u_{i_k} \rangle\!\rangle$ be a subsequence of P s.t. the subpaths $\langle u_1, \ldots, u_{i_1} \rangle$, $\langle u_{i_k}, \ldots, u_n \rangle$, and $\langle u_{i_j}, \ldots, u_{i_{j+1}} \rangle$ of P are unique shortest paths for all $1 \leq j < k$. Then, we call Q a *representation of P (with via nodes)*. Certainly, P is completely determined by Q. For $|Q| \ll |P|$ we can speak of a compressed representation.

Algorithm 1. Generic frame algorithm computing a representation with via nodes for a path P. Requires a subprocedure *uniqueShortestPrefix*$(Path) : Path$.

```
1 function frameAlgorithm(P : Path) : Sequence
2     Q := ⟪⟫ : Sequence
3     while P ≠ ⟨⟩ do
4         R := uniqueShortestPrefix(P)
5         append last node of R to the end of Q
6         remove prefix R from P
7     remove last node of Q
8     return Q
```

Observation 1. *If all edges in G are unique shortest paths, then all paths in G can be represented with via nodes.*

A trivial representation of P is P itself. If not all edges of G are unique shortest paths, then this property can be established easily by a simple transformation of G: For every edge (u, v) in G we run a Dijkstra search starting from u with the constraint that the edge (u, v) must not be relaxed. This yields a shortest path P_{uv} in the graph $(V, E \setminus \{(u, v)\})$. If $c(P_{uv}) \leq c(u, v)$ holds, we introduce a new node x to G and replace (u, v) by the new edges (u, x) and (x, v) with $c(u, x) := c(x, v) := c(u, v)/2$. In the following we assume that all edges of G are unique shortest paths. Thus, every path in G can be represented with via nodes. A representation Q of P with via nodes is called *minimal* if there exists no other representation Q' of P s.t. $|Q'| < |Q|$. If P is a unique shortest path, then the minimal representation is $\langle\langle\rangle\rangle$.

A prefix $R_i := \langle u_1, \ldots, u_i \rangle$ with $i < n$ is called a *real* prefix of P. If R_i is a unique shortest path, then R_i is called a *unique shortest prefix* of P. If R_{i+1} is *not* a unique shortest path, then R_i is called the *maximal* unique shortest prefix of P. A generic frame algorithm (see Algorithm 1) computes a representation with via nodes for a given path. It requires that a procedure *uniqueShortestPrefix* is present, which computes a unique shortest prefix of a given path. Obviously, *uniqueShortestPrefix* is called $O(|Q|)$ times. The frame algorithm can also be used to find *minimal* representations as the following theorem shows.

Theorem 2. *If uniqueShortestPrefix provides the maximal unique shortest prefix, then Algorithm 1 yields a minimal representation with via nodes.*

Proof. Set $\langle s, \ldots, t \rangle := P$ and let $Q = \langle\langle u_1, \ldots, u_k \rangle\rangle$ be the result of the frame algorithm. Assume there is another representation $Q' = \langle\langle v_1, \ldots, v_\ell \rangle\rangle$ of P with $\ell < k$. Then $\langle s, \ldots, v_i \rangle$ is a prefix of $\langle s, \ldots, u_i \rangle$ for $1 \leq i \leq \ell$. For $i = 1$ this is true because $\langle s, \ldots, u_1 \rangle$ is computed by *uniqueShortestPrefix* and hence the maximal unique shortest prefix of P. For $i > 1$ we apply induction and assume $\langle s, \ldots, v_{i-1} \rangle$ is prefix of $\langle s, \ldots, u_{i-1} \rangle$. But then $\langle s, \ldots, v_i \rangle$ must also be prefix of $\langle s, \ldots, u_i \rangle$. Otherwise $\langle u_{i-1}, \ldots, u_i \rangle$, which is the maximal unique shortest prefix of $\langle u_{i-1}, \ldots, t \rangle$, would be a real prefix of the unique shortest path $\langle u_{i-1}, \ldots, v_i \rangle$.

Algorithm 2. Modification of Dijkstra's algorithm computing a maximal unique shortest prefix path of a given path $P = \langle u_1, \ldots, u_n \rangle$.

```
1  function uniqueShortestPrefix(⟨u_1, ..., u_n⟩ : Path) : Path
2      d[u] := ∞, p[u] := ⊥, unique[u] := true for all u ∈ V
3      k := n, d[u_1] := 0
4      M := {(u_1, 0)} : PriorityQueue
5      while M ≠ ∅ do
6          u := M.deleteMin()                                  // settle node u
7          if u = u_i ∈ P then
8              if ¬unique[u_i] or c(⟨u_1, ..., u_i⟩) ≠ d[u_i] then
9                  k := min{k, i − 1}
10         for (u, v) ∈ E do
11             if d[u] + c(u, v) < d[v] then                   // relax edge (u, v)
12                 if d[v] = ∞ then M.insert(v, d[u] + c(u, v))
13                 else M.decreaseKey(v, d[u] + c(u, v))
14                 d[v] := d[u] + c(u, v)
15                 p[v] := u
16                 unique[v] := true
17             else if d[u] + c(u, v) = d[v] then unique[v] := false
18         if d[u] > c(⟨u_1, ..., u_k⟩) then break
19     return ⟨u_1, ..., u_k⟩
```

