Light Contraction Hierarchies: Hierarchical Search Without Shortcuts

Claudius Proissl
Universität Stuttgart
claudius.proissl@fmi.uni-stuttgart.de

Abstract

Hierarchical search such as Contraction Hierarchies is a popular and successful branch of optimization techniques for shortest path computation. Existing hierarchical techniques have one component in common: they add edges to the graph, so called shortcuts. This component usually causes a considerable space overhead but is mandatory in order to preserve correctness. In this work we show a hierarchical method that requires to store only two additional bytes per node and no shortcuts at all. We prove the correctness of our method and experimentally show that it improves query times by one order of magnitude compared to Dijkstra's bidirectional algorithm.

Introduction

In this work we address the problem of finding the shortest path from a source node s to a target node t in a directed graph $G(V, E)$ with non-negative edge weights $c: E \to \mathbb{R}_+$. We define a path $p = e_1 e_2 \dots e_k$ to be a sequence of edges such that the target node of edge e_i is the source node of edge e_{i+1} for every $1 \leq i < k - 1$. A node v is in path p if v is adjacent to an edge in p. Furthermore, we say that p goes from node s to node t if $e_1 = (s, \cdot)$ and $e_k = (\cdot, t)$. The weight $c(p)$ of a path p is defined to be the sum of the weights of its edges. The path $\pi(s, t)$ from node s to node t with minimum weight is called shortest st-path and its weight is the distance from s to t, also written $d(s, t)$. A basic, yet versatile approach to finding π is Dijkstra's algorithm, which performs a greedy search based on the distances from the source node s. There is also a bidirectional variant that starts a search from s and from t. The search from node t traverses the edges backwards, hence we call it a backward search (in contrast to the forward search from node s). In many large-scale applications, though, Dijkstra's algorithm is too time consuming to be practical. Hence, many speed-up techniques have been proposed to improve the shortest path query times. A very successful approach is to identify whether the shortest paths exhibit a hierarchical structure during a preprocessing step and to perform a search that takes this structure into account.

Important contributions to hierarchical search are, for instance, Highway Hierarchies (Sanders and Schultes 2006), Highway Node Routing (Schultes and Sanders 2007) and Contraction Hierarchies (Geisberger et al. 2008). All three techniques have in common that they are in particular designed for street networks. This is also the point of view we take in our experiments. In the following, we discuss Contraction Hierarchies (CH) in more detail as they build the basis for our method.

In CH, a (bijective) node ordering $r : V \to \{1, 2, \dots, n\}$ is defined, where n is the number of nodes in the graph. Given a node $v \in V$ the mapping $r(v)$ is called the rank of v. The greater the rank, the more important the node is supposed to be. With the ranks one can divide the edges into upward edges and downward edges. An edge $e = (u, v)$ goes upwards if $r(u) < r(v)$ and downwards otherwise. An upward (downward) path is a path that only consist of upward (downward) edges. If this path is additionally a shortest path we say that it is a shortest upward (downward) path.

Let $V(R)$ be the subset of V of all nodes with rank greater than R. We say that a graph $G(V, E)$ is unimodal with respect to the node ordering r if for any value R the subgraph of G induced by $V(R)$ shows the same distances between any two nodes in $V(R)$ as in G itself. In other words, if we computed the distance from any node $s \in V(R)$ to any node $t \in V(R)$ first in G and then in the subgraph of G induced by $V(R)$ we would always get the same result (given that G was unimodal). Most graphs are not unimodal as shortest paths in G between nodes in $V(R)$ typically also contain nodes in $V \setminus V(R)$. The preprocessing step of CH takes a graph $G(V, E)$, defines a node ordering r and computes a set of shortcuts E' of minimum cardinality such that $G'(V, E \cup E')$ is unimodal with respect to r and all distances in G and G' are identical. The name Contraction Hierarchies comes from the way the preprocessing step finds the set E' by performing node contraction operations. Nodes of low rank are contracted first. These contractions are typically performed in a round-based manner. In each round, a set of independent nodes is contracted simultaneously (see, for instance, (Geisberger et al. 2012) for further details). If a graph G is unimodal, one can show that between any two nodes s and t there is a shortest path $\pi := (s, v_2)(v_2, v_3) \dots (u, w) \dots (v_{k-1}, t)$ that is the concatenation of an upward and downward path (Geisberger et al. 2008). Thus, we find a node w of greatest rank in π

