Algorithms for Landmark Hub Labeling

Sabine Storandt
Universität Konstanz, Germany

Abstract

Landmark-based routing and Hub Labeling (HL) are shortest path planning techniques, both of which rely on storing shortest path distances between selected pairs of nodes in a preprocessing phase to accelerate query answering. In Landmark-based routing, stored distances to landmark nodes are used to obtain distance lower bounds that guide A* search from node s to node t. With HL, tight upper bounds for shortest path distances between any s-t-pair can be interfered from their stored node labels, making HL an efficient distance oracle. However, for shortest path retrieval, the oracle has to be called once per edge in said path. Furthermore, HL often suffers from a large space consumption as many node pair distances have to be stored in the labels to allow for correct query answering. In this paper, we propose a novel technique, called Landmark Hub Labeling (LHL), which integrates the landmark concept into HL. We prove better worst-case path retrieval times for LHL in case it is path-consistent (a new labeling property we introduce). Moreover, we design efficient (approximation) algorithms that produce path-consistent LHL with small label size and provide parametrized upper bounds, depending on the highway dimension h or the geodesic transversal number gt of the graph. Finally, we show that the space consumption of LHL is smaller than that of (hierarchical) HL, both in theory and in experiments on real-world road networks.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Approximation algorithms analysis

Keywords and phrases Hub Labeling, Landmark, Geodesic, Hitting Set, Highway Dimension

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.5

1 Introduction

There exists a plethora of shortest path planning algorithms, many of which follow the preprocessing paradigm to achieve better running times than Dijkstra's algorithm [8]. Combining orthogonal techniques often leads to further acceleration [11, 26]. However, such combinations usually escape rigorous formal analysis and are only justified empirically. In this paper, we propose and analyze the concept of Landmark Hub Labeling as a symbiosis of two preprocessing-based techniques, Hub Labeling and Landmark-based routing.

Hub Labeling (HL) is an elegant technique for shortest path computation in weighted graphs $G(V, E, c)$ which has been extensively studied both theoretically and empirically [14, 3, 4, 17, 5, 16, 7]. The core idea of HL is to construct labels $L : V \to 2^V$ for all nodes $v \in V$ which have to fulfill the so called *cover property*. The property demands that for any node pair $s, t \in V$, the intersection of their labels $L(s) \cap L(t)$ contains a node w on the shortest path from s to t. Now, if shortest path distances are stored along with the nodes in the label, the shortest path distance $c(s, t)$ from s to t can simply be computed as the sum of the distances from s to w and from w to t. As for all nodes $v \in L(s) \cap L(t)$ we have $c(s, v) + c(v, t) \geq c(s, t)$ and the inequality is tight for at least one node, the correct distance value can be computed purely based on label information. HL exhibits excellent distance query times on a large variety of graphs. For example, on road networks, running times in the microsecond range are achieved (a speed-up of more than six orders of magnitude over the Dijkstra baseline) [4], outperforming its state-of-the-art competitors. The price to pay is the rather high space consumption to store the node labels. On the road network of Western Europe, average label sizes of 69 were obtained (with an elaborate HL construction algorithm), increasing the space consumption from 0.4GB for the original graph to 18GB.

Landmark-based routing requires less space to store preprocessed data and is more flexible in that regard. Here, a small set of nodes is selected to be landmarks and distances from all nodes in the graph to the landmarks are precomputed. Shortest s-t-path queries are answered with ALT (A* + landmarks + triangle inequality) [23] which utilizes the precomputed distances to the landmark nodes to get lower bounds for the distance of a node to the target t. These lower bounds serve as an admissible heuristic for A* search and help to significantly reduce the search space size compared to running Dijkstra's algorithm. For example, on the road network of Western Europe, using 16 landmarks results in a reduction of the query time from about 5 seconds to 50 milliseconds, with 32 landmarks to 30 milliseconds [20].

There is seemingly a huge gap between the query times of ALT and HL but one has to note that ALT always returns the full shortest path while HL is a distance oracle. Shortest paths can also be computed based on HL but the respective algorithm needs to issue one HL distance query per edge in the shortest path. Thus, if the path contains thousands of edges, the gap is less pronounced.

We will show that our new Landmark Hub Labeling scheme exhibits interesting theoretical properties and allows for novel trade-offs between space consumption and query time.

1.1 Related Work

Hub Labeling (HL) was originally proposed by Cohen et al. [14] under the name 2-*hop labeling*. It was proven that minimizing the total label size is NP-hard in [7] and a $\mathcal{O}(\log n)$ approximation algorithm with a running time of $\mathcal{O}(n^5)$ was designed. The running time was later reduced to $\mathcal{O}(n^3 \log n)$ [16]. Furthermore, parameterized upper bounds for label sizes in HL were investigated. In [10], it was shown that search space sizes in a related preprocessing-based shortest path technique, called Contraction Hierarchies, can be upper bounded by $\mathcal{O}(tw \log n)$ where tw denotes the treewidth of the graph. The bounds transfers to label sizes in a HL [4]. In [2], it was proven that labels with sizes in $\mathcal{O}(h \log D)$ exist, where h denotes the highway dimension of the graph. Label sizes in $\mathcal{O}(h \log D \log n)$ can be computed in polytime. Furthermore, the skeleton dimension κ was introduced to analyze HL in [29]. Via a randomized construction algorithm, label sizes in $\mathcal{O}(\kappa \log n)$ were shown to be possible. For practical application, however, usually fast heuristics are used instead of algorithms with provable guarantees [15, 16, 32, 13]. These heuristics typically produce a special type of labeling, called hierarchical Hub Labeling (HHL). While hierarchical labelings might be larger by a factor of $\Omega(\sqrt{n})$ than general HL [7], the respective algorithms were shown to efficiently produce concise labels on real-world networks.

The ALT algorithm was proposed in [23] to improve the performance of A* via landmark-based lower bounds. Subsequent work has mainly focused on how many landmarks are needed and how they should be selected to get strong lower bounds while exhibiting low space consumption [19, 24, 21, 33]. It was proven in [9] that selecting a landmark node set of fixed size is NP-hard when the goal is to minimize the expected number of nodes settled with ALT. In [25], the concept of *active landmarks* was introduced and explored, where during query answering not the distances to all available landmarks are evaluated but only to a promising subset. This reduces the time for lower bound computation and hence accelerates query answering. Landmark-based routing was also studied for dynamic graphs with changing edge weights, either as a standalone method [20, 21, 27] or in combination with other (hierarchical) techniques [18, 30].

We note that also synergies between HL and landmarks as well as innovative labeling schemes were explored before. The Pruned Landmark Labeling (PLL) algorithm [5, 6] first subdivides the input network into shortest paths and then infers node labels that encode distances towards these paths. The PLL approach utilizes landmarks to compute a concise labeling efficiently. Note that despite the similarity in name this is a vastly different approach to the concept we propose in this paper, as PLL uses landmark information only in the preprocessing step and the correctness of query answering relies on tight upper distance bounds just as for classical HL.