But this is not possible. Hence, $\langle s, \ldots, v_\ell \rangle$ is a prefix of $\langle s, \ldots, u_\ell \rangle$. But this means that $\langle u_\ell, \ldots, t \rangle$ is subpath of $\langle v_\ell, \ldots, t \rangle$ and hence a unique shortest path. But this contradicts the fact that $\langle u_\ell, \ldots, u_{\ell+1} \rangle$ with $u_{\ell+1} \neq t$ is the maximal unique shortest prefix of $\langle u_\ell, \ldots, t \rangle$. □

All compression methods described in this work are instantiations of the frame algorithm with different realizations of the procedure *uniqueShortestPrefix*.

3 Dijkstra-Based Compression and a Generic Scheme

The first realization of the procedure *uniqueShortestPrefix* which we describe is actually Dijkstra's well known algorithm plus some additional actions. It returns the maximal unique shortest prefix of a given path as we will see (Algorithm 2).

The original version of Dijkstra's algorithm computes shortest paths from a given start node s to all reachable nodes in a graph. To do so it successively labels all nodes w with labels $d[w]$ and $p[w]$, where $d[w]$ is the tentative cost from s to w and $p[w]$ the predecessor of w on the corresponding tentative path. After termination we can obtain a shortest path P_{su} from s to a node u by successively selecting the next predecessor node starting from u:

$$P_{su} \;=\; \big\langle s = p[\ldots p[u]\ldots], \ldots, p[p[u]], p[u], u \big\rangle$$

When a node is *settled* (i.e., removed from the priority queue, Line 6), its tentative cost equals the cost of a shortest path and changes no more. A detailed explanation of Dijkstra's algorithm can be found in textbooks (e.g., [18]). Our modified Dijkstra starts from u_1 and maintains an index k which is initialized with n and repeatedly decreased until $\langle u_1, \dots, u_k \rangle$ is a unique shortest prefix.

Lemma 3. *For $u \in V$ let $P_u := \langle u_1, \dots, p[p[u]], p[u], u \rangle$ be the shortest path found by Algorithm 2. If $unique[w]$ holds for all $w \in P_u$, then P_u is unique.*

Proof. Otherwise, there is a shortest path $P' := \langle u_1, \dots, w', w \rangle$ with $w' \notin P_u$ for some $w \in P_u$. After $(p[w], w)$ and (w', w) have been *relaxed* (Lines 11 to 17) we have $\neg unique[w]$ because P' and $\langle u_1, \dots, p[w], w \rangle \subseteq P_u$ are shortest paths, and $d[w] = d[p[w]] + c(p[w], w) = d[w'] + c(w', w)$ can not further decrease. But this also means that $unique[w]$ will not be changed anymore – a contradiction. □

Lemma 4. *Algorithm 2 computes a unique shortest prefix.*

Proof. Let k_0 be the final value of k. Surely, $\langle u_1, \dots, u_{k_0} \rangle$ is a shortest path. Otherwise, the algorithm would return a real prefix of $\langle u_1, \dots, u_{k_0} \rangle$ because of the second condition in Line 8. Also, $\langle u_1, \dots, u_{k_0} \rangle$ is unique according to Lemma 3 as $unique[u_j]$ holds for $1 \leq j \leq k_0$. Otherwise, we would have $\neg unique[u_j]$ for some u_j at the time when u_j is settled, because *unique* does not change for settled nodes as their tentative cost is already minimal. But then, the algorithm would perform $k := j - 1 < k_0$ which can not be the case. □

Theorem 5. *Algorithm 2 computes the maximal unique shortest prefix.*

Proof. Otherwise, $\langle u_1, \dots, u_{k_0+1} \rangle$ is a unique shortest prefix with k_0 the final value of k. As the algorithm sets k to k_0, one of the conditions in Line 8 must be fulfilled. But as $\langle u_1, \dots, u_{k_0+1} \rangle$ is a shortest path, it is $\neg unique[u_{k_0+1}]$ which holds when u_{k_0+1} is settled. This means that u_{k_0+1} is reached by two different paths with the same cost which must be minimal as it is not decreased afterwards. But we assumed that the shortest path $\langle u_1, \dots, u_{k_0+1} \rangle$ is unique. □

So, if we instantiate the procedure *uniqueShortestPrefix* in the frame algorithm (Algorithm 1) by our modified Dijkstra search (Algorithm 2), we get a method to compute the minimal representation of a given path with via nodes.

But we also consider another realization of *uniqueShortestPrefix* which we call the *generic binary scheme* (Algorithm 3). It is heavily inspired by binary search and like the frame algorithm it is also generic. It requires that a procedure *isUniqueShortestPath* is present, which decides whether a given path is a unique shortest path or not. This subprocedure is invoked $O(\log |P|)$ times.

Corollary 6. *Algorithm 3 computes the maximal unique shortest prefix.*

Instantiating the procedure *uniqueShortestPrefix* in the frame algorithm (Algorithm 1) by the generic binary scheme (Algorithm 3) we get a further generic method to compute the minimal representation with via nodes Q of a path P. Obviously, the procedure *isUniqueShortestPath* is called $O(|Q| \log |P|)$ times.