such that $r(s) < r(v_2) < r(v_3) < \cdots < r(w) > \cdots > r(v_{k-1}) > r(t)$. We call such a path π a shortest up-down path. Hence, we can modify Dijkstra's bidirectional search to prune all paths that are not up-down paths. We do so by only expanding upward edges in the forward direction and downward edges in the backward direction. In the context of minimizing travel time in street networks, CH has proven to be a very effective speed-up technique with modest space overhead (Geisberger et al. 2012).

Another important hierarchical approach to compute shortest paths is Hierarchical Hub Labeling (HHL) (Abraham et al. 2012), which is based on the work of (Cohen et al. 2003). Although unconventional, HHL can be considered as a CH with more shortcuts such that the bidirectional search has to expand only the edges of the source node and of the target node. In other words, the resulting up-down path π consists of at most two edges. A generalization of the idea to restrict the number of edges of the up-down paths in CH to a certain value can be found in (Bahrdt et al. 2022). HHL further improves the query times compared to CH but typically requires to store many more shortcuts.

All mentioned hierarchical approaches heavily rely on the idea of adding shortcuts to the graph. There is also no doubt that shortcuts greatly help to improve query times. However, these algorithms do not only use shortcuts to improve query times but require them in order to ensure correctness. Hence, it is worth raising the question whether there is an efficient way to perform hierarchical search without using any shortcuts. This work is about answering this question with yes.

Our Contribution

We propose a new hierarchical approach to compute shortest paths based on CH that does not require any shortcuts during the query phase. In fact, it suffices to store two additional bytes per node and, therefore, the space overhead is significantly reduced compared to existing hierarchical approaches. Furthermore, our method is very simple to implement as soon as a way to construct CH is available. Our method is able to improve Dijkstra's bidirectional algorithm in terms of query times by one order of magnitude.

Related Work

We are aware of merely one hierarchical search technique that does not require shortcuts. It is called REACH (Gutman 2004), an early contribution to hierarchical search. It stores one number per node that describes the importance of the node regarding long-distance journeys. This number is called *reach*. In the following, we describe how this number is defined given a graph $G(V,\, E)$. Let $\pi(s,\, t)$ be the shortest st-path in G and let u be any node in π. The reach of u regarding π is then defined as

$$re\,(u,\, \pi) := \min\left\{d\,(s,\, u)\,,\, d\,(u,\, t)\right\}.$$

The (global) reach of u, written as $re\,(u)$, is then simply the maximum reach over all shortest paths that contain u. This number is computed and stored for each node in V during the preprocessing step. In the query phase a modified version of Dijkstra's bidirectional algorithm is performed that prunes all nodes w for which it is already known from the preliminary search results that $re(w) < \min\left\{d\,(s,\, w)\,,\, d\,(w,\, t)\right\}$. The results stay correct because if $re(w) < \min\left\{d\,(s,\, w)\,,\, d\,(w,\, t)\right\}$ is true, then w is certainly not part of the shortest st-path. The main drawback of REACH is the complexity of its preprocessing step, which requires an all-to-all shortest path computation. This is not feasible for many applications. It is possible to reduce the computational effort in the preprocessing step by computing upper bounds of the reach values as shown in (Gutman 2004). While the idea behind REACH is similar to ours, our method is simpler, requires less additional space and can be implemented in a straightforward manner as soon as a way to construct CH is available. Our method therefore inherits the efficient preprocessing step of CH but requires less space than CH during the query phase.

