Landmark Hub Labeling: Improved Bounds and Faster Query Answering

Justine Cauvi
École Normale Supérieure de Lyon, France
Department of Computer and Information Science, University of Konstanz, Germany

Ruoying Li
Department of Computer and Information Science, University of Konstanz, Germany

Sabine Storandt
Department of Computer and Information Science, University of Konstanz, Germany

Abstract

Hub Labeling (HL) is a state-of-the-art method for answering shortest-distance queries between node pairs in weighted graphs. It provides very fast query times but also requires considerable additional space to store the label information. Recently, a generalization of HL, called Landmark Hub Labeling (LHL), has been proposed, that conceptionally allows a storage of fewer label information without compromising the optimality of the query result. However, query answering with LHL was shown to be slower than with HL, both in theory and practice. Furthermore, it was not clear whether there are graphs with a substantial space reduction when using LHL instead of HL. In this paper, we describe a new way of storing label information of an LHL such that query times are significantly reduced and then asymptotically match those of HL. Thus, we alleviate the so far greatest shortcoming of LHL compared to HL. Moreover, we show that for the practically relevant hierarchical versions (HHL and HLHL), there are graphs in which the label size of an optimal HLHL is a factor of $\Theta(\sqrt{n})$ smaller than that of an optimal HHL. We establish further novel bounds between different labeling variants. Additionally, we provide a comparative experimental study between approximation algorithms for HL and LHL. We demonstrate that label sizes in an LHL are consistently smaller than those of HL across diverse benchmark graphs, including road networks.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Route Planning, Shortest Path, Hub Labeling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.1

1 Introduction

Efficiently determining the shortest path distance between node pairs in a given weighted graph $G(V, E, c)$ is important for a multitude of applications, including navigation, vehicle routing, and network design [3, 4, 13, 14, 15]. Using Dijkstra's algorithm, such distance queries can be answered in $\mathcal{O}(n \log n + m)$ with $|V| = n$ and $|E| = m$. However, if many queries need to be answered on the same input graph, preprocessing methods can help to tremendously reduce subsequent query times. One state-of-the-art method is Hub Labeling (HL) [6]. In a HL, each node $v \in V$ gets assigned a node label $L(v) \subseteq V$, and the shortest path cost $c(\pi(v, w))$ is stored along for each $w \in L(v)$. The labels need to fulfill the so called cover property that demands that $\forall s, t \in V : \exists v \in L(s) \cap L(t) \cap \pi(s, t)$. In other words, for each shortest path, its end nodes need to have at least one common node from that path in their respective labels. The node is then called a hub for s, t.

This allows for a very simple and fast query answering routine: For each $v \in L(s) \cap L(t)$ compute $c(\pi(s, v)) + c(\pi(t, v))$, with both values being precomputed, and keep track of the minimum. The resulting value is guaranteed to coincide with the shortest path distance

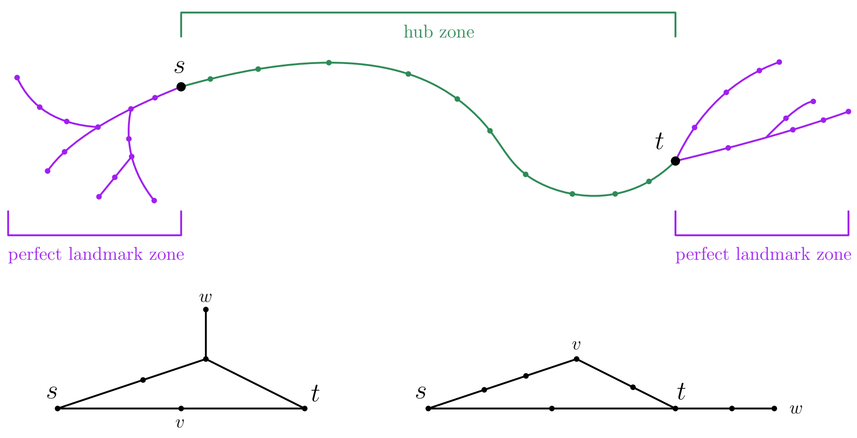

Figure 1 Top: Illustration of hub zone (green) and perfect landmark zone (purple) for the node pair s, t. Bottom: Two LHL query answering scenarios for s, t with $L(s) \cap L(t) = \{v, w\}$: In the left image (assuming uniform edge weights), the best lower bound is realized via w with $c(\pi(s,w)) - c(\pi(t,w)) = 3 - 2 = 1$ and the best upper bound is realized via v with $c(\pi(s,v)) + c(\pi(t,v)) = 1 + 1 = 2$. Here, the upper bound is tight. In the right image, the best upper bound via v is $3 + 2 = 5$ and the best lower bound via w is $4 - 2 = 2$. Here, the lower bound is tight.

between s and t. If the labels are stored sorted by ID of the contained nodes, the intersection of two labels can be computed in linear time. Thus, query answering only takes $\mathcal{O}(|L(s)| + |L(t)|) \in \mathcal{O}(L_{\max})$ where $L_{\max} := \max_{v \in V} |L(v)|$.