In [34], it was observed that stored distances to landmark nodes can not only be used to deduce lower bounds but also upper bounds in a similar fashion as in HL. However, in contrast to labeling approaches, their goal is not necessarily to be able to answer queries exactly purely based on label information. Instead, they either return only a distance estimate (e.g. the geometric mean of upper and lower bound) or fall back on A* if the gap between the bounds is too large. But they also prove that the Landmark Cover problem, in which one has to select the smallest number of landmarks to have tight bounds for all node pairs, is NP-hard. Interestingly, they define the tightness only via the upper bound and hence they rather prove that a variant of HL is NP-hard in which the labels of all nodes need to be the same. They also show that this problem can be approximated within a factor of $\mathcal{O}(\log n)$ in $\mathcal{O}(n^3)$. In our definition of Landmark Hub Labeling, we allow nodes to have individual label sets and the tightness criterion to be fulfilled either via an upper or a lower bound – again individually for each node pair – while still guaranteeing exact query answering.

1.2 Contribution

In this paper, we extend the concept of labeling beyond Hub Labeling (HL), and we design novel query algorithms as well construction schemes with provable guarantees regarding the label sizes. The following are our main contributions:

- We propose two novel labeling schemes, Landmark Labeling (LL) and Landmark Hub Labeling (LHL), which integrate distance lower bounds into the labeling framework. We show that there exist graphs on which the label sizes of LL and LHL are smaller than the label sizes of classical HL by a logarithmic factor.
- We also introduce a new property for general labelings called path-consistent (PC). We show that the PC property allows to answer shortest path queries significantly faster.
- We prove that the maximum label size in a PC-LHL is upper bounded by gt, where gt denotes the geodesic transversal number, a recently introduced graph parameter [31]. We discuss how a PC-LHL with label sizes in $\mathcal{O}(gt \log n)$ can be constructed in $\mathcal{O}(n^3)$.
- We investigate the relationship between gt and the highway dimension h, a graph parameter which was used in previous work to bound label sizes of classical HL [2]. While it turns out that gt and h are incomparable, we show that nevertheless the label sizes of PC-LHL can be also upper bounded in dependency of h.
- We design an approximation algorithm for PC-LHL (as well as PC-LL and PC-HL) that runs in $\mathcal{O}(n^6)$ and guarantees a total label size within a factor of $\mathcal{O}(\log n)$ of the optimum. The algorithm builds on the greedy set cover framework for HL computation introduced in [7] but needs to be significantly modified to be able to deal with the PC property.
- Inspired by hierarchical HL (HHL), we also study properties of hierarchical PC-LHL (HPC-LHL). In a proof-of-concept experimental study on medium sized road networks, we demonstrate that label sizes in HPC-LHL are indeed smaller than those in HHL.

2 Preliminaries

Throughout the paper, we assume to be given a weighted graph $G(V, E, c)$ with positive edge weights $c : E \to \mathbb{R}^+$ and a unique shortest path between any node pair $s, t \in V$. The latter is a common assumption (in particular when dealing with road networks) but it can also be enforced in arbitrary graphs via (symbolic) edge weight perturbation. Furthermore, we assume the input graph to be undirected. Both, HL and ALT can also be applied to directed graphs by storing label information or landmark distances for each direction separately. The same is true for LHL. But for ease of exposition, we restrict our decriptions to the undirected case. We use $c(s,t) = c(t,s)$ to denote the shortest path distance between nodes $s, t \in V$.

3 Landmarks and Labelings

In this section, we first recall existing notions of landmarks and labelings and then introduce the new properties and schemes that will be studied in the remainder of the paper.

3.1 Notions of Landmarks

Landmark-based routing relies on selecting a (small) set of nodes as landmarks and pre-computing the shortest path distances from the landmarks to all other nodes. For any node pair $s, t \in V$ and any landmark node l, it follows from the triangle inequality that $|c(s,l) - c(t,l)| \leq c(s,t)$. Hence we get a valid lower bound for the shortest path distance between s and t. Choosing the maximum lower bound over all landmark nodes is an admissible heuristic for A*. The tightness of the lower bounds depends on the number and distribution of the selected landmarks. Landmarks with a roughly equal distance to s and t (and hence in particular landmarks in the middle of the shortest path from s to t) provide little to no information. In contrast, some landmarks even allow to obtain tight lower bounds. Such landmarks are called *perfect*.

▶ **Definition 1** (Perfect Landmark). *A landmark $l \in V$ is called* perfect *for a node pair $s, t \in V$ if $|c(s,l) - c(t,l)| = c(s,t)$.*

To characterize perfect landmarks, we next introduce the notion of a *path landmark*.

▶ **Definition 2** (Path Landmark). *A landmark $l \in V$ is a* path landmark *for a node pair $s, t \in V$, if t lies on a shortest path from s to l or s lies on a shortest path from t to l.*

The relation between path landmarks and perfect landmarks is described in the following two simple lemmas.

▶ **Lemma 3.** *Path landmarks are perfect.*

Proof. If w.l.o.g. t lies on a shortest path from s to l, then we have $c(s,t) + c(t,l) = c(s,l)$. Rearranging the formula, we get $c(s,l) - c(t,l) = c(s,t)$. ◀

▶ **Lemma 4.** *All perfect landmarks are path landmarks.*

Proof. Let $l \in V$ be a perfect landmark for s, t. Then we know that w.l.o.g. $c(s,l) - c(t,l) = c(s,t)$ and hence $c(s,t) + c(t,l) = c(s,l)$. The latter implies that t lies on a shortest path from s to l and hence l is a path landmark for s, t. ◀

It follows that if shortest paths are unique, then for an s-t-perfect landmark l the shortest path from s to t is always a prefix of the shortest path from s to l.

The lemmas also provide some justification why landmark selection strategies as e.g. the *farthest strategy* [23] which prefer nodes in the periphery of the graph usually work very well.

3.2 Notions of Labelings

Next, we delve into labeling schemes. Based on the definition of classical Hub Labeling, we propose two new labeling schemes, namely Landmark Labeling and Landmark Hub Labeling, which build on the notion of path landmarks discussed above.

▶ **Definition 5** (Hub Labeling). *A* Hub Labeling *(HL) is a labeling $L : V \to 2^V$ which fulfills the* (hub) cover property*, i.e., for any two nodes $s, t \in V$ there is a node $w \in L(s) \cap L(t)$ which is a hub, i.e. w is on the shortest path from s to t.*

A HL is called a *hierarchical* Hub Labeling (HHL), if for some node ranking $r : V \to \mathbb{N}$ the nodes in $L(v)$ all have rank at least $r(v)$. It was proven in [7] that for fixed r the HHL with minimum total label size is the so called *canonical* HHL in which $w \in L(v)$ if and only if there is a shortest path emerging from v on which w (and only w) has maximum rank.