A survey that still covers most important speed-up techniques in transportation networks can be found in (Bast et al. 2016).

Light Contraction Hierarchies

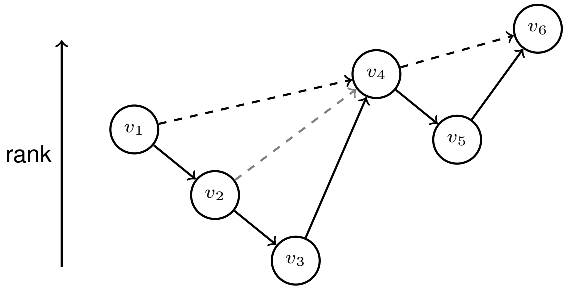
In this section we describe our method to compute shortest paths and prove its correctness. As it essentially takes a CH and removes its shortcuts we call the method Light Contraction Hierarchies or LCH. Let $G(V,\, E)$ be our street network and $G'(V,\, E \cup E')$, $r : V \to \{1,\, 2,\, \ldots,\, n\}$ the corresponding output of the CH preprocessing step as described in the introduction. Since all distances between nodes in G and G' are identical, for each shortest path $\pi'\,(s,\, t)$ in G' we find a shortest path $\pi\,(s,\, t)$ in G such that each node $w \in \pi'$ is also contained in π. Let Π be the function that maps shortest paths in G' to their corresponding shortest path in G. We say that Π unpacks the paths in G' to their original form in G. Note that each shortcut $e' := (u,\, v) \in E'$ is a shortest path itself. Let $\pi\,(u,\, v) \in \Pi\,(e')$ be the unpacked path of e', what ranks do the nodes in π have? Can it happen that there is a node $w \in \pi$ with greater rank than u and v? Lemma 1 answers this question with no, which is going to be important for our further considerations.

Lemma 1. *For every shortcut $e' := (u,\, v) \in E'$ and every node $w \in \Pi\,(e')$ with $w \notin \{u,\, v\}$ it holds $r\,(w) < \min\{r\,(u)\,,\, r\,(v)\}$.*

Proof. We assume that $r\,(u) < r\,(v)$. The other case can be shown analogously. Let path $\pi := \Pi\,(e')$ and suppose that we find a node $w \in \pi$ with $r\,(w) > r\,(u)$ and $w \neq v$. The edge set E' is, by definition, of minimum cardinality such that G' is unimodal with respect to the node ordering r. Thus, if we remove edge e' from G', the distance from u to v in the subgraph of G' induced by $V\,(r\,(u) - 1)$ (as defined in the introduction) must be greater than $d\,(u,\, v)$. However, we have $d\,(u,\, w) + d\,(w,\, v) = d\,(u,\, v)$, which is a contradiction. Thus, such a node w cannot exist. □

We define the rank $r_E\,(e)$ of an edge or shortcut e as follows.

$$\begin{aligned} &r_E : E \cup E' \to \{1,\, 2,\, \ldots,\, n\}, \\ &r_E\,((u,\, v)) = \min\left\{r\,(u)\,,\, r\,(v)\right\} \end{aligned}$$



(a) Example CH G' with dashed shortcuts. The path $\pi' = v_1 v_4 v_6$ in G' consisting of two shortcuts (dashed black arrows). The corresponding mimicked path in the original graph G (without shortcuts) is $\pi = v_1 v_2 v_3 v_4 v_5 v_6$.

	v_1	v_2	v_3	v_4	v_5	v_6
$r(v)$	4	2	1	5	3	6
$r_{max}(v)$	4	4	4	5	5	6

(b) The mappings r and r_{max} of the example CH G'. The node v_3, for instance, is part of two shortcuts (v_1, v_4) and (v_2, v_4). Therefore, $r_{max}(v_3) = \max\{r(v_3), \min\{r(v_2), r(v_4)\}, \min\{r(v_1), r(v_4)\}\} = 4$.