While HL allows for fast query answering, storing labels that meet the cover property often requires a substantial amount of space. Landmark Hub Labeling (LHL) is a generalization of HL which uses a weaker cover property and thus potentially allows for smaller label sizes [16]. The underlying observation is that if s and t both store the distance to a node v that is not on the shortest path from s to t but on a shortest path from a to b that contains both s and t, one can still deduce the correct shortest path distance between s and t, now by computing $|c(\pi(s,v)) - c(\pi(t,v))|$. Such a node v where $|c(\pi(s,v)) - c(\pi(t,v))| = c(\pi(s,t))$ is also called a perfect landmark for s, t. The relaxed cover property of LHL now demands that for each $s, t \in V$ their label intersection $L(s) \cap L(t)$ either contains a hub or a perfect landmark for s, t. The issue is that query answering becomes more intricate with LHL, as one needs to figure out whether a hub or a perfect landmark leads to the tight distance bound. To accomplish this, each node in $I := L(s) \cap L(t)$ is considered in both roles. This results in an upper bound $UB := \min_{v \in I} c(\pi(s,v)) + c(\pi(t,v))$ as well as a lower bound $LB := \max_{v \in I} |c(\pi(s,v)) - c(\pi(t,v))|$ on the shortest path distance $c(\pi(s,t))$ where either UB or LB needs to be tight, see Figure 1 for an illustration. To check if LB is tight, one needs to reconstruct the path from (w.l.o.g.) s to v up to a distance of LB and see whether t is found. If this is the case, $c(\pi(s,t)) = LB$ follows, otherwise it is certified that $c(\pi(s,t)) = UB$. To make the path traversal step efficient, a variant of LHL called path-consistent LHL (PC-LHL) was introduced in [16]. It allows to reconstruct the shortest path from a node s to one of its stored label nodes v in $\mathcal{O}(k)$ where k denotes the number on edges on that shortest path. However, this still results in distance query times of $\mathcal{O}(L_{\max} + D)$ where D denotes the diameter of the input graph. Depending on the structure of the input graph, the query time might thus be significantly larger than the $\mathcal{O}(L_{\max})$ query time of HL.

In this paper, we show that one can actually answer distance queries with a PC-LHL also in $\mathcal{O}(L_{\max})$ by storing and indexing the labels in a novel manner. Furthermore, we show that both in theory and practice substantial space savings are possible when imposing the weaker cover property of LHL instead of the stronger one required for HL.

1.1 Related Work

Hub Labeling (HL) has been studied extensively, both from a theoretical and practical perspective. Constructing a Hub Labeling (HL) with minimum total or average label size poses an NP-hard problem [2]. Interestingly, the complexity of computing a HL with minimum average label size is still open on trees. For this special case, a PTAS was described in [1]. In general graphs, an approximation algorithm with a factor of $\mathcal{O}(\log n)$ and a running time of $\mathcal{O}(n^3 \log n)$ is known [2, 8], as well as an $\mathcal{O}(\log D)$ approximation algorithm which is based on a LP-relaxation [1]. For practical applications, however, oftentimes fast heuristics are used instead of the slower and more intricate approximation algorithms. Usually, those construct a special type of HL, called a hierarchical HL (HHL) [7, 8, 12]. Here, nodes are ranked by some notion of importance and the node label of a node v can only contain nodes that have at least the same rank as v. Approximation algorithms were also studied for HHL, with the best known guarantee being in $\mathcal{O}(\sqrt{n} \log n)$ [2].

As storing the labels of a HL or a HHL requires a lot of memory, label compression techniques have been investigated [9]. Recently, Landmark Hub Labeling (LHL) has been introduced as a complementary way to reduce the label size [16]. It was shown that the $\mathcal{O}(\log n)$ approximation algorithm for HL can be adapted to also work for LHL. Furthermore, a $\mathcal{O}(\log n)$ approximation algorithm with a running time of $\mathcal{O}(n^6)$ was proposed for the path-consistent variant (PC-LHL) as well as a heuristic construction methods that run in $\mathcal{O}(n^3)$ [16, 17]. However, the query time analysis as well as a proof-of-concept-study revealed worse query times when using the PC-LHL framework instead of HL.

1.2 Contribution

In this paper, we provide numerous new structural insights into LHL.

We first carefully study the relationships between different LHL and HL variants. For example, we prove that for the hierarchical variants (HLHL and HHL), the optimal label size of the HLHL is always smaller than that of a HHL and there exist graphs where the factor between the respective label sizes is $\Theta(\sqrt{n})$. This label size gap even applies to HHL and PC-HLHL, where PC-HLHL denotes the practically relevant path-consistent HLHL variant. We show several further novel gaps and bounds, see Figure 2 for an overview of results on tree graphs, and Figure 4 for results on general graphs. Our new gaps illustrate the great potential of reducing the space consumption when using LHL instead of HL.

Next, we prove that computing an optimal PC-HLHL is NP-hard by reduction from HHL, which itself was proven to be NP-hard in [2]. To nevertheless see whether the theoretical space improvements we show for (PC-H)LHL over (H)HL also translate into practical improvements, we describe how to efficiently implement known and novel (approximation) algorithms. In an experimental study on diverse benchmark graphs, we demonstrate that label sizes of LHL are indeed consistently smaller than those of HL.

Finally, we tackle the most pronounced weakness of LHL with respect to HL, which is the query time of $\mathcal{O}(L_{\max} + D)$. The $+D$ term stems from the necessity to conduct a path traversal in the LHL query in order to check whether the computed lower or upper distance bound is tight. We describe a new method for storing labels of a (PC-)LHL such that no path traversal is needed anymore for this check. Accordingly, we can now match the $\mathcal{O}(L_{\max})$ query time of HL. We thus establish LHL as a viable alternative to HL, both with respect to space consumption and query time.

2 Preliminaries

Throughout the paper, we assume to be given an undirected graph $G = (V, E, c)$ with positive edge costs $c : E \to \mathbb{R}^+$ and a unique shortest path between any pair of nodes $s, t \in V$. The latter property can be ensured via symbolic perturbation of the edge costs. We use $\pi(s, t)$ to denote the shortest path between nodes $s, t \in V$ and $c(\pi(s, t)) = c(\pi(t, s))$ for its cost.

We already defined Hub labeling (HL) and Landmark Hub labeling (LHL) in the introduction. We now provide additional definitions for the labeling variants which are also studied in the remainder of the paper.