The idea behind (H)HL is that for each node $w \in L(s) \cap L(t)$ an upper bound for the shortest path distance from s to t can be derived (if distances to the nodes in the labels are stored along), and that this upper bound is tight for an s-t-hub. But we can also design a labeling scheme, where for each node $w \in L(s) \cap L(t)$ a lower bound for the shortest path distance from s to t can be inferred and then demand that the largest lower bound has to be tight. This leads us to the definition of a Landmark Labeling.

▶ **Definition 6** (Landmark Labeling). *A* Landmark Labeling *(LL) is a labeling $L : V \to 2^V$ which fulfills the* landmark cover property*, i.e., for any two nodes $s, t \in V$ there is a node $w \in L(s) \cap L(t)$ which is a path landmark for s, t.*

Consider any shortest path π that contains the nodes s and t. Then in a HL, only nodes on the shortest subpath from s to t contribute to fulfilling the *hub cover property* for this node pair. In contrast, in a LL, only nodes on π except for the nodes between s and t contribute to the *landmark cover property* for s, t. Accordingly, for any node w on a shortest path through s and t, the correct shortest s-t-path distance could be interferred if w is contained in both of their labels, but the two labeling schemes considered so far restrict the selection to specific subsets. We now lift this restriction by allowing any node w on any shortest path π which contains s and t to cover the pair s, t.

▶ **Definition 7** (Landmark Hub Labeling). *A* Landmark Hub labeling *(LHL) is a labeling $L : V \to 2^V$ which fulfills the* landmark hub cover property*, i.e., for any two nodes $s, t \in V$ there is a node $w \in L(s) \cap L(t)$ which either is a hub or a path landmark for s, t.*

An important property to be considered in our study of LHL will be *path-consistency.*

▶ **Definition 8** (Path-Consistent Labeling). *A* path-consistent *(PC) labeling is a labeling, where for each $w \in L(v)$, we also have $w \in L(u)$ for all u on the shortest path from v to w.*

To illustrate that path-consistency is a rather natural labeling property, we prove that a canonical HHL is always PC.

▶ **Lemma 9.** *Canonical hierarchical Hub Labelings are path-consistent.*

Proof. Let $L(v)$ be the label of v in a canonical HHL that respects ranking r. Let $w \in L(v)$ be a node contained in the label. Clearly, for any node u on the shortest path from v to w, we have $r(u) < r(w)$. Otherwise w would not be included in $L(v)$. Hence w also is the node of highest rank in any suffix of the shortest path from v to w. Accordingly, for all nodes u on the shortest path from v to w, it holds that $w \in L(u)$. ◀

Table 1 Optimal maximum (and average) label sizes for Hub Labeling (HL), Landmark Labeling (LL) and path-consistent Landmark Hub Labeling (PC-LHL) on example instances.

	star graph	path graph
HL	$\Theta(1)$	$\Theta(\log n)$
LL	$\Theta(n)$	$\Theta(1)$
PC-LHL	$\Theta(1)$	$\Theta(1)$

We will argue that path-consistency is beneficial for all three types of labelings (HL, LL, LHL) when it comes to shortest path retrieval. However, for LHL we will show that the property is crucial for efficient query answering. So from now on, we focus on path-consistent Landmark Hub Labelings (PC-LHL). Note that by Lemma 9, the optimal label sizes in a PC-LHL are upper bounded by the optimal label sizes in any HHL, as we have shown that any canonical HHL is a PC-HL and any PC-HL is a PC-LHL by definition.

To further see why PC-LHL is interesting, Table 1 shows instances on which the label sizes in a PC-LHL are smaller than those in an HL or LL. This is a result of the greater freedom to choose cover nodes in LHL. For LL, the endpoints of any maximal shortest path have to contain at least one of them in both of their labels to fulfill the *landmark cover property*. Hence, LL requires large maximum label sizes on star graphs (and in general on connected graphs with many dead-ends). For HL and PC-LHL, it suffices to include the middle node of the star in all labels. On path graphs, however, the *hub cover property* of HL results in label sizes that are logarithmic in the number of nodes on the path [9]. For LL and LHL, choosing one endpoint as label for all nodes is sufficient (and also trivially PC).

4 Query Algorithms

We distinguish two different types of queries: (i) distance queries and (ii) path queries. We first describe how these types of queries are answered with classical HL and argue that similar algorithms work for LL. For PC-LHL, however, we describe a novel query answering algorithm that takes care of the possible mix of hubs and path landmarks in the labels.

4.1 Distance and Path Queries with HL and LL

For HL, answering a distance query for nodes s and t is very simple and can be summarized with the following formula: $c(s,t) = \min_{w \in L(s) \cap L(t)} c(s,w) + c(t,w)$. If the labels are presorted by node ID, the intersection of two labels $L(s)$ and $L(t)$ can be computed in time linear in $\max\{|L(s)|, |L(t)|\}$ and the query time is hence in $\mathcal{O}(L_{max})$ with $L_{\max} := \max_{v \in V} |L(v)|$. Similarly, for LL, we have $c(s,t) = \max_{w \in L(s) \cap L(t)} |c(s,w) - c(t,w)|$ and can hence answer distance queries in $\mathcal{O}(L_{\max})$ as well.

For path queries, where not only the length but the sequence of edges on a shortest path should be returned, query answering becomes more involved. To make shortest path retrieval more efficient, not only the distances to the hub nodes are stored with the labels but also the first edge on the shortest path. Then, in a path query, we first issue a distance query to identify the hub node w and the shortest path distance $c(s,t)$. The shortest path from s to t is then a concatenation of the shortest path from s to w and the shortest path from w to t. The two subpaths can be extracted independently.

To extract any shortest a-b-path, first a distance query is issued and the respective hub w_{ab} is indentified. If $w_{ab} \neq a$, then the first edge on the shortest path from a to b is the one associated with the label $w \in L(a)$. Otherwise, we have $a \in L(b)$ and the edge associated with this label is the last edge on the shortest path from a to b. Hence we either retrieve the first edge (a, a') or the last edge (b', b) of the shortest path and can then recursively repeat the procedure.

Overall, shortest path retrieval demands to issue k distance queries, where k denotes the number of edges on the shortest path. The running time for a path query is therefore in $\mathcal{O}(D \cdot L_{\max})$ where D denotes the graph diameter. A similar procedure works for LL, but we only need to return the subpath from s to t instead of the whole path to the landmark node.

4.2 Path Queries with Path-Consistent Labelings

In case we deal with a path-consistent (PC) labeling, path queries can be answered faster as follows. To each node $l \in L(s)$, we assign a pointer to the entry of $l \in L(s')$ where s' is the node after s on the shortest path from s to l. Note that $l \in L(s')$ follows directly from the PC property and that the space consumption of the labeling only increased linearly by the addition of these pointers. The shortest path from s to $l \in L(s)$ can then simply be retrieved by following the pointers towards l iteratively. Thus, path retrieval takes only $\mathcal{O}(k)$ where k denotes the number of shortest path edges. As a path query is composed of a distance query and subsequent path retrieval, we get the following corollary.