Figure 1: An example CH to demonstrate the mappings r and r_{max} and what it means to mimic a path.

The overall idea of our method is to save for each node u the rank of the most important shortcut e' for which $u \in \Pi(e')$ holds. We call this new mapping r_{max}. Hence, if $r_{max}(u)$ is small for a node u, it means that u is not part of any important shortcut and we may be able to ignore u during our bidirectional search. We formally define r_{max} as follows.

$$r_{max} : V \to \{1, 2, \ldots, n\},$$
$$r_{max}(u) = \max\{r(u), \max_{\{e' \in E' : u \in \Pi(e')\}} r_E(e')\}$$

Furthermore, let $r_P(\pi) := \max_{u \in \pi} r(u)$ for a path π. Our method tries to mimic the CH query algorithm in G' without having access to the shortcuts E'. That means, if the upward search G' expands an edge $e \in E$, our method expands this edge as well. And if the upward search in G' expands a shortcut $e' \in E'$, our method follows along the unpacked path $\Pi(e')$. See Figure 1 for an example. However, since E' is unavailable, it is unclear how to find the path $\Pi(e')$. In the following lemma we show how the mappings r and r_{max} can help with this task.

Lemma 2. *Given a shortest upward (or downward) path $\pi'(s, t)$ in G'. For every node $w \in \Pi(\pi')$ it holds $r_{max}(w) \geq r_P(\pi(s, w))$, where π is the subpath of $\Pi(\pi')$ in G that starts in s and ends in w.*

Proof. Let $\pi'(s, t)$ be a shortest upward path in G' and let node w and path $\pi(s, w)$ be defined as above. Clearly, there is an edge $e := (u, v)$ in π' with $w \in \Pi(e)$. Thus, $r_{max}(w) \geq r_E(e) = r(u)$. As π' is an upward path, it follows from Lemma 1 that $r_P(\pi) = r(u)$. The case when π' is a shortest downward path can be shown analogously. □

Lemma 2 says that whenever we find a shortest path $\pi(s, w)$ during our search with $r_P(\pi) > r_{max}(w)$ we know that π cannot be a subpath of an unpacked shortest upward (or downward) path in G'. Therefore and because we aim to mimic the upward search in G' we do not need to continue our search at w.

We are now ready to describe our hierarchical search, which is a modification of Dijkstra's bidirectional algorithm similar to the query algorithm of CH. We only describe the forward search as the backward search is analogous with flipped edges. Our search performs the same operations as Dijkstra's algorithm but additionally keeps track of $r_P(\pi)$ for each search branch π. Furthermore, the method only expands an edge $e = (u, v)$ if the condition $r_P(\pi(s, u)) \leq r_{max}(v)$ is satisfied. If this is not the case, it follows from Lemma 2 that we do not need to settle node v.

The mappings r and r_{max} cause the only space overhead of LCH. In theory, a rank is a number between one and n, where n is the number of nodes in G. Fortunately, the rank can be replaced by the contraction round in which the node was contracted. In this way, many nodes share the same rank and it suffices to spend one byte per rank even for large graphs without any negative impact on the correctness and the query performance. If the CH happens to have more than 256 contraction rounds, we map all nodes contracted after round 255 to rank 255. This does not affect correctness but may slightly slow down the queries. Similar rank plateaus are, for instance, proposed in (Funke, Laue, and Storandt 2017). Hence, in summary we spend two additional bytes per node. This space overhead is compensated by an efficient way to store the edges as we show and discuss in the following section.

	Germany	South America
Nodes	$25,115,477$	$62,562,908$
Edges	$50,774,067$	$129,634,220$
Shortcuts of CH	$40,956,462$	$122,964,150$

Table 1: Details about the size of the street networks that were subject of our experiments.