▶ **Definition 1** (Hierarchical Hub Labeling). *A HL L is a Hierarchical Hub Labeling (HHL) if there is a ranking $r : V \to \{1, \dots n\}$, with $n = |V|$, such that for every $v \in V$, all $w \in L(v)$ are such that $r(w) \geq r(v)$.*

For a fixed ranking r, one can find in polynomial time the HHL with minimum total label size which is called the canonical HHL.

▶ **Definition 2** (Canonical HHL). *Given a node ranking r, the canonical HHL is such that, for $v, w \in V$, $w \in L(v)$ if and only if there is a shortest path starting at v on which w has the highest rank.*

We can also define a hierarchical LHL (HLHL) in the same way as HHL by allowing all nodes to contain only nodes of higher or equal rank in their label. Another important property on labelings that we will explore is path-consistency.

▶ **Definition 3** (Path-Consistent Labeling). *A labeling is path-consistent (PC) if for each $w \in L(v)$, we also have $w \in L(u)$ for all u on a shortest path from v to w.*

In what follows, we will use PC-(L)HL to refer to a path-consistent (Landmark) Hub Labeling and PC-H(L)HL to denote a path-consistent hierarchical (Landmark) Hub Labeling.

It was shown in [16] that every canonical HHL is path-consistent. Given a ranking r, we can also define the canonical PC-HLHL, which is the PC-HLHL respecting r with the smallest labeling size [16].

▶ **Definition 4** (Canonical PC-HLHL). *Given a node ranking r, the canonical PC-HLHL is such that, for $v, w \in V$, $w \in L(v)$ if and only if there is a maximal shortest path containing v and w on which w has the highest rank.*

Finally, also the concept of a Landmark Labeling (LL) was defined in [16].

▶ **Definition 5** (Landmark Labeling). *A Landmark Labeling (LL) is a labeling $L : V \to 2^V$ which satisfies the landmark cover property, that is for every pair of nodes $s, t \in V$, there is a node $v \in L(s) \cap L(t)$ which is a perfect landmark for s, t.*

Clearly, LL is a special type of LHL just like HL.

3 Relationships between Labeling Variants

With the plethora of labeling variants that have been proposed in the literature, it is interesting to investigate how optimal label sizes behave with respect to each other. Obviously, label sizes of more general variants are at most as large as those of the respective special cases. But we are also interested in the size of the gaps that can arise between them.

In this section, we provide novel results on trees and general graphs, and summarize the known gaps between different LL, LHL, and HL classes in relationship diagrams.

3.1 On Trees

In [1], the authors show that the optimal HL on trees is hierarchical. Based on the same idea we can show that the optimal PC-LHL on trees is also hierarchical.

▶ **Theorem 6.** *For every tree $T = (V, E)$ and any valid PC-LHL L for T, there exists a hierarchical PC-LHL L' such that $|L'| \leq |L|$.*

Proof. The idea is to construct a hierarchical PC-LHL L' based on the given L without increasing the label size.

For each $u \in V$, let $L(u)$ be the label set of u, we define an induced subtree $T_u \subseteq T$ in the same way as in [1]: Let u be the root of T_u, there is a path $\pi(u, v) \in T_u$ if $v \in L(u)$. A vertex w belongs to T_u if there is a label $v \in L(u)$ such that $w \in \pi(u, v)$. We call this tree the label tree of u on T respects $L(u)$.

We now show that $T_u \cap T_v \neq \emptyset$, for every $u, v \in V$. Let $w \in L(u) \cap L(v)$, then $w \in T_u$ and $w \in T_v$. Due to *Helly Property* ([10],[11]), given a family of subtrees of T, if every two subtrees in the family intersect, then all subtrees intersect on at least one node r. To construct L', we first let $L' = L$. Secondly, we assign the node r the highest rank and add r into the label set L' of every node, then remove the labels $w \in L'(u)$ for $u \in T$, such that $r \in \pi(u, w)$. Now L' is still a valid PC-LHL with $|L'| \leq |L|$.

Let $T_1, \ldots, T_k$ be the subtrees rooted at the neighbors of r. The shortest paths between u, v are covered by r, where $u \in T_i, v \in T_j, i, j \in \{1, \ldots, k\}, i \neq j$. Therefore, we can deal with the subtrees $T_1, \ldots, T_k$ separately. Let T_i be one subtree rooted at the neighbor of r. Let $\pi(r, w)$ be the degree two path such that any node on the path, except r and w, has degree two, and the degree of w is not two. Let $u \in \pi(r, w) \setminus \{r, w\}$, then the shortest path from any node in T can be covered by r. We, therefore, remove the labels in $v \in L(u)$ such that $v \neq r$ and $v \neq u$. Now L' is still valid and $|L'| \leq |L|$. If T_i for $i \in \{1, \ldots, k\}$ is a path, we assign all nodes in T_i the lowest ranks, then $L'[T_i]$ is a valid hierarchical PC-LHL.

We obtain the subtrees that are not just a path. Let T' be a subtree after removing the degree 2 chain from r. Let $l_1, \ldots, l_p$ be the leaves in T'. Then $r \notin \pi(l_i, l_j)$ for $i \neq j$. There exists a node $u \in L(l_i) \cap L(l_j)$, and $u \in L(w)$ for all $w \in \pi(l_i, l_j)$. This means all nodes that lie on the path of two leaves contain a common label. Hence, as a pair of nodes always lies on a path from a leaf to another leaf, for any $u, v \in T'$, $L(u) \cap L(v) \neq \emptyset$. Therefore, the label trees $T'_u, T'_v \subseteq T'$ have an intersection. Again according to the *Helly Property*, all label trees have a common intersection. We assign the node the next highest rank.

For each subtree, we can apply this construction recursively, and the resulting L' is hierarchical and PC with $|L'| \leq |L|$. ◀

Since the optimal HHL is canonical and thus also PC, HHL is a special case of PC-HL. Hence, the optimal labeling of PC-HL on trees has the same size as HHL. It thus follows that it also has the same size as HL.