▶ **Corollary 10.** *Path query times for PC labelings are in* $\mathcal{O}(L_{\max} + D)$.

4.3 Query Answering with PC-LHL

For PC-LHL, we know that $L(s) \cap L(t)$ needs to contain either a hub or a path landmark for s, t but we don't know which of the two it is. And the role of a node in $L(s)$ might be different for different targets t. We hence apply both, the HL and the LL distance query answering routine to the label sets. This provides us with an upper bound $UB := \min_{w \in L(s) \cap L(t)} c(s, w) + c(t, w)$ and a lower bound $LB := \max_{w \in L(s) \cap L(t)} |c(s, w) - c(t, w)|$ on the shortest path distance, with at least one of them being tight.

If $UB > LB$, we have to figure out whether a shortest path of length LB exists. Let w_l be the label in $L(s) \cap L(t)$ from which LB resulted. If w_l is a path landmark for s, t, then w.l.o.g. t has to lie on the shortest path from s to w_l at distance LB from s. We hence need to extract the prefix of the shortest path from s to w_l up to a distance of LB and check whether t is found. For prefix extraction, we can use the path query algorithm for PC labelings, see Corollary 10. If t is found, the extracted prefix is the shortest path from s to t. Otherwise, UB is tight and the shortest path from s to t can be identified by extracting the paths from s and t to the respective hub node w_h (again using PC queries). The total running time of the query algorithm for PC-LHL is hence in $\mathcal{O}(L_{\max} + D)$ for both, distance and path queries.

Accordingly, PC-LHL has a worse distance query time than HL and LL, as path retrieval is necessary to decide whether UB or LB is tight. For path queries, however, the worst-case running time is better due to the enforced PC property. If we consider PC-HL and PC-LL then worst case path retrieval times match, but PC-LHL allows for smaller label sizes.

5 Parametrized Bounds for Landmark Hub Label Sizes

Next, we investigate how to construct PC-LHL with bounded label size. For that purpose, we exploit two parameters that measure the complexity of the shortest path structure of the input graph G, namely the *geodesic transversal number* [31] and the *highway dimension* [2].

5.1 Obtaining Labels from Geodesic Transversal Sets

The geodesic transversal number, abbreviated by $gt(G)$, denotes the size of a smallest hitting set for all maximal shortest paths in G. A shortest path (also called geodesic) is maximal if there is no superpath that is also a shortest path. A hitting set S for the set of maximal geodesics Π is also called a gt-set. Given a gt-set, we can easily construct a valid PC-LHL as shown in the following theorem.

▶ **Theorem 11.** *Given a graph $G(V, E)$, the maximum label size $L_{\max}$ of a PC-LHL can be bounded by the geodesic transversal number $gt(G)$.*

Proof. Let S be a gt-set in G. We claim that setting $L(v) = S$ for all $v \in V$ consitutes a valid PC-LHL. To see that, consider any pair of nodes s, t and the respective shortest path π. Then there is at least one maximal geodesic π' in G which contains π as a subpath. If $\pi \cap S \neq \emptyset$, then $L(s) \cap L(t) = S$ contains a valid hub for π. Otherwise, S has to contain at least one node from $\pi' \setminus \pi$ which constitutes a path landmark for s, t. Hence the landmark cover property is fulfilled. Path-consistency is obviously obeyed in case all labels are identical. ◀

Note, that the bound does not apply to HL or LL. This follows from the examples in Table 1. For path and star graphs, we have $gt = 1$ but HL and LL label sizes depend on n, respectively.

On trees, optimal gt-sets can be computed in linear time, but on general graphs deciding whether a graph has a gt-set of size $k \in \mathbb{N}$ is NP-hard [31].

However, one can find a gt-set of size $\mathcal{O}(gt \log n)$ in $\mathcal{O}(n^3)$ by extracting the set Π of all maximal geodesics from the input graph and computing a greedy hitting set thereupon. More precisely, after enumerating the $\mathcal{O}(n^2)$ maximal geodesics, we create a boolean array A which encodes whether a geodesic was already hit. Initially, the array is filled with *false* entries. Then we sweep over the geodesics and assign their IDs to each node contained therein. Additionally, we store the size of the resulting set at each node in form of a counter. This takes $\mathcal{O}(n^3)$ overall. Subsequently, in each of the $\mathcal{O}(n)$ rounds of the algorithm, the node with the largest counter value is identified in $\mathcal{O}(n)$ and added to the initially empty hitting set S. The counter of the node is set to zero. For all geodesics assigned to the selected node, we check in A whether they are newly hit. This takes constant time per geodesic and hence $\mathcal{O}(n^2)$ for all geodesics assigned to the node. For each newly hit geodetic, we set its entry in A from *false* to *true*, and then sweep over all nodes contained in that geodesic in order to decrement their counters. Note that we do not need to find or remove the respective ID of the geodesic from the stored sets at the nodes. Hence the update time of the data structures is in $\mathcal{O}(n)$ for each newly hit geodesic and in $\mathcal{O}(n^3)$ for all of them. The algorithm terminates if only nodes with a counter of zero are left. The total running time of the algorithm is hence in $\mathcal{O}(n^3)$.

We further remark that incorporating the fact that shortest paths have low VC-dimension, one can even compute gt-sets of size $\mathcal{O}(gt \log gt)$ in polytime [1].

5.2 Relation to Highway Dimension

The highway dimension h is a graph parameter that was specifically designed to study preprocessing-based shortest path planning techniques in road networks. For brevity, we don't include the rather involved definition here. For our considerations, it will only be important that the highway dimension is at least as large as the maximum node degree in the graph [2].

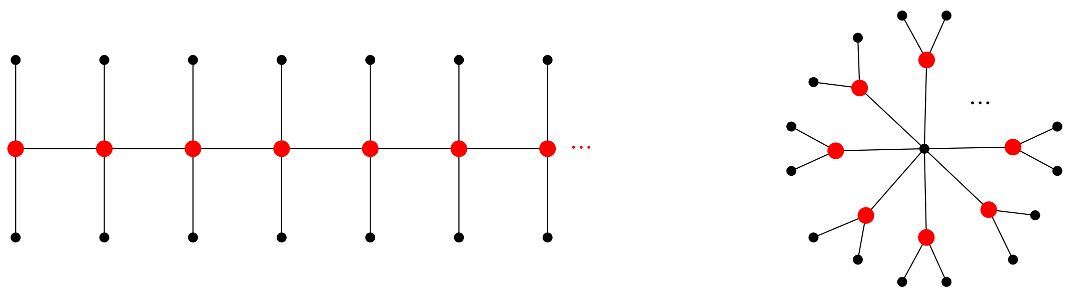

Figure 1 Example graphs with their optimal gt-sets in red. Left: special caterpillar. Right: augmented star graph.