Experiments

We tested our method on the road network of Germany and of South America, which we both obtained from OpenStreetMap (OpenStreetMap contributors 2018). We conducted all experiments on an AMD Threadripper 2950X with 256 RAM and Ubuntu 20.04 as operating system. Our implementations of CH, Dijkstra's bidirectional algorithm and LCH are written in C++ (using g++, version 9.3.0) and are publicly available[1]. An experiment consisted of one thousand source-target shortest path queries, selected uniformly at random. We built our CH using a standard approach described in (Geisberger et al. 2012), which always contracts independent sets of nodes with low edge difference. Our CH queries conduct the common speed-up technique *stall-on-demand* (Geisberger et al. 2008).

The construction of the LCH that takes a CH as input, computes the mapping r_{max} and removes the shortcuts took

[1] https://github.com/proisscs/chlight

	Germany	South America
BiDijk	$3,024$	$5,693$
LCH	347	349
CH	2	9

Table 2: Average shortest path query times in milliseconds.

	Germany	South America
$\|V_{BiDijk}\|$	$7,578,599$	$21,384,160$
$\|V_{LCH}\|$	$973,928$	$1,232,816$
$\|V_G^{\wedge}\|$	$405,612$	$598,501$
$\|V^{\wedge}\|$	898	984

Table 3: Comparison of the average number of visited nodes in the forward and backward search of BiDijk ($|V_{BiDijk}|$), LCH ($|V_{LCH}|$), CH ($|V^{\wedge}|$) and the average number of nodes that are part of edges/shortcuts expanded during CH queries ($|V_G^{\wedge}|$). The size of $V_G^{\wedge}$ is a lower bound of the size of V_{LCH}.

68 seconds for Germany and 166 seconds for South America (both conducted on a single core). The input CH can be constructed within a couple of minutes in both cases. The CH of Germany was constructed within 303 and the CH of South America within 332 contraction rounds. In both cases less than 120 nodes were contracted after round 255.

Table 2 shows the average query times of Dijkstra's bidirectional algorithm (BiDijk), LCH and CH. Note that we do not compare LCH to REACH as the preprocessing of REACH would be too time consuming for the chosen graphs. First, we can observe that LCH is able to improve the query times compared to BiDijk considerably for both street networks. However, the factor of improvement is smaller for Germany (8.71) than for South America (16.31). The average query time of LCH for South America is almost equal to that of Germany despite South America being the larger street network (see Table 1). We see the opposite when comparing the query times of BiDijk and CH. Here, the factor of improvement is $1,414$ for Germany and 622 for South America. This result indicates that optimizing the query times of CH and of LCH (by choosing appropriate node orderings) are two different objectives. Note that if it is not necessary to have a complete representation of the shortest path, the CH queries can be considerably faster.

We now address the question whether our method is able to mimic the CH upward search efficiently. Let V_{BiDijk} and V_{LCH} be the set of nodes visited during the BiDijk query and LCH query, respectively. Furthermore, let $V_G^{\wedge}$ be the set of nodes that are adjacent to edges or part of unpacked shortcuts expanded during the CH query. If our method was able to mimic the CH search perfectly, the two sets $V_G^{\wedge}$ and V_{LCH} would be equal. Thus, the size of $V_G^{\wedge}$ is a lower bound of the size of V_{LCH}. We examine the efficiency of our method by comparing these two numbers. Table 3 shows the results. In both cases, our method is surprisingly close to the lower bound and visits significantly less nodes than BiDijk. The factor to the lower bound is 2.4 for Germany and 2.1 for South America. This shows that the mapping r_{max} is a good indicator whether a node is part of $V_G^{\wedge}$.

	Germany	South America
BiDijk	0.94 GB	2.40 GB
LCH	0.91 GB	2.41 GB
CH	1.21 GB	3.27 GB

Table 4: Comparison of space requirements.