In Figure 2, we present the known optimal label size relations between different labelings on trees, and the blue color highlights the relations of our contribution in this work. On trees, the optimal LHL has the smallest size of any labeling. For a star graph with n nodes, the average label size in optimal LL is $\mathcal{O}(n)$, where in the optimal LHL it is constant. For a path, the optimal solution of HL has a maximum label size of $\mathcal{O}(\log n)$, while the optimal solution of LHL has a maximum size of $\mathcal{O}(1)$. The HLHL is a special labeling of LHL, but whether the optimal LHL on trees is hierarchical is still open. As we have shown that the optimal PC-LHL on trees is hierarchical and PC-HLHL is a special case of HLHL, we conclude that the optimal HLHL is at most as large as PC-LHL, PC-HLHL, and canonical PC-HLHL.

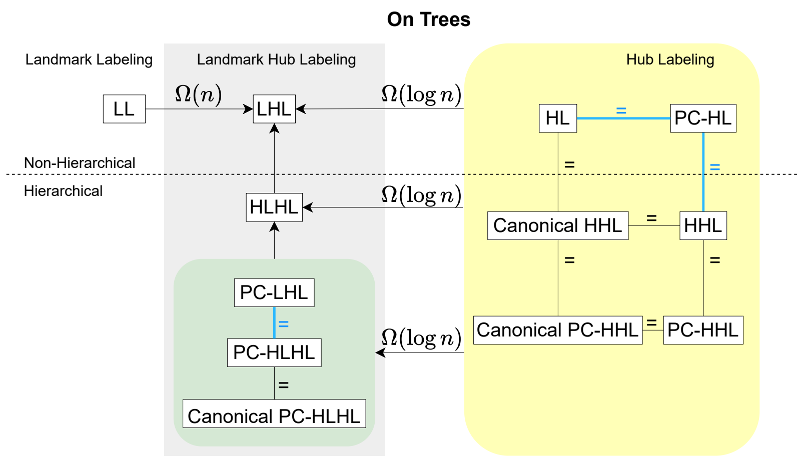

Figure 2 Relation of optimal solution between different labelings on trees. This diagram consists of three labelings, LL, LHL, and HL. In particular, we present the relations for optimal solutions on hierarchical and PC labeling of LHL and HL. The arrows point to the labeling whose optimal solution is always smaller or equal to the other one. The function classes on the arrow are the largest known factors of the optimal label size of the larger labeling to the smaller labeling on particular tree families. Equal means that both labelings have the same optimal label size on trees. The blue highlighted relations are our contributions.

3.2 On General Graphs

In [2] it was shown that the gap between HHL and HL can be in $\Omega(\sqrt{n})$. We prove that the optimal HL on the same instance satisfies the PC property. Thus, the gap between HHL to both PC-HL and PC-LHL is also at least a factor of $\Omega(\sqrt{n})$. Using the same example, we show that the gap between hierarchical and non-hierarchical (PC-)LHL is also at least a factor of $\Omega(\sqrt{n})$.

▶ **Theorem 7.** *There is a graph family for which the optimal HLHL size is $\Omega(\sqrt{n})$ times larger than the optimal label size of LHL.*

Proof. We take the same example as in [2] (that is the graph in Figure 3 without the b_i nodes) in which they prove that the gap between HL and HHL can be $\Omega(\sqrt{n})$. They exhibit a HL for the considered graph of total labeling size in $\mathcal{O}(n)$. Each node has to contain at least one label in the label set, thus the smallest summed label size of a graph without isolated nodes is in $\mathcal{O}(n)$. Since HL is also an LHL, the optimal LHL size is also in $\mathcal{O}(n)$. We show that the total size of any HLHL is $\Omega(n^{\frac{3}{2}})$. Now we consider a ranking of the nodes on the clique, let $r(c_1) < ... < r(c_k)$, and consider p_{ix} and c_j for $i < j$ and $1 \leq x \leq k$. The only shortest path containing p_{ix} and c_j is (p_{ix}, c_i, c_j). Since c_i cannot be in the label of c_j, then either p_{ix} is in the label of c_j or c_j is in the label of p_{ix}. This produces at least one label for each such pair. The total number of such pairs is:

$$k \sum_{i=1}^{k-1} i = \frac{k^2(k-1)}{2} \in \Omega(n^{\frac{3}{2}}).$$

Then the total label size of HLHL is $\Omega(n^{\frac{3}{2}})$. Hence, the gap between the optimal HLHL and the optimal LHL is $\Omega(\sqrt{n})$. ◀

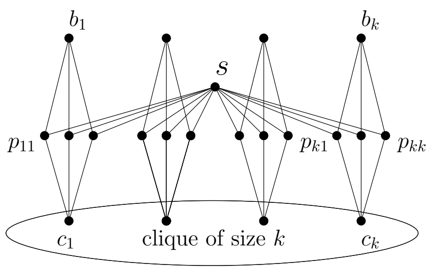

Figure 3 Example graph with $\Theta(\sqrt{n})$ label size gap between HHL and HLHL.

Now consider the gap between PC-HLHL and PC-LHL. It is easy to verify that the HL proposed in [2] is PC. Furthermore, PC-HLHL is an HLHL. It follows that in this graph family, the optimal PC-HLHL size is $\Omega(\sqrt{n})$ times larger than the label size of PC-LHL.

Note that the graph in Figure 3 without the b_i nodes contains non-unique shortest paths from c_i to s. However, one can assign each $\{c_i, p_{i1}\}$ and each $\{p_{i1}, s\}$ edge with a cost of $2/3$, and all other edges with a cost of one. Then the graph has unique shortest paths and the optimal label sizes of the discussed labelings are still the same.

We now modify their example and show the gap between the optimal label size of HHL and the optimal label size of (PC-)HLHL is $\Omega(\sqrt{n})$.