For classical HL, labels of size $\mathcal{O}(h \log D \log n)$ can be computed in polytime, where D is the graph diameter. The construction also relies on hitting sets, more precisely on so called multi-level sparse shortest path hitting sets (SPHS) [2]. The number of levels is $\Theta(\log D)$. The top level is a hitting set for all shortest paths of length at least $\frac{D}{2}$ which can be greedily constructed in $\mathcal{O}(n^3)$. The lowest level simply includes all nodes. The other SPHS levels each require to solve up to n local shortest path hitting set instances (each considering paths of length 2^{i-1} at level i for $i = 1, \ldots, \lceil \log D \rceil - 1$). Hence (without further improvements) the running time per level is in $\mathcal{O}(n^4)$, resulting in a total running time of $\mathcal{O}(n^4 \log D)$. The node labels can then be interferred from the multi-level SPHS in $\mathcal{O}(n^3)$.

Again, exploiting the fact that shortest path sets have small VC-dimension, the bound can be improved to $\mathcal{O}(h \log D \log h)$ [1]. The question is whether this bound is tighter than the *gt*-bound. To answer this question, we study the relationship of *gt* and *h*.

▶ **Lemma 12.** *The geodesic transversal number $gt(G)$ and the highway dimension $h(G)$ are incomparable.*

Proof. Consider a star graph. The geodesic transversal number is 1, as the center node is a valid hitting set for all maximal geodesics. As the center node has a degree of $n-1$ and the highway dimension is lower bounded by the maximum node degree, we have $h \in \Omega(n)$.

For an example graph in which $h \ll gt$, we consider a specific tree graph, called a caterpillar. In a caterpillar, there exists a backbone path that contains all nodes of degree larger than two. We assume now that the backbone contains $n/3$ nodes, each of which is adjacent to two leaf nodes. Then, for each backbone node, the path between its two attached leaves is a maximal geodesic; each of which requires its own hitter in a *gt*-set. See Figure 1 (left) for an illustration. Therefore, we have $gt \in \Theta(n)$. In [12], it was shown that in caterpillars with constant node degree the highway dimension is a constant. ◀

Interestingly, we will show that the *h*-dependent upper bound nevertheless also holds for PC-LHL by proving that the SPHS-based labeling algorithm produces a path-consistent HL.

▶ **Lemma 13.** *The HL returned by the multi-level SPHS algorithm is path-consistent.*

Proof. Node labels are computed in the algorithm as follows: Based on the multi-level SPHS each node receives a value between 0 and $\lceil \log D \rceil$ indicating the highest level in which it appears as a hitter. For a node $v \in V$, the shortest path tree emerging from v is computed. Then along each shortest path starting at v, nodes are included in the label of v that have a higher level than all the ones previously seen on that shortest path. Hence the resulting labeling is a canonical HHL with respect to the level function. As proven in Lemma 9, such a labeling is always PC. ◀

The implications of the lemma are twofold: Firstly, the labels produced by the SPHS-based algorithm can benefit from the improved path queries times described in Section 4.2. Secondly, as every PC-HL is also a valid PC-LHL by definition, we get the following corollary.

▶ **Corollary 14.** *The label size in a PC-LHL can be bounded by $\mathcal{O}(h \log D)$ and labels of size $\mathcal{O}(h \log D \log h)$ can be computed in polytime.*

So now we have two valid upper bounds for the maximum label size in a PC-LHL, one depending on gt and one depending on h. The maximum label size $L_{\max}$ can be significantly smaller than those graph parameters, though, as shown in the next lemma.

▶ **Lemma 15.** *There exist graphs $G(V, E)$ in which the maximum label size of a PC-LHL is in $\mathcal{O}(1)$ but $gt \in \Theta(n)$ and $h \in \Theta(n)$.*

Proof. Consider an augmented star graph with $n/3$ arms where at the end of each arm is adjacent to the center node of the star but also to two other nodes of degree 1 each (as shown in Figure 1, right). Then there are $n/3$ maximal shortest paths that consist of two edges each, namely the ones that connect the degree-1 nodes via the intermediate star arm nodes. As none of those $n/3$ paths overlap, we have $gt \in \Theta(n)$. From the center node of the star having a degree of $\Theta(n)$, it follows that $h \in \Theta(n)$ holds as well. For a PC-LHL, each degree-1 node needs to contain its one adjacent arm node in its label and the center node of the star. All other node labels only need to contain the center node of the star. The resulting labeling is path-consistent. Thus, we get $L_{\max} \in \mathcal{O}(1)$. ◀

It follows from the lemma that the obtained parameterized upper bounds do not provide us with an approximation guarantee for the (maximum or total) label size of a PC-LHL. Therefore, we will next discuss how to design a proper approximation algorithm.

6 An Approximation Algorithm for Path-Consistent Labelings

For classical HL, an $\mathcal{O}(\log n)$-aproximation with a running time of $\mathcal{O}(n^5)$ for the total (and hence also average) label size was proposed in [7], which is based on the greedy set cover algorithm. As the respective set cover instance would have exponential size, it cannot be constructed explicitly, though. But it was shown that the greedy selection of a sufficiently good set can be accomplished in polytime based on a reduction to the Densest Subgraph Problem. We will recap the algorithm details below and show that the algorithm can easily be modified to also approximate the size of a LL or LHL. However, incorporating the PC property poses a new challenge.

6.1 Hub Labels and Densest Subgraphs

To obey the *hub cover property*, there needs to be node for each shortest path that is contained in the labels of both of its endpoints. To construct a cooresponding set cover instance $(U, \mathcal{S})$, the universe U is the set of all (unordered) pairs of nodes (representing the set of all shortest paths). The possible sets $\mathcal{S}$ are induced by a hub node w and a node subset W. Then $S(w, W)$ covers all pairs of nodes in W for which w is a valid hub, and the cost of S is equal to the size of W, that is $c(S) = |W|$. An optimal set cover solution for $(U, \mathcal{S})$ constitutes a valid hub labeling of minimum size (by including w in $L(v \in W)$ for all selected sets). A greedy set cover solution guarantees an approximation factor of $\mathcal{O}(\log |U|)$. However, the problem is, that there are exponentially many subsets W to consider for each potential hub node $w \in V$. Hence the set cover instance cannot be constructed efficiently.