Algorithm 3. A generic binary scheme checking whether $\langle u_1, \ldots, u_n \rangle$ is a unique shortest path. Requires a subprocedure *isUniqueShortestPath*(*Path*) : *bool*.

```
1 function uniqueShortestPrefix(⟨u_1, ..., u_n⟩ : Path) : Path
2     (ℓ, m, r) := (1, n, n)
3     while ℓ + 1 < r do
4         if isUniqueShortestPath(⟨u_1, ..., u_m⟩) then ℓ := m
5         else r := m
6         m := ⌊(ℓ + 1 + r)/2⌋
7     return ⟨u_1, ..., u_ℓ⟩
```

In the following we instantiate the subprocedure *isUniqueShortestPath* in the generic binary scheme using the aforementioned CH. This yields a quite fast realization. It should be noted, however, that the structure of a CH is different from the structure of the original road network. As a result, via nodes are no longer nodes where unique shortest paths meet, but nodes where paths meet that are *uniquely representable* with respect to CH.

4 Representing Paths Uniquely with CH

In the CH framework [13, 14] we derive a *hierarchical* structure from the original road network G in a relatively expensive *preprocessing* step. There, all nodes of G are ordered by some notion of *importance* with more important nodes higher up in the hierarchy. Roughly, a node is more important, the more shortest paths run over it. The hierarchy is constructed bottom up by successively *contracting* the least important remaining node. Contracting a node v means, that v is removed from the graph while preserving all shortest paths. To preserve the shortest paths we have to insert an artificial *shortcut* edge (u, w) for every removed path $\langle u, v, w \rangle$ which is a unique shortest path at that time. If a shortcut (u, w) is inserted, we set $c(u, w) := c(u, v) + c(v, w)$ and annotate (u, w) with the *middle* node v such that (u, w) can be *expanded* to $\langle u, v, w \rangle$. When inserting (u, w) it may happen that an edge (u, w) is already present. In this case we *merge* the two edges. This means we check whether $c(u, w) > c(u, v) + c(v, w)$ holds, and if it does we replace the middle node by v and the weight by $c(u, v) + c(v, w)$.

Having contracted all nodes we have a hierarchy of graphs which we store in a condensed way: Every node is materialized exactly once and the original edges and all shortcuts are put together. The resulting graph is the actual contraction hierarchy (also abbreviated CH) H. We have $G \subseteq H$. Fig. 1 shows an example.

Corollary 7. *Let H be a CH derived from G. Then he shortest path distances in H and G are equal (but usually H contains paths not present in G).*

We say an edge (u, v) in H leads *upward* if u is less important than v. Otherwise, we say (u, v) leads *downward*. Let $H_\uparrow \subseteq H$ and $H_\downarrow \subseteq H$ be the subgraphs that only consist of upward and downward edges respectively. Then

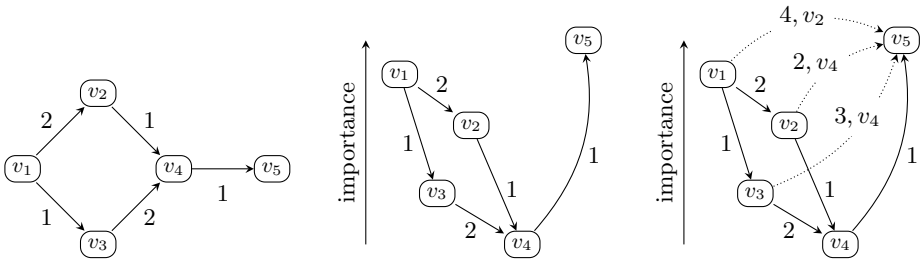

Fig. 1. Example road network (left) whose nodes are ordered by importance according to $v_4 \prec v_3 \prec v_2 \prec v_1 \prec v_5$ (middle). Doing preprocessing we get a CH (right) with three shortcut edges (dotted) annotated with their weight and the respective middle node. When contracting v_3 we insert no shortcut because there is more than one shortest path from v_1 to v_5 in the remaining network which still contains the nodes v_1, v_2, v_5. sec:appendix

we have $H = H_\uparrow \cup H_\downarrow$ where $H_\uparrow$ and $H_\downarrow$ are edge disjoint DAGs. Most probably, a path $P := \langle s, \dots, t \rangle$ in H contains shortcuts. These can be expanded recursively until no shortcuts are present in the resulting path $P' \subseteq G$. We say that P *represents* P'. A path $\langle s, \dots, x, \dots, t \rangle$ in H with $\langle s, \dots, x \rangle \subseteq H_\uparrow$ and $\langle x, \dots, t \rangle \subseteq H_\downarrow$ is called an *up-down-path* in H with *top* node x.

Lemma 8 ([14]). *Let H be a CH derived from G and s, t be two nodes. Then, there is an up-down-path in H that represents a shortest path from s to t in G.*

Up-down-paths being also shortest paths are called *shortest* up-down-paths. Shortest paths in G can only be represented by shortest up-down-paths in H. If there is exactly one shortest up-down-path in H representing a shortest path $P \subseteq G$, we say P is *uniquely representable (by a shortest up-down-path) in H*.