▶ **Theorem 8.** *There is a graph family for which the optimal HHL size is $\Omega(\sqrt{n})$ times larger than the optimal label size of (PC-)HLHL.*

Proof. Consider the undirected graph in Figure 3. The graph consists of k stars, denoted as $c_1, \ldots, c_k$. Each star has k leaves, denoted as $p_{i1}, \ldots, p_{ik}$ for a star with center c_i. The centers of the stars form a clique. The leaves of each star connect to another center denoted as $b_1, \ldots, b_k$. Finally, all leaves connect to the node s. The total number of nodes is $k^2 + 2k + 1$, and the length of every edge is 1.

Now consider the following HLHL for this graph. The nodes are sorted according to their ranking from high to low: $s, b_k, \ldots, b_1, c_k, \ldots, c_1, p_{kk}, \ldots, p_{11}$. The node s is in every label set. The label b_i is in the label set of nodes $p_{i1}, \ldots, p_{ik}$, $c_1, \ldots, c_k$, and b_i. And c_i is in the label set of node c_j where $j \leq i$. Finally, every p node adds itself to its label set. It is easy to verify the cover property and path consistency holds. Each of node s, b and p has a label size of $\mathcal{O}(1)$. The label size of every c node is in $\mathcal{O}(k)$. It follows that the total label size is $\mathcal{O}(n)$.

Now we show the optimal HHL size is in $\Omega(n^{\frac{3}{2}})$. W.l.o.g., let $r(c_1) < r(c_2) < \cdots < r(c_k)$. Consider any p_{ix} and c_j where $i < j$, the shortest path between them is $\pi(p_{ix}, c_j) = (p_{ix}, c_i, c_j)$. Since c_i is not in $L(c_j)$, then either p_{ix} in $L(c_j)$ or c_j in $L(p_{ix})$. Due to the same reason as in Theorem 7, the total label size of HHL is $\Omega(n^{\frac{3}{2}})$. Hence, the gap between the optimal (PC-)HLHL and the optimal HHL is $\Omega(\sqrt{n})$. ◀

In Figure 4, the label size bounds between different labels are shown for general graphs. Clearly, the gaps are at least as large as those for trees. LHL has the smallest optimal label size. Between the LHL and HL labels, the optimal HHL is the largest. There exists a graph family such that every optimal hierarchical labeling of LHL and HL is at least a factor of $\Omega(\sqrt{n})$ larger than the non-hierarchical one. The gap between HHL and (PC-)LHL is at least a factor of $\Omega(\sqrt{n})$ on a graph family that allows multiple shortest paths between two nodes. For unique shortest paths, the known gap is at least a factor of $\Omega(\log n)$ on paths.

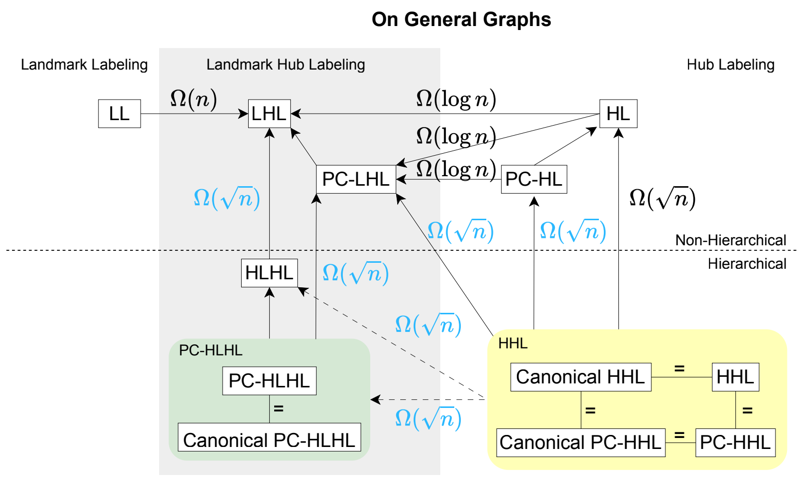

Figure 4 Relation of optimal solution between different labelings. The dashed arrows mean that the gap of the corresponding labelings is found on a graph family with non-unique shortest paths.

4 NP-Completeness of PC-HLHL

In this section, we argue that finding an optimal PC-HLHL is NP-complete by a reduction from HHL. The reduction takes an instance of HHL consisting of a graph $G = (V, E)$ and an integer k and constructs a graph G^+ and an integer k' such that the following conditions are equivalent: (i) There is an HHL of size k in G. (ii) There is an PC-HLHL of size k' in G^+. We prove the following theorem based on this reduction idea.

▶ **Theorem 9.** *The problem of deciding whether an undirected graph has a PC-HLHL of size at most k is NP-complete.*

Before showing the reduction, we first prove the following useful Lemmas. Note that for the cover property, we do not consider the pair of (v, v) for $v \in V$. Therefore, a node v is in its label set if and only if it is in the label set of other nodes.

▶ **Lemma 10.** *Given a graph $G = (V, E)$ in which all nodes in this graph have a degree of at least 3, and there is a unique shortest path between every two nodes. Let $G^+ = (V^+, E^+)$ be the graph obtained by connecting one single node to each node in G. Denote $v' \in V^+ \setminus V$ as the node added to the node $v \in V$, called leaf of v. There is an optimal PC-HLHL such that leaves have lower ranks than non-leaf nodes and no leaf is contained in any label set.*