But for the greedy algorithm to work, one only needs fast access to the set $S \in \mathcal{S}$ with the largest ratio $p(S)/c(S)$ where $p(S)$ denotes the number of new node pairs covered by S and $c(S)$ the cost of the set. The idea put forward in [7] is to compute in each round the currently best subset W for each potential hub node w by solving an instance of the Densest Subgraph Problem (DSG) and then simply select the best among those $\mathcal{O}(n)$ sets. Given a graph, the DSG demands to find the node subset D for which the subgraph induced by D has maximum density, i.e. largest ratio of number of edges and number of nodes. The problem can be solved to optimality in polytime and there is also linear time 2-approximation [28]. To use DSG for HL set cover, a so called *center graph* $G_w(V, E_w)$ is constructed in each round for each potential hub w. The center graph contains the same node set as the input graph and there is an edge between nodes s, t if and only if w is on a shortest path from s to t, i.e. if w is a valid hub for the node pair s, t. The DSG of the center graph then equals the currently best node subset $W = D$ to include node w in their labels. Note that in the course of the algorithm it might happen that the same node w is selected multiple times with different node subsets, and that there might be node pairs newly covered by w where one node was selected in a previous round. To make sure that such node pairs are included in $p(S)$ without double counting the addition of w to the label, nodes that already have w in their label receive a cost of 0 in the center graph and all others a cost of 1. Note that this results in a change of $c(S)$, the cost of set S. While the greedy set cover algorithm does not necessarily uphold its approximation guarantee if the costs of the sets change in the process, it was proven in [7] that for each potential hub w, the density of the best subgraph in each round may only decrease and hence the approximation factor is preserved.

For LL and LHL, the very same approximation framework can be used. The only difference is that edges in the center graphs are inserted in accordance with the respective cover property.

6.2 Ensuring Path-Consistency

The greedy set cover algorithm does not automatically produce PC labelings, though. Hence we need to change the model and also the greedy selection method appropriately to compute a valid PC-LHL (or PC-HL or PC-LL).

First of all, $\mathcal{S}$ can be restricted to sets $S(w, W)$ where W is a connected subgraph of the shortest path tree emerging from w. Otherwise the PC property would be violated. We refer to such sets W from now on as *trimmed shortest path trees* (rooted in w). The number of trimmed shortest path trees might still be exponential, though, and solving the DSG problem for the center graph of w might return a node set D which does not coincide with a valid set $S(w, W)$, i.e. where $W = D$ is not a trimmed shortest path tree.

To be able to use the greedy set cover framework nevertheless and to preserve its approximation guarantee, we need an efficient algorithm for selecting a valid set $S(w, W)$ with close-to-optimal ratio of newly covered pairs and cost. We hence investigate the problem of computing the densest subgraph of the center graph where the node set corresponds to a trimmed shortest path tree rooted in w. Thus, in each round, we construct the center graph $G_w(V, E_w)$ for w as before. We also compute the full shortest path tree T rooted in w. The density $d(S)$ of a subgraph S of T is defined as $m(S)$, the number of edges incident to the respective nodes in G_w, divided by $n(S)$, the number of nodes in the subgraph with weight 1 (again, nodes will receive a cost of zero if they already contain node w in their label). For technical purposes, we assume that $w \in L(w)$ for all nodes $w \in V$. This obviously does not affect the PC-property. As the label size of all nodes has to be at least 1 for correct query answering anyway, this will also not affect the approximation factor we are going to show.

We then propose the following greedy algorithm to find a trimmed shortest path tree with close-to-maximum density: Let T be the current trimmed shortest path tree with edges directed away from the root. Compute the density of all proper subtrees of T. Here a subtree of T rooted at node $v \in T$ contains all nodes reachable from v in T via the directed edges. Remove the subtree with smallest density from T and also remove the contained nodes and their incident edges from G_w. Repeat until T consists of only the root or until we have $n(S) = 0$ for all subtrees. Keep track of the temporary trimmed tree T with maximum density over the course of the algorithm and return it in the end.

▶ **Theorem 16.** *The greedy algorithm returns a trimmed shortest path tree with density $\lambda/2$ where λ denotes the maximum density among all trimmed shortest path trees rooted in w.*

Proof. The proof follows the argumentation for the 2-approximation of DSG [28] but is more intricate as it has to deal with the tree structure.

Let T_{OPT} be the optimal trimmed tree, i.e. the one with maximum density λ. Observe that every directed subtree of T_{OPT} has to have a density of at least λ. Otherwise, the removal of that subtree would increase the density.

Now consider the first iteration in the greedy algorithm in which a subtree S^* is removed which intersects T_{OPT}. Before the removal of S^*, all nodes of T_{OPT} are still contained in the current T and G_w. Hence the intersection tree T^* of S^* and T_{OPT} needs to have a density of at least λ by the observation made above. It follows that $d(S^*) \geq \lambda$, as either S^* is a subtree of T_{OPT}, or, in case S^* contains nodes outside of T_{OPT}, its subtree $S \setminus T_{OPT}$ needs to have higher density than S (or equal density) to justify the greedy choice. As we know that there are at least $\lambda \cdot n(T^*)$ edges incident to nodes from T^* which connect them to other nodes from T_{OPT} in G_w, adding T^* to $S \setminus T_{OPT}$ cannot decrease the density to a value below λ.

If S^* – as the subtree of currently lowest density – has $d(S^*) \geq \lambda$, then all proper subtrees of the current T need to have a density $\geq \lambda$ as well. We now show that this implies that $d(T) \geq \lambda/4$. Each node in T except for the root belongs to one maximal proper subtree S_m of T. For each S_m, we know that based on $d(S_m) \geq \lambda$ there are at least $\lambda \cdot n(S_m)$ edges in G_w that are incident to nodes from S_m. By summing up over all S_m and accounting for double counting, we conclude that the total number of edges in the current G_w and hence $m(T)$ is at least $\sum_{S_m} n(S_m) \cdot \lambda/2 = n(T) \cdot \lambda/2$. The latter follows as $w \in L(w)$ anyway and hence the root has a weight of zero and does not contribute to $n(T)$. In conclusion, we get $d(T) = m(T)/n(T) \geq \lambda/2$. As the algorithm returns the T with highest density over the course of the algorithm, the theorem follows. ◀

Note that this quality guarantee could not be shown if in each greedy step only the leaf of lowest density instead of the subtree of lowest density would be removed.

▶ **Lemma 17.** *Over the course of the set cover algorithm, the maximum density of a trimmed shortest path tree rooted in w can only decrease.*

Proof. Assume for contradiction that at some stage of the algorithm, the maximum density of a trimmed shortest path tree rooted at w is $D = d(T_1)$ but in some later round, there is a trimmed shortest path tree T_2 with $d(T_2) > D$. As the number of edges in G_w can only decrease over the course of the algorithm (as more and more node pairs are already covered), the only way to increase density for a subgraph is that some of its contained nodes already include w in their label sets (due to a selection of some set $S(w, W)$) and hence those nodes now have cost zero and do not contribute anymore to $n(T)$. We hence consider a moment where a tree T^* was selected as trimmed tree of largest density with $T^* \cap T_2 \neq \emptyset$. It follows that $d(T_2) \leq d(T^* \cup T_2)$ at this point and hence also $d(T^*) > d(T_2 \setminus T^*)$. But as the density of T_2 coincides with $d(T_2 \setminus T^*)$ after the removal of T^*, it follows that its density can not be larger than that of T^* at the time of its removal. ◀

Based on Theorem 16 and Lemma 17, the greedy set cover algorithm utilizing trimmed shortest path trees maintains an $\mathcal{O}(\log n)$ approximation guarantee.