A shortest up-down-path from a node s to a node t can be found by performing a *bidirectional* Dijkstra search that runs *upward*. That is two Dijkstra searches that run at the same time, a *forward* and a *backward* search each starting from s and t and running in $H_\uparrow$ and $H_\downarrow^\top$ respectively.[2] This is exactly what Algorithm 4 does. Similar to the Algorithm 2 we maintain tentative cost, predecessor, and uniqueness information, but separately for forward and backward search: d_s and d_t, for example, denote the tentative cost of forward and backward search respectively. Whenever the two searches meet in a node u, we put u into the set C of top node *candidates*, but only if the weight of the corresponding up-down-path with top node u is minimal at that time (Line 10). After the bidirectional search is finished, we check whether the shortest of the up-down-paths that we found is unique (Lines 19 to 26). Algorithm 4 runs very fast because well-constructed CH are *flat* and *sparse*. This means H only contains few shortcuts and the paths in $H_\uparrow$ and $H_\downarrow^\top$ only have few hops. Note that we also apply *stall-on-demand* [13, 14] to further reduce running time.

[2] In reality we run the two searches in an alternating manner instead of simultaneously.

Algorithm 4. Modified bidirectional Dijkstra search checking whether $\langle s, \dots, t \rangle$ is uniquely representable by a shortest up-down-path in the CH $H = H_\uparrow \cup H_\downarrow$.

```
1  function isUniquelyRepresentable(P := ⟨s, ..., t⟩ : Path) : bool
2      d_X[u] := ∞, p_X[u] := ⊥, unique_X[u] := true for all u ∈ V, X ∈ {s, t}
3      d_s[s] := d_t[t] := 0
4      M_s := {(s, 0)}, M_t := {(t, 0)} : PriorityQueue
5      X := t, C := ∅                                  // search direction, candidate set
6      while M_s ≠ ∅ or M_t ≠ ∅ do
7          if min M_s ∪ M_t > min{d_s[x] + d_t[x] | x ∈ C} ∪ {∞} then break
8          if M_¬X ≠ ∅ then X := ¬X                    // with s = ¬t and t = ¬s
9          u := M_X.deleteMin()
10         if d_s[u] + d_t[u] ≤ min{d_s[x] + d_t[x] | x ∈ C} then C := C ∪ {u}
11         foreach edge (u, v) in H_X do               // H_s := H_↑, H_t := H_↓^⊤
12             if d_X[u] + c(u, v) < d_X[v] then
13                 if d_X[v] = ∞ then M_X.insert(v, d_X[u] + c(u, v))
14                 else M_X.updateKey(v, d_X[u] + c(u, v))
15                 d_X[v] := d_X[u] + c(u, v)
16                 p_X[v] := u
17                 unique_X[v] := true
18             else if d_X[u] + c(u, v) = d_X[v] then unique_X[v] := false
19     if there is exactly one x ∈ C minimizing d_s[x] + d_t[x] then
20         x_0 := argmin_{x∈C} d_s[x] + d_t[x]
21         P_s := ⟨s, ..., p_s[p_s[x_0]], p_s[x_0], x_0⟩ ⊆ H_↑
22         P_t := ⟨x_0, p_t[x_0], p_t[p_t[x_0]], ..., t⟩ ⊆ H_↓
23         if concatenated up-down-path P_sP_t not represents P then return false
24         if there is X ∈ {s, t}, w ∈ P_X s.t. ¬unique_X[w] then return false
25         return true
26     return false
```

Theorem 9. *A path $\langle s, \dots, t \rangle \subseteq G$ is uniquely representable by a shortest up-down-path in H, if and only if Algorithm 4 returns true.*

Proof. First note that both forward and backward search settle the top node of every shortest up-down-path from s to t adding it to C (Line 10). So, in the end C contains the top nodes of all shortest up-down-paths from s to t and $\min\{d_s[x] + d_t[x] \mid x \in C\}$ is the respective cost of these shortest up-down-paths.

Now, assume the algorithm returns *false*. Then, only the following reasons are possible: First, there is no up-down-path from s to t in H at all, or there are multiple shortest up-down-paths with different top nodes (Line 19 with the above statement). Second, P_s or P_t is not unique in $H_\uparrow$ or $H_\downarrow$ respectively (Line 24 with Lemma 3), so there are multiple shortest up-down-paths even if they have the same top node. Third, the concatenated up-down-path P_sP_t does not represent P (Line 23) even it is the only shortest up-down-path from s to t in H.

Assume the algorithm returns *true*. We know that all shortest up-down-paths from s to t have the same top node (Line 19). Also, P_s and P_t are unique in

$H_\uparrow$ and $H_\downarrow$ respectively (Line 24 with Lemma 3). So, $P_s P_t$ is the only shortest up-down-path from s to t in H and it represents P (Line 23). □

5 Compression Based on CH

If we instantiate *uniqueShortestPrefix* in the frame algorithm (Algorithm 1) with the binary scheme (Algorithm 3) and *isUniqueShortestPath* in the binary scheme with *isUniquelyRepresentable* (Algorithm 4), we get a very fast *CH-based method* to compute representations with via nodes.