Proof. Any maximal geodesics that v' lies on contains v too, and v' is always one of the end nodes. Let L be one optimal PC-HLHL on G^+, with the ranking r which is also a canonical PC-HLHL. It was shown in [16] that the canonical PC-HLHL is the minimum PC-HLHL respected to the rank r. We now construct a canonical PC-HLHL ranking r' such that all leaves have a lower rank than its neighbor and the respected labeling L' has a size at most as the size of L. For any $r(u') > r(u)$, let $r'(u) = r(u')$ and $r'(u') = r(u)$, we now show the total label set size does not change. Case one: For a node v such that $u' \in L(v)$, its label set does not increase. Since u lies on all the maximal geodesics that u' lies on. Then $L'(v) \subseteq L(v) \setminus \{u'\} \cup \{u\}$. Case two: For a node v such that $u \notin L(v)$, its label set does not increase either. Assume that after the ranking swap, there is a node v that $u \notin L(v)$, but now u is in $L'(v)$. Then it follows that u has now the highest rank on one maximal geodesic $\pi' \in \pi^+(v, u)$, where $\pi^+(v, u)$ is the set of maximal geodesics that contain both u and v. Let

s, t be the end nodes of the maximal geodesic and $\pi' = \pi(s,v) \cup \pi(v,u) \cup \pi(u,t)$. Then u has now the highest rank on the maximal geodesic $\pi(s,u')$. Then u' has the highest rank on it in the rank r. It follows that u' must be in $L(v)$. It is a contradiction. Therefore, for a node v, if $u' \notin L(v)$, then $u \notin L'(v)$, their label set size does not increase either. We update the rank for such leaves using the same methods until every leaf has a rank lower than their neighbor, and $|L'| = |L|$. It follows that any label set contains no leaf. Furthermore, one can assign r' in the way that all leaves have lower ranks than the nodes in V and it is still optimal. ◀

▶ **Lemma 11.** *Given a graph $G = (V,E)$ and $G^+ = (V^+, E^+)$ as described in Lemma 10. Let L be an optimal PC-HLHL such that all leaves have lower ranks than the nodes in V, then for a leaf v', its label set $L(v') = L(v)$.*

Proof. Let L be an optimal PC-HLHL such that all leaves have lower ranks than the nodes in V. Due to path consistency, for a leaf v', $L(v') \subseteq L(v)$. Now we show $L(v) \subseteq L(v')$. Assume that there is a label $w \in L(v)$ and $w \notin L(v')$. Since $w \notin L(v')$, w does not have the highest rank on any maximal geodesics end by node v', and any shortest path reaches v is a part of one maximal geodesics end by v'. Therefore, on any maximal geodesics that contains v, w does not have the highest rank. According to canonical property, $w \notin L(v)$. This leads to a contradiction. Therefore $L(v) \subseteq L(v')$ and $L(v) = L(v')$. ◀

▶ **Lemma 12.** *Given a graph $G = (V,E)$ and $G^+ = (V^+, E^+)$ as described in Lemma 10. Let L be an optimal PC-HLHL on G^+ such that all leaves have lower ranks than the nodes in V, and $L(G^+[G])$ be the sublabeling of a PC-HLHL in G^+ that is induced by the graph G, then $L(G^+[G])$ is an HHL for nodes in V.*

Proof. Let L be an optimal PC-HLHL such all leaves have lower ranks than non-leaf nodes. We show that there is no landmark in $L(G^+[G])$. Assume that there is a landmark $w \in L(u) \cap L(v) \cap \pi^+(u,v)$, and $L(u) \cap L(v) \cap \pi(u,v) = \emptyset$. It follows $L(u') \cap L(v') \cap \pi(u',v') = \emptyset$. However, L is a valid PC-HLHL. This leads to a contradiction. Therefore, $L(u) \cap L(v) \cap \pi(u,v) \neq \emptyset$ for $u, v \in V$. Hence, $L(G^+[G])$ is an HHL for G. ◀

▶ **Lemma 13.** *Given a graph $G = (V,E)$ and $G^+ = (V^+, E^+)$ as described in Lemma 10. The graph G has an HHL of size k if and only if G^+ has a PC-HLHL of size $2k$.*

Proof. Let $|V| = n$, $|E| = m$. It follows $|V^+| = 2n$ and $|E^+| = m + n$. Assume G has a canonical HHL L of size k with the rank r. We construct a canonical PC-HLHL of G^+ as follows. We assign each non-leaf node v in G^+ with the rank of $r(v) + n$. And assign the rank of leaves randomly from 1 to n. The label set of non-leaf stays the same, and for each leaf v' let $L(v') = L(v)$. Since all maximal geodesics which end by a leaf v' contain the node v, and all shortest paths reaching node v are covered in the HHL, therefore all maximal geodesics are also covered by assigning $L(v') = L(v)$. And the canonical HHL is path-consistent. Hence, the constructed labeling is a canonical PC-HLHL with a size of $2k$.

Assume that L is an optimal PC-HLHL of G^+ of size $2k$. Let L be an optimal PC-HLHL L such all leaves have lower ranks than the nodes in V. According to Lemma 11 and Lemma 12, $L(G^+[G])$ is an HHL for nodes in V with $|L(G^+[G])| = k$. Assuming that $L(G^+[G])$ is not an optimal HHL, then we can construct a PC-HLHL for G^+ with a smaller HHL, then L is not optimal. It is a contradiction. Therefore, the optimal HHL size is k. ◀

5 Construction Algorithms

In [6], the authors formulate HL as a weighted set cover problem (U, S) so that one can use a greedy algorithm to compute an $\mathcal{O}(\log n)$ approximation of the optimal solution. The universe U is a set of uncovered node pairs, S is the collection of all subsets of uncovered pairs that can be covered by the same node. They introduce the center graph of a node v for the undirected graph as follows. In the center graph $G_v = (V', E_v)$, for all $u, w \in V$ there is an edge between node u and w if and only if they are not yet covered and v lies on the shortest path between them. The center graph does not contain isolated nodes. The greedy algorithm chooses the maximal density subgraph (MDS) of all center graphs and adds the center to the label set of the nodes in the MDS. It removes the covered pairs from U and repeats until all pairs are covered. This greedy algorithm can also be applied to LHL [16]. The difference from the HL formulation is that for a center graph G_v, there is an edge between uncovered pair (u, w), if and only if v lies on some shortest path containing u and w. The $\mathcal{O}(\log n)$ approximation factor still holds. We call this algorithm w-LHL.