Regarding the running time, we observe that determining a dense trimmed shortest path tree for root w requires $\mathcal{O}(n^2)$ time for initializing T and constructing G_w (assuming shortest path trees are precomputed and looking up whether a certain node pair is already covered takes constant time). Then there are $\mathcal{O}(n)$ rounds of greedy subtree removal. In each round, we compute the density of all subtrees of T and delete the one of smallest density from T an G_w, both of which can be done in $\mathcal{O}(n^2)$. Hence for each root w, the dense trimmed shortest path tree computation takes $\mathcal{O}(n^3)$. As we have to compute those trees for all nodes, this amounts to $\mathcal{O}(n^4)$ per iteration in the set cover algorithm. The set cover algorithm works then in the same way as the one described in [7] for HL. As our dense trimmed shortest path tree computations need a factor of n longer than the linear 2-approximation for DSG used in their approach, we end up with a total running time of $\mathcal{O}(n^6)$.

▶ **Theorem 18.** *A PC-LHL (as well as a PC-HL or PC-LL) with a total/average label size within a factor of $\mathcal{O}(\log n)$ of the optimal size can be computed in $\mathcal{O}(n^6)$.*

7 Hierarchical Labelings

A hierarchical labeling is a special kind of labeling $L : V \to 2^V$ that respects a given node ordering $r : V \to [n]$. Here, nodes in the label of a node have to be at least as important as the node itself, that is, for all $w \in L(v)$, we have $r(w) \geq r(v)$. For HHL, it was shown that for a given ordering r there is a simple rule that produces labels that are both necessary and sufficient for correct query answering. These are called *canonical* labels $L_c : V \to 2^V$. As discussed in Section 3.2, the canonical condition for HHL is that $w \in L_c(v)$ if and only if w is the node of highest rank on any shortest v-w-path. For any valid HHL L, it then yields $L(v) \supseteq L_c(v)$. Hence the theoretical study of optimal HHL sizes can be restricted to canonical HHL and practical algorithms should strive to produce labels close to the canonical ones. As proven in Lemma 9, a canonical HHL is always path-consistent. This implies that a canonical HHL is also a valid (hierarchical) PC-LHL. Therefore, label sizes in a (H)PC-LHL are at most as large as those in an HHL, and parametrized upper bounds for HHL transfer to (H)PC-LHL. But the hope is, of course, that HPC-LHL actually produces smaller labels than HHL. To investigate this, we now define the *canonical* HPC-LHL.

▶ **Definition 19** (Canonical HPC-LHL). *For a given node ordering $r : V \to [n]$, the* canonical *HPC-LHL that respects r assigns to each node v a label that consists of the the nodes of highest rank on each maximal geodesic that contains v.*

We first prove the sufficiency of the condition.

▶ **Lemma 20.** *A canonical HPC-LHL is a valid PC-LHL.*

Proof. Let s, t be any node pair and π^+ a maximal geodesic that contains both s and t (and is hence a superpath of the shortest s-t-path π). Let v be the node of highest rank on π^+. It follows by the canonical condition that $v \in L(s) \cap L(t)$. Accordingly, v is either a hub for s, t, in case $v \in \pi$ or it is a path landmark for s, t, in case $v \in \pi^+ \setminus \pi$. In both cases, a tight distance bound can be inferred from v. As all nodes on π^+ contain v in their labels, path-consistency is trivially obeyed. ◀

Next, we prove that the condition is also necessary.

▶ **Lemma 21.** *For a given node ordering $r : V \to [n]$, canonical hierarchical labels are a subset of any valid HPC-LHL.*

Proof. We want to show that for all $v \in V$, we have $L_c(v) \subseteq L(v)$ where L_c denotes the canonical HPC-LHL and L any valid HPC-LHL for the same r. Consider a $w \in L_c(v)$ and assume for contradiction that $w \notin L(v)$. Let Π be the set of maximal geodesics containing both v and w. We know that in this set there has to be at least one geodesic on which w has maximum rank for it to be included in $L_c(v)$. Consider such a geodesic π and let its end nodes be s and t. Based on the unique shortest path assumption, this is the only geodesic that contains both s and t. Therefore, to ensure correct query answering, s and t need to have a common node from π in their labels. And by the sake of the PC condition, all other nodes on π need to include the same node in their label. Hence, if we assume that this common node is not w but some other node w', we would demand that $w' \in L(w)$. But as w has maximum rank on π, including w' in $L(w)$ with $r(w') < r(w)$ would violate the definition of a hierarchical labeling. Accordingly, our initial assumption that $w \notin L(v)$ is wrong for all v where there exists a maximal geodesic containing v and w and where w has maximum rank on it. It follows that $w \in L_c(v) \Rightarrow w \in L(v)$. ◀

As the canonical condition is both necessary and sufficient, we know that among all valid HPC-LHL, the canonical one has the smallest possible label size. Note that for a given ranking $r : V \to [n]$, the canonical HPC-LHL can be computed in polytime by extracting the set of all maximal geodesics Π, determining the node of maximum rank on each $\pi \in \Pi$ and assigning that node to the label of all nodes in π.

8 Proof-Of-Concept Study

While it is out of scope of this paper to provide an extensive experimental study and to develop engineered algorithms for HPC-LHL that scale to large networks, we provide a small feasibility study to see whether the concept might have practical merits. For that purpose, we implemented the naive algorithm to compute a canonical HPC-LHL as well as the respective query routine in C++. Experiments were conducted on a single core of an AMD Ryzen Threadripper 3970X 32-Core Processor clocked at 2.061GHz with 256 GB RAM.

As benchmark networks, we used random rectangular cutouts of the European road network extracted from OpenStreetMap[1], restricted to the largest strongly connected component contained therein. Because the computation of all maximal geodesics is space- and time-consuming, we restricted ourselves to subgraphs of up to 100,000 nodes. Then we computed the canonical HHL based on [4] and used the resulting ordering r as basis for HPC-LHL; hence both labelings use the same node hierarchy. We observed that the average label size in the HPC-LHL was about 5% to 12% smaller than the respective label size in the HHL. We expect this gap to increase in larger networks where we have longer chains of degree-2 nodes on which HPC-LHL needs provably less label nodes than HHL. Furthermore, we analyzed and compared the different query answering methods for 10,000 random shortest path queries in each network. For distance queries, both methods answer queries within a few microseconds. HHL query answering is slightly superior to HPC-LHL query answering as it never requires path retrieval. We found that for HPC-LHL in about 88% of the cases the upper bound was tight, in 11% of the cases both upper and lower bound, and in the remaining 1% solely the lower bound. However, as path retrieval is very fast when exploiting

[1] `https://i11www.iti.kit.edu/resources/roadgraphs.php`

the PC property, the query times did never differ by more than a factor of 1.4 and on average by less than two percent. For path queries, however, using the PC-property allows for a speed-up of a factor of roughly 500. Here, both HHL and HPC-LHL can benefit from the speed-up based on our proof that HHL are always PC. We conclude that at least on the investigated graphs, HPC-LHL allows to produce smaller label sizes with a negligible increase in query time for distance queries. Furthermore, we have clearly demonstrated that the PC property is very useful in practice as it enables significantly accelerated shortest path retrieval. This effect should also be even more pronounced in bigger graphs with larger diameters.