As mentioned before, the resulting representations are no longer in terms of the original road network G but in terms of the CH H which has different properties. Consider a path $P := \langle u_1, \dots, u_n \rangle \subseteq G$ which is not necessarily a shortest path. Let $Q := \langle\!\langle u_{i_1}, \dots, u_{i_k} \rangle\!\rangle$ be a subsequence of P with the property that the subpaths $\langle u_1, \dots, u_{i_1} \rangle$, $\langle u_{i_k}, \dots, u_n \rangle$, and $\langle u_{i_j}, \dots, u_{i_{j+1}} \rangle$ of P with $1 \leq j < k$ are all uniquely representable by up-down-paths in H. Then, we call Q a *CH-based representation of P (with via nodes)*. Note that the original road network G is not enough to reconstruct the path from a CH-based representation with via nodes. Instead, we have to compute unique shortest up-down-paths between the via nodes using bidirectional upward searches in the CH. This is due to Observation 10.

Observation 10. *Let H be a CH derived from G. Then, a not unique shortest path $P \subseteq G$ may still be uniquely representable by an up-down-path in H.*

To understand that take a look at Fig. 1. There, the CH contains exactly one shortest up-down-path from v_1 to v_5, namely $\langle v_1, v_5 \rangle$ which represents the shortest path $\langle v_1, v_2, v_4, v_5 \rangle$ in the original road network. However, this shortest path is not unique as the original road network also contains another shortest path from v_1 to v_5, namely $\langle v_1, v_3, v_4, v_5 \rangle$.

As a consequence of Observation 10 less via nodes may be needed by a representation in terms of CH than in terms of the original road network. Again, look at Fig. 1. The minimal representation with via nodes of the path $\langle v_1, v_2, v_4, v_5 \rangle$ in terms of the original road network is $\langle\!\langle v_2 \rangle\!\rangle$. The minimal CH-based representation is $\langle\!\langle \rangle\!\rangle$.

All subpaths of unique shortest paths in G are unique shortest paths themselves. In case of CHs, however, the analogous condition does not hold. Again, Fig. 1 shows an example: The path $\langle v_1, v_2, v_4, v_5 \rangle$ in the original network is uniquely representable by an up-down-path in H but its subpath $\langle v_1, v_2, v_4 \rangle$ is not uniquely representable as there are two up-down-paths from v_1 to v_4.

Observation 11. *Let H be a CH derived from G. Then, a shortest path $P \subseteq G$ may be uniquely representable in H, but one of its subpaths may be not.*

It is because of Observation 11 that the CH-based method does not necessary yield the minimal possible number of via nodes with respect to H. However, we are never worse than the minimal representation in terms of the original road network G. This is due to Lemma 12 as we show in the proof of Theorem 13.

Lemma 12. *Let H be a CH derived from G. Then, every unique shortest path $P \subseteq G$ is uniquely representable by an up-down-path in H.*[3]

Theorem 13. *A CH-based representation computed by our binary CH-based method needs not more via nodes than a minimal representation with respect to the original road network.*

Proof. Let $P = \langle s, \dots, t \rangle \subseteq G$ be the given path. Let $Q_G := \langle\langle u_1, \dots, u_k \rangle\rangle$ be a minimal representation of P with respect to G. Assume our binary method finds a CH-based representation $Q_H := \langle\langle v_1, \dots, v_\ell \rangle\rangle$ with respect to H such that $\ell > k$. Then, we know that i, j exist such that the subpath $\langle v_j, \dots, v_{j+1} \rangle \subseteq P$ is also a subpath of $R := \langle u_i, \dots, u_{i+1} \rangle$ with $v_{j+1} \neq u_{i+1}$.[4] But all subpaths $\langle v_j, \dots, w \rangle \subseteq R$ are unique shortest paths in G and thus, by Lemma 12, uniquely representable by a shortest up-down-path in H. So, the binary scheme (Algorithm 3) instantiated with Algorithm 4 does not return $\langle v_j, \dots, v_{j+1} \rangle$ as resulting prefix path but a longer one – a contradiction. □

6 Experiments

Setup. As input we use a German road network provided by PTV AG for scientific use. It has 4.7 M nodes, 10.8 M edges, and 7.2 % time-dependent edge weights reflecting the travel times of midweek (Tuesday till Thursday) traffic collected from historical data – that is a high traffic scenario. For all edges (u, v) also the *driving distance* $dd(u, v)$ is available. The units of time and distance are 0.1 s and 1 m respectively. From this we obtain four different *metrics*, that is edge weights and objective functions defining different kinds of optimal routes. With these metrics we simulate the hybrid route planning scenario, where server-provided routes are not necessary optimal with respect the client's objective function.

In the *time-dependent* metric edge weights are time-dependent travel times. Optimal routes minimize the travel time depending on the departure time [2–4, 6]. In the *free flow* metric we also minimize travel time but there is no time-dependency. As weight of an edge (u, v) we use the minimum travel time $mtt(u, v)$ of the respective time-dependent edge weight. In the *distance* metric we simply use the driving distance $dd(u, v)$ as weight of an edge (u, v). Optimal routes are minimum distance routes. With the *energy* metric we optimize an approximation of energy consumption. As weight of an edge (u, v) we use $dd(u, v) + 4 \cdot mtt(u, v)$. With typical gasoline prices we assume that driving 1 km costs 0.1 €. This implies that travel time is prized with a rate of 14.4 € per hour.

To simulate the server, we compute optimal routes with respect to the metrics *time-dependent* and *distance*. To simulate possible objective functions of the client, we use the metrics *free flow*, *distance*, and *energy*. This leads to five combinations of *server metrics* and *client metrics*. For all three client metrics the road network contains edges which are not unique shortest paths. This means

[3] A proof can be found in Appendix A.
[4] This can be shown by induction over k, see Appendix A.

Table 1. Behavior of the Dijkstra-based and the binary CH-based compression for all five combinations of server and client metrics. Algor.= numbers of combined algorithms as used in this work, max.= maximum, rate= compression rate. All figures except for the maxima are average values.

# route nodes	client metric	method	Algor.	via nodes #	via nodes max.	via nodes rate[%]	time [ms]
server metric: time-dependent							
996	free flow	Dijkstra-based	1+2	0.071	3	0.006	1 500.78
		binary CH-based	1+3+4	0.068	3	0.006	0.36
	distance	Dijkstra-based	1+2	9.771	26	1.045	481.60
		binary CH-based	1+3+4	9.677	25	1.036	20.98
	energy	Dijkstra-based	1+2	1.103	6	0.125	1 326.17
		binary CH-based	1+3+4	1.094	6	0.124	1.72
server metric: distance							
1 763	free flow	Dijkstra-based	1+2	29.312	76	1.689	162.49
		binary CH-based	1+3+4	29.284	76	1.688	12.56
	energy	Dijkstra-based	1+2	24.902	69	1.434	182.87
		binary CH-based	1+3+4	24.876	69	1.433	15.63

we have to transform the road network a little as described in Sect. 2. For the client metrics *free flow*, *distance*, and *energy* this increases the number of nodes by 2.37 %, 1.48 %, and 2.16 % respectively. The reported average numbers of nodes of the uncompressed paths refers to the non-transformed network.

The experimental evaluation was done on different 64 bit machines with Ubuntu Linux 10.04. The running times have been measured on a machine with 8 GiB main memory and a Core i5 Double-Core CPU at 3.33 GHz. There, all programs were compiled using GCC 4.4.3 with optimization level 3. We evaluate the performance of our compression algorithms in terms of running time, number of via nodes, and compression rate. The compression rate is defined as the number of via nodes divided by the number of nodes of the uncompressed path.

Results. To generate "server provided" routes, we randomly select 1 000 pairs of start and destination nodes computing the optimal routes with respect to both server metrics. For the metric *time-dependent* we also select random departure times from $[0, 24\text{h})$. Table 1 shows the resulting performance of the Dijkstra-based and the binary CH-based compression. The average compression rate is never worse than 1.7 % which means that 29 via nodes are needed to represent a path with 1 763 nodes. The maximum number of via nodes is 76. The number of via nodes gets larger if server and client metrics are less correlated. The compression rate achieved by the CH-based method is only sightly better than for the Dijkstra-based method. However, the minimum number of via nodes possible with CH is unknown. And unfortunately our implementation of Algorithm 4 is a bit pessimistic and potentially rejects some uniquely representable subpaths.

The binary CH-based method runs much faster than the Dijkstra-based one. With average compression times below 21 ms it is fast enough for high throughput servers. Previous experiments with mobile CH [15] suggest that a client needs

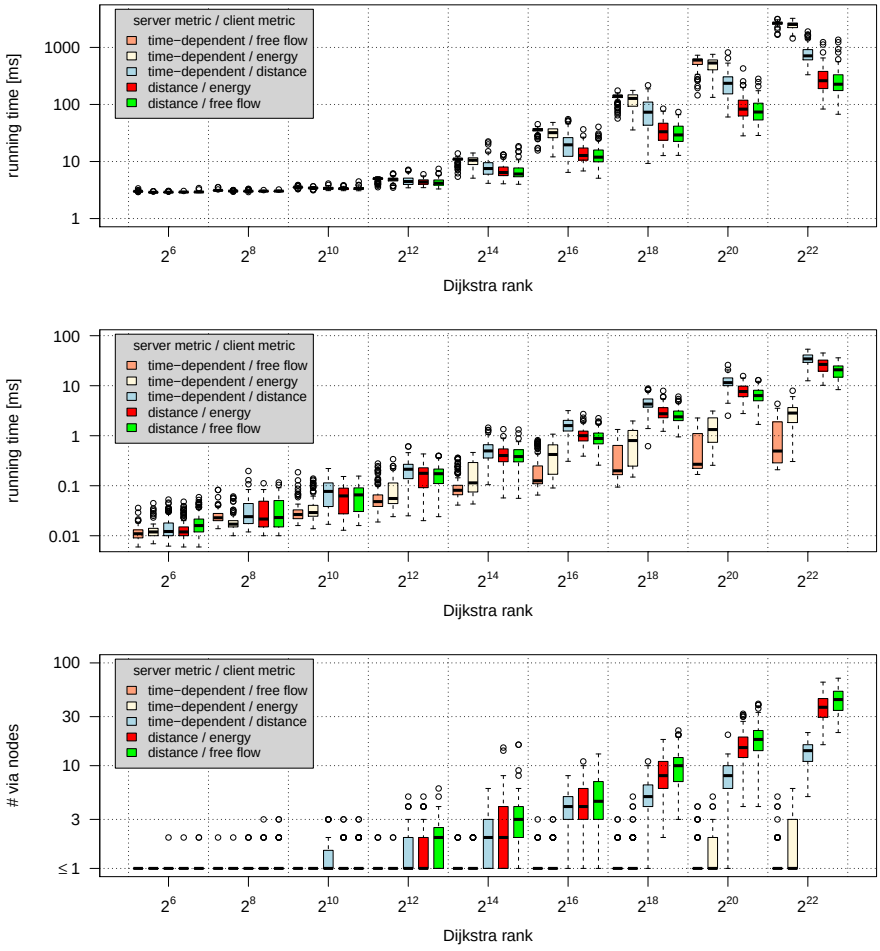

Fig. 2. The compression time of the Dijkstra-based (top) and the CH-based method (middle) plotted over the Dijkstra rank for all five combinations of server and client metrics. The number of via nodes computed by the CH-based method is also plotted (bottom). There are 100 compressed routes per rank and combination.

clearly less than 0.1 s to decompress each segment of a compressed route. So, for our German road network compression and decompression should not raise any noticeable latency – remember that it is enough to decompress the *next* segment fast. It is not surprising that the CH-based method runs faster if the number of via nodes is very small as *isUniqueShortestPath* is invoked $O(|Q| \log |P|)$ times. It looks surprising, however, that the compression runs faster with 29.3 than with 9.7 nodes. A possible explanation is that different metrics entail different distributions of via nodes as well as different numbers of shortcuts in the CH. Both can influence the compression time of a single segment.

The running time of the Dijkstra-based method behaves contrary to the CH-based one: It runs faster the more via nodes we need. Whenever Algorithm 2 finds a maximum unique shortest prefix, it can be stopped (Line 18). But Dijkstra's algorithm has roughly quadratic running time on road networks. So, if we stop it more early but invoke it more often, the overall running time decreases.

Fig. 2 shows the compression time of both methods as well as the number of via nodes plotted over Dijkstra rank[5]. For all combinations of server and client metrics the compression time of both methods as well as the number of via nodes increases with the Dijkstra rank.

7 Conclusions and Future Work

We describe an algorithmic framework for convenient hybrid route planning. Routes computed by servers can be transmitted to client devices in cars efficiently, even when the bandwidth is low. To do so routes are represented as sequences of only a few via nodes. These are the connection points when routes are decomposed into unique shortest subpaths with respect to the clients objective function. Utilizing CH we achieve very good performance: On a German road network an average compression takes less than 21 ms and yields less than 30 via nodes. The maximum number of via nodes we observe is 76. Using mobile CH the client can decompress the first subpath of the route in less than 0.1 s as previous experiments suggest [15]. The following subpaths can be decompressed one after another during driving. So, except for the time needed by the mobile communication the driver will most likely not experience any latency. Note that the low number of via nodes also helps to keep the communication time small.

We also describe a Dijkstra-based method that runs much slower than the CH-based one. But applying Arc-Flags [19] or ALT [20], two algorithmic techniques for fast and exact route planning, may bring a substantial speedup there. An interesting question is, whether subpaths that are uniquely representable with CH can be computed "directly", that is without repeated bidirectional Dijkstra searches. This could further speedup the CH-based compression. Finally, it should be noted that we could also use via *edges* instead of via nodes.

References

1. Delling, D., Wagner, D.: Time-Dependent Route Planning. In Ahuja, R.K., Möhring, R.H., Zaroliagis, C., eds.: Robust and Online Large-Scale Optimization. Volume 5868 of Lecture Notes in Computer Science. Springer (2009) 207–230
2. Batz, G.V., Delling, D., Sanders, P., Vetter, C.: Time-Dependent Contraction Hierarchies. In: Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09), SIAM (April 2009) 97–105
3. Batz, G.V., Geisberger, R., Neubauer, S., Sanders, P.: Time-Dependent Contraction Hierarchies and Approximation. [21] 166–177

[5] For $i = 6..22$ we each select 100 random queries such that the time-dependent variant of Dijkstra's algorithm settles 2^i nodes. We call 2^i the *Dijkstra rank*.