5.1 Improvements

A naive way to compute the LHL using the greedy algorithm is to precompute all maximal shortest paths (also called geodesics). To construct the center graph of a node v, one must check for each uncovered pair (s, t) whether v lies on one of their maximal geodesics. This can be done by iterating over all maximal geodesics containing v, and checking whether they also contain s and t. However, this takes linear time to check. This implementation costs $\mathcal{O}(n^3)$ space and $\mathcal{O}(n^5)$ running time. We propose two improvements for computing the greedy algorithm efficiently. The improved algorithm needs only $\mathcal{O}(n^2)$ space and has a running time of $\mathcal{O}(n^4 \log n)$. However, it is faster in practice.

The first improvement is to efficiently check whether a node can cover a pair. Instead of storing all maximal geodesics, one can use a distance matrix to store only the shortest path distances. This reduces the space consumption from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Given a node v and a pair (s, t), v is a hub or a perfect landmark if and only if $c(s, v) + c(v, t) = c(s, t)$ or $c(s, t) + c(v, t) = c(s, v)$ or $c(s, v) + c(s, t) = c(v, t)$. Therefore, checking whether a node covers a pair can be done in constant time. The second improvement is to avoid unnecessary recomputation of the MDS of a node's center graph every round by using "lazy updates". Note that the MDS of a graph cannot be increased by removing nodes or edges in the graph. We use a max-heap to store for each node the density of the MDS of its center graph. In each iteration, a threshold d is initially set to $-\infty$. While the node v at the top has a corresponding stored density of at least d, we recompute the density md of its new center graph and update $d = \max(d, md)$. Then, the node is pushed back to the heap. Note that the density may decrease because some pairs may be covered from the last round. Finally, we cover the pairs in the best MDS with the respective center node. For the nodes with lower density of the center graph, the MDS is not updated every round to avoid the redundant calculation. The total running time is now in $\mathcal{O}(n^4 \log n)$.

5.2 PC-Labeling

The resulting label from the greedy algorithm is not necessarily PC. Given a valid LHL, one can produce a PC-LHL in polynomial time. We run Dijkstra for each node v until it reaches all nodes w with $v \in L(w)$. Then we check for the resulting shortest path tree whether there is a node u such that $v \notin L(u)$. If so, we add v into the label set of u, thus ensuring PC.

5.3 Hierarchical LHL

In [2], authors propose the weighted greedy HHL algorithm, which selects the whole center graph of maximum density instead of its MDS. Since every center graph is only chosen once, the resulting labeling is hierarchical. We now also select the whole center graph for hierarchical LHL. This algorithm is called w-HLHL. Another greedy algorithm is selecting the center graph which has the most edges. We call this g-(H)LHL. Note that w-HLHL and g-(H)LHL are not necessarily PC. For w-HLHL and g-(H)LHL, one can also apply the “lazy updates”. The heap now stores the density of the whole center graph for w-HLHL or the number of edges for g-(H)LHL. The respective running times are in $\mathcal{O}(n^3 \log n)$.

6 Improved Query Answering

In an LHL distance query for nodes $s, t \in V$, we first compute the lower and the upper distance bound LB and UB and then check whether LB is tight by traversing the path from (w.l.o.g.) s to the assumed perfect landmark node v up to distance LB. If we detect t, a path from s to t with cost equal to LB is found and thus this is the correct distance value. The path traversal from s to v can easily dominate the query time.

To improve the query time, we need a more efficient way to characterize whether v is indeed a perfect landmark for s, t. To accomplish this, we now assume that the shortest path tree T_v emerging from v is known. We furthermore denote with $LCA_v(s,t)$ the lowest common ancestor of s and t in the shortest path tree rooted at node v. The following lemma describes the connection between T_v and the role of v for s, t in an LHL query.

▶ **Lemma 14.** *The node v is a perfect landmark for $s, t \in V$ if and only if $LCA_v(s,t) = s$ or $LCA_v(s,t) = t$. The node v is a hub for $s, t \in V$ only if $LCA_v(s,t) = v$.*

Proof. To be a perfect landmark, there needs to be a shortest path from w.l.o.g. s to v over t. Thus, in T_v, t has to be an ancestor of s and $\text{LCA}_v(s,t) = t$ follows. To be a hub, the shortest path from s to t needs to go over v and thus it follows $\text{LCA}_v(s,t) = v$. These observations hold because we have a unique shortest path between each pair of nodes. ◀

It was shown in [5] that for a given tree graph T_v, an LCA data structure can be computed in linear space and time that allows to answer $\text{LCA}_v(s,t)$ queries in $\mathcal{O}(1)$. Below, we discuss how to leverage the lemma and this data structure to improve LHL query times without increasing the asymptotic space consumption and to efficiently check whether a given labeling fulfills the weak cover property.

6.1 An Alternative Method for Label Storage

Conventionally, each node simply stores its labels as a list of nodes with corresponding distance values (sorted by node ID to enable efficient computation of label intersections). But to make use of Lemma 14, we need a label-centered perspective. We thus propose the following alternative way to store label information: Each node $v \in V$ stores a partial shortest path tree T_v^* defined as the smallest subtree of T_v that contains all nodes w with $v \in L(w)$. Shortest path distances are stored along with the tree nodes as well as a pointer to the root node. For each T_v^*, we compute an LCA data structure as described above.

The nodes then additionally hold a list of pointers to their positions in these partial shortest path trees, sorted by the ID of the respective root node. On query time, the lists of s and t are processed and common root node IDs are identified in linear time. For each such root node v, we then query the LCA data structure to check whether v is a perfect landmark for s, t according to Lemma 14. If that is the case, we can immediately abort the procedure and return the respective distance. Otherwise, we check whether $LCA_v(s,t) = v$ and if that is the case, update the UB value (initially set to ∞) if necessary. If no perfect landmark was detected, we return the UB value at the end which is now certified to be correct. We thus get the following corollary.