Finally, we look at the actual space requirements of BiDijk, LCH and CH. The edges and shortcuts of our CH are stored in separate arrays to save space as the shortcuts additionally need to contain information about how to unpack them. This could either be two pointers to their child shortcuts/edges or one pointer to the midnode of the shortcut (see (Geisberger et al. 2008) and (Geisberger et al. 2012) for details). The first option allows to unpack the shortcut faster while the second is space conservative. We decided to implement the second option to have a fair comparison. Table 4 shows the results. We observe that the space overhead of LCH compared to BiDijk is almost non-existent (South America) or even negative (Germany). LCH does save a considerable amount of space compared to CH but probably not as much as one might have expected. The reason is that for BiDijk it is necessary to store each edge (u, v) twice, once as forward edge for u and once as backward edge for v, to allow an efficient bidirectional search. CH have the great advantage that it suffices to store each edge only once despite conducting bidirectional search. This is because in CH each edge is either an upward edge or a downward edge (as explained in the introduction). The forward search only expands upward edges and the backward search only expands downward edges. In the case of LCH, an edge $e = (u, v)$ is an upward edge if $r_{max}(u) < r(v)$. Analogously, the edge e is a downward edge if $r(u) > r_{max}(v)$. These edges need to be stored only once. This is the reason why LCH can achieve a negative space overhead compared to BiDijk. Most edges are neither an upward nor a downward edge, though, and must be stored for both directions.

Conclusions

We presented a method to perform hierarchical search without shortcuts. We call it Light Contraction Hierarchies or LCH as it is based on Contraction Herarchies, one of the most popular hierarchical search techniques. Our method is very simple to implement (provided a way to construct CH is available). We showed that our method, despite its very modest space overhead, is able to improve the average query time of shortest path computations by one order of magnitude compared to Dijkstra's bidirectional algorithm. Furthermore, we were able to show that our method is also efficient with respect to the number of visited nodes.

An interesting open problem is the relationship between the query times of CH and LCH. Our results indicate that finding a node ordering that achieves good CH query times is a different objective than finding such an ordering for LCH. Furthermore, a drawback of our method is that the preprocessing step needs a CH as input. We plan to investigate if it is possible to construct the LCH efficiently without having the complete CH.

References

Abraham, I.; Delling, D.; Goldberg, A. V.; and Werneck, R. F. 2012. Hierarchical hub labelings for shortest paths. In *European Symposium on Algorithms*, 24–35. Springer.

Bahrdt, D.; Funke, S.; Makolli, S.; and Proissl, C. 2022. Distance Closures: Unifying Search- and Lookup-based Shortest Path Speedup Techniques. In *2022 Proceedings of the Twenty-Fourth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM.

Bast, H.; Delling, D.; Goldberg, A.; Müller-Hannemann, M.; Pajor, T.; Sanders, P.; Wagner, D.; and Werneck, R. F. 2016. Route planning in transportation networks. In *Algorithm engineering*, 19–80. Springer.

Cohen, E.; Halperin, E.; Kaplan, H.; and Zwick, U. 2003. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5): 1338–1355.

Funke, S.; Laue, S.; and Storandt, S. 2017. Personal routes with high-dimensional costs and dynamic approximation guarantees. In *16th International Symposium on Experimental Algorithms (SEA 2017)*.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *International Workshop on Experimental and Efficient Algorithms*, 319–333. Springer.

Geisberger, R.; Sanders, P.; Schultes, D.; and Vetter, C. 2012. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3): 388–404.

Gutman, R. J. 2004. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. *ALENEX/ANALC*, 4: 100–111.

OpenStreetMap contributors. 2018. Street Network of South America. https://www.openstreetmap.org. Accessed: 2018-05-20.

Sanders, P.; and Schultes, D. 2006. Engineering highway hierarchies. In *European Symposium on Algorithms*, 804–816. Springer.

Schultes, D.; and Sanders, P. 2007. Dynamic highway-node routing. In *International Workshop on Experimental and Efficient Algorithms*, 66–79. Springer.