4. Kieritz, T., Luxen, D., Sanders, P., Vetter, C.: Distributed Time-Dependent Contraction Hierarchies. [21] 83–93
5. Batz, G.V., Sanders, P.: Time-Dependent Route Planning with Generalized Objective Functions. In Epstein, L., Ferragina, P., eds.: Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12). Volume 7501 of Lecture Notes in Computer Science., Springer (2012)
6. Delling, D.: Time-Dependent SHARC-Routing. Algorithmica **60**(1) (May 2011) 60–94 Special Issue: European Symposium on Algorithms 2008.
7. Brunel, E., Delling, D., Gemsa, A., Wagner, D.: Space-Efficient SHARC-Routing. [21] 47–58
8. Geisberger, R., Kobitzsch, M., Sanders, P.: Route Planning with Flexible Objective Functions. In: Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10), SIAM (2010) 124–137
9. Delling, D., Wagner, D.: Pareto Paths with SHARC. In Vahrenhold, J., ed.: Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09). Volume 5526 of Lecture Notes in Computer Science., Springer (June 2009) 125–136
10. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Customizable Route Planning. In Pardalos, P.M., Rebennack, S., eds.: Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11). Volume 6630 of Lecture Notes in Computer Science., Springer (2011) 376–387
11. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Alternative Routes in Road Networks. [21] 23–34
12. Luxen, D., Schieferdecker, D.: Candidate Sets for Alternative Routes in Road Networks. In: Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'12). Volume 7276 of Lecture Notes in Computer Science., Springer (2012)
13. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In McGeoch, C.C., ed.: Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08). Volume 5038 of Lecture Notes in Computer Science., Springer (June 2008) 319–333
14. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact Routing in Large Road Networks Using Contraction Hierarchies. Transportation Science (2012) Accepted for publication.
15. Sanders, P., Schultes, D., Vetter, C.: Mobile Route Planning. In: Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08). Volume 5193 of Lecture Notes in Computer Science., Springer (September 2008) 732–743
16. Batz, G.V., Geisberger, R., Luxen, D., Sanders, P.: Compressed Transmission of Route Descriptions. Technical report, Karlsruhe Institute of Technology (KIT) (2010) arXiv:1011.4465v1.
17. Tao, Y., Sheng, C., Pei, J.: On k-skip shortest paths. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. SIGMOD '11, New York, NY, USA, ACM (2011) 421–432
18. Mehlhorn, K., Sanders, P.: Algorithms and Data Structures: The Basic Toolbox. Springer (2008)
19. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speed Up Dijkstra's Algorithm. In: Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05). Volume 3503 of Lecture Notes in Computer Science., Springer (2005) 189–202

20. 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'05), SIAM (2005) 156–165
21. Festa, P., ed.: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10). In Festa, P., ed.: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10). Volume 6049 of Lecture Notes in Computer Science., Springer (May 2010)

A Some Details Omitted in the Original Version

In the original version of this paper, which is available at www.springerlink.com, we omitted some details due to page limit. These are the proof of Lemma 12 and a statement used in the proof of Theorem 13.

Lemma 12. *Let H be a CH derived from G. Then, every unique shortest path $P \subseteq G$ is uniquely representable by an up-down-path in H.*

Proof. According to Lemma 8 there is a shortest path $P' = \langle s, \dots, t \rangle \subseteq G$ that is represented by some up-down-path in H. But as P is the only shortest path from s to t we have $P = P'$. □

In the proof of Theorem 13 we used the following statement without proof just saying in a footnote that the statement can be shown by induction.

Lemma. *Given a path $P = \langle s, \dots, t \rangle \subseteq G$ as well as two subsequences $Q = \langle\!\langle u_1, \dots, u_k \rangle\!\rangle$ and $Q' = \langle\!\langle v_1, \dots, v_\ell \rangle\!\rangle$ of P excluding s, t with $\ell > k$. Then, with $u_0 := v_0 := s$ and $u_{k+1} := t$, there must be i, j such that the subpath $\langle v_j, \dots, v_{j+1} \rangle$ of P is also a subpath of $\langle u_i, \dots, u_{i+1} \rangle$ with $v_{j+1} \neq u_{i+1}$.*

Proof. For the base case $k = 0$ (that is $Q = \langle\!\langle \rangle\!\rangle$) this is obvious. For $k > 0$ we apply induction and assume that the statement holds true for $k-1$. There are three possible cases: (1) $\langle s, \dots, v_\ell \rangle$ is a real prefix of subpath $P_k := \langle s, \dots, u_k \rangle$. Then the statement follows directly from the induction hypothesis as $\langle\!\langle u_1, \dots, u_{k-1} \rangle\!\rangle$ is a subsequence of P_k of size $k-1$ and Q' is also a subsequence of P_k but with size $\ell > k > k-1$. (2) $\langle s, \dots, v_{\ell-1} \rangle$ is a real prefix of P_k but $\langle s, \dots, v_\ell \rangle$ is not. Then, $\langle\!\langle v_1, \dots, v_{\ell-1} \rangle\!\rangle$ is subsequence of P_k of size $\ell - 1 > k - 1$ and again the statement follows from the induction hypothesis. (3) $\langle s, \dots, v_{\ell-h} \rangle$ with $h > 1$ is a real prefix of P_k but $\langle s, \dots, v_{\ell-h+1} \rangle$ is not. Then, we have the subsequences $\langle\!\langle \rangle\!\rangle$ and $\langle\!\langle v_{\ell-h+1}, \dots, v_\ell \rangle\!\rangle$ of the subpath $\langle u_k, \dots, t \rangle$. But this is the base case and we are finished. □