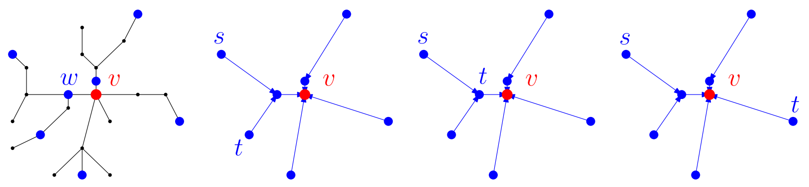

Figure 5 In the left image, the shortest path tree from node v (marked red) is depicted and nodes w with $v \in L(w)$ are marked blue. In the three images to its right, the compressed tree is shown and different query answering scenarios are indicated. In the first, $LCA_v(s,t)$ is neither s,t nor v and thus v is not a perfect landmark or a hub for s,t. In the second, $LCA_v(s,t) = t$ and thus v is a perfect landmark. In the third, $LCA_v(s,t) = v$ and thus v is a potential hub for s,t.

▶ **Corollary 15.** *Distance queries can be answered with LHL in $\mathcal{O}(L_{\max})$.*

However, compared to simply storing $L(v)$ with every node $v \in V$, we might have increased the space consumption significantly by also storing T_v^*. For PC-LHL, this is luckily not the case as shown in the next lemma.

▶ **Lemma 16.** *In a PC-LHL, we have $\sum_{v \in V} |L(v)| = \sum_{v \in V} |T_v^*|$.*

Proof. The PC property demands that for a node $v \in L(w)$ also all nodes u on the shortest path from w to v have v in their label $L(u)$. Therefore, if we consider T_v^*, we know that for each $u \in T_v^*$ we also have $v \in L(u)$. The lemma follows. ◀

Accordingly, the space consumption is at most doubled by the alternative method to store the label information. But as $L(v)$ is replaced by a list of pointers, it demands at most half of the space than it takes to store the node ID and the distance explicitly, reducing the overhead further. Moreover, with T_v^* being stored, we can also answer path queries very efficiently by simply traversing the respective tree edges. To answer path queries with (H)HL efficiently, either substantially larger query times are necessary or additional information needs to be stored with every label [16]. With our method we can guarantee to answer path queries in $\mathcal{O}(L_{\max} + k)$ where k denotes the number of edges on the shortest path.

For non-PC LHL, the summed size of the partial shortest path trees T_v^* might indeed be much larger than the summed label sizes. However, we can use the same trick as proposed in [9] (there applied to the labels themselves for later compression) to also ensure $\sum_{v \in V} |L(v)| = \sum_{v \in V} |T_v^*|$ as for PC-LHL. The idea is simply to only keep nodes $w \in T_v^*$ with $v \in L(w)$ and insert shortcuts between $w, w' \in T_v^*$ if the shortest path $w, .., w', ..v$ is part of T_v but for all nodes u on the shortest path from w to w' it yields $v \notin L(u)$. Figure 5 illustrates this concept as well as how to use the compressed T_v^* for query answering.

So while for the query answering routine described in [16] the PC-property was vital to achieve a query time in $\mathcal{O}(L_{\max} + D)$, we can now guarantee query times in $\mathcal{O}(L_{\max})$ even for non-PC LHL based on our novel method to store the label information.

6.2 LHL Verification

With our improved query answering routine described above, we can also efficiently check whether a given labeling L is a feasible LHL. We compute the shortest path trees rooted at v for all $v \in V$ and the respective LCA data structures. For each node pair s, t, we can then check in T_s the correct distance and compare it to the distance obtained by running the LHL query algorithm. If these do not match for some pair, L is not a valid LHL. This check procedure requires $\mathcal{O}(n^2)$ space, a preprocessing phase in $\mathcal{O}(nm + n^2 \log n)$ and a running time for the verification in $\mathcal{O}(n^2 L_{max})$.

Table 1 Benchmark set summary.

Benchmark Set	# Instances	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert E\rvert/\lvert V\rvert$
OSM	12	[100, 2000]	[99, 2132]	1
PACE	100	[10, 491]	[15, 4100]	[1, 32]

7 Experiments

We implemented the approximation algorithms described in Section 5 in C++, including the lazy variants. The primary goal of the evaluation is to compare HL, HHL, (PC-)LHL and HLHL label sizes in practice. As shown in Table 1, we use road networks extracted from OpenStreetMap[1] (OSM) as well as the PACE challenge 2020 benchmark [2]. The PACE challenge 2020 benchmark contains a diverse set of graphs (100 instances in total) with number of nodes up to 500 nodes and density up to 32. Due to the large running times of the approximation algorithms, we restricted our tests to road networks with up to 2000 nodes (12 graphs in total). Experiments were conducted on a single core of an AMD Ryzen Threadripper 3970X (3.6 GHz) with 128 GB main memory. The time-out was set to 20 minutes per instance.

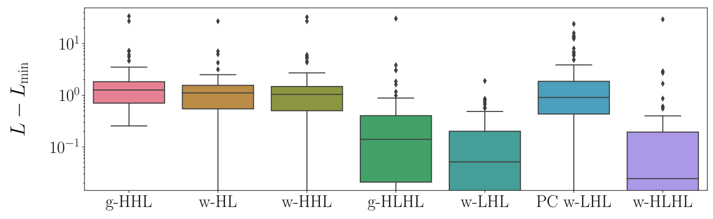

Figure 6 Label size comparison on the PACE benchmark set. The depicted value is the difference between the average label size and the minimal average label size among the solutions of all algorithms. Note the logscale of the y-axis.

Figure 6 and Table 2 show the label size comparison on the PACE benchmark set