9 Conclusions and Future Work

We proposed to augment the Hub Labeling idea with lower bound information and introduced Landmark Hub Labeling (LHL) as an alternative to classical labelings. As LHL allows for a greater variety of valid node labels than standard (hierarchical) Hub Labeling, label sizes are potentially smaller. This is important as one of the main drawbacks of labeling approaches in practical applications is the huge space consumption. We showed interesting theoretical properties of (path-consistent) LHL and conducted an initial feasibility study. To make the PC-LHL applicable to larger graphs, better scalable algorithms have to be designed that nevertheless compute concise labels. Hierarchical Hub Labels can be efficiently infered from a Contraction Hierarchy (CH) data structure [22]. It might be possible to also exploit CH for PC-LHL construction or to modify and prune labels from a given HHL.

On the theoretical side, it would be interesting to investigate whether the logarithmic gap between HL and PC-LHL on path graphs is tight or whether there exist other types of graphs with a larger separation. Furthermore, the $\mathcal{O}(n^6)$ running time of the approximation algorithm can potentially be reduced by incorporating ideas from [16], which were used to improve the running time of the HL approximation algorithm from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^3 \log n)$. Regarding graph parameters, we showed that label sizes in PC-LHL can be upper bounded by functions depending on the highway dimension or the geodesic transversal number. Another parameter to investaiget would be the skeletoon dimension, which was used in previous work to also uppper bound label sizes of (non-hierarchical) HL. Also there are no parametrized bounds for Landmark Labeling so far.

Finally, PC-LHL might provide a good basis for an approximate distance oracle by weakening the property that either the label based lower or upper bound needs to be tight. Instead, one could demand that the (absolute or relative) difference of the two bounds is below a certain threshold which would then automatically provide a quality guarantee.

References

1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Vc-dimension and shortest path algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 690–699. Springer, 2011.

2 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension and provably efficient shortest path algorithms. *Journal of the ACM (JACM)*, 63(5):1–26, 2016.

3 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Proc. 10th Int. Symp. Experimental Algorithms (SEA '11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011. `doi:10.1007/978-3-642-20662-7_20`.

4 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Hierarchical hub labelings for shortest paths. In *Proc. 20th Ann. Europ. Symp. Algorithms (ESA '12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012. doi:10.1007/978-3-642-33090-2_4.

5 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD '13)*, pages 349–360. ACM, 2013. doi:10.1145/2463676.2465315.

6 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *Proc. 23rd Int. Conf. World Wide Web (WWW '14)*, pages 237–248. ACM, 2014. doi:10.1145/2566486.2568007.

7 Maxim A. Babenko, Andrew V. Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller. On the complexity of hub labeling (extended abstract). In *Proc. 40th Int. Symp. Mathematical Foundations of Computer Science (MFCS '15)*, volume 9235 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 2015. doi:10.1007/978-3-662-48054-0_6.

8 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

9 Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing speed-up techniques is hard. In *International Conference on Algorithms and Complexity*, pages 359–370. Springer, 2010.

10 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016.

11 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, 15:2–1, 2010.

12 Johannes Blum. Hierarchy of transportation network parameters and hardness results. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

13 Zitong Chen, Ada Wai-Chee Fu, Minhao Jiang, Eric Lo, and Pengfei Zhang. P2H: efficient distance querying on road networks by projected vertex separators. In *Proc. 2021 Int. Conf. Management of Data (SIGMOD '21)*, pages 313–325. ACM, 2021. doi:10.1145/3448016.3459245.

14 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003. doi:10.1137/S0097539702403098.

15 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust distance queries on massive networks. In *Proc. 22th Ann. Europ. Symp. Algorithms (ESA '14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, 2014. doi:10.1007/978-3-662-44777-2_27.

16 Daniel Delling, Andrew V. Goldberg, Ruslan Savchenko, and Renato F. Werneck. Hub labels: Theory and practice. In *Proc. 13th Int. Symp. Experimental Algorithms (SEA '14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2014. doi:10.1007/978-3-319-07959-2_22.

17 Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hub label compression. In *Proc. 12th Int. Symp. Experimental Algorithms (SEA '13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2013. doi:10.1007/978-3-642-38527-8_4.

18 Daniel Delling and Giacomo Nannicini. Core routing on dynamic time-dependent road networks. *INFORMS Journal on Computing*, 24(2):187–201, 2012.

19 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Highway hierarchies star. In *The Shortest Path Problem*, pages 141–174, 2006.

20 Daniel Delling and Dorothea Wagner. Landmark-based routing in dynamic graphs. In *International Workshop on Experimental and Efficient Algorithms*, pages 52–65. Springer, 2007.

21 Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 25–30, 2013.

22 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

23 Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, pages 156–165. Citeseer, 2005.

24 Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Better landmarks within reach. In *International Workshop on Experimental and Efficient Algorithms*, pages 38–51. Springer, 2007.

25 Andrew V Goldberg and Renato Fonseca F Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

26 Daniel Harabor and Peter Stuckey. Forward search in contraction hierarchies. In *International Symposium on Combinatorial Search*, volume 9(1), 2018.

27 Famei He, Yina Xu, Xuren Wang, and Anran Feng. Alt-based route planning in dynamic time-dependent road networks. In *Proceedings of the 2019 2nd International Conference on Machine Learning and Machine Intelligence*, pages 35–39, 2019.

28 Samir Khuller and Barna Saha. On finding dense subgraphs. In *International colloquium on automata, languages, and programming*, pages 597–608. Springer, 2009.

29 Adrian Kosowski and Laurent Viennot. Beyond highway dimension: small distance labels using tree skeletons. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1462–1478. SIAM, 2017.

30 Lei Li, Sibo Wang, and Xiaofang Zhou. Time-dependent hop labeling on road network. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 902–913. IEEE, 2019.

31 Paul Manuel, Boštjan Brešar, and Sandi Klavžar. The geodesic-transversal problem. *Applied Mathematics and Computation*, 413:126621, 2022.

32 Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *Proc. 2018 Int. Conf. Management of Data (SIGMOD '18)*, pages 709–724. ACM, 2018. doi:10.1145/3183713.3196913.

33 Genaro Peque, Junji Urata, and Takamasa Iryo. Implementing an alt algorithm for large-scale time-dependent networks. In *22nd International Conference of Hong Kong Society for Transportation Studies: Transport and Society, HKSTS 2017*, pages 515–522. Hong Kong Society for Transportation Studies Limited, 2017.

34 Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 867–876, 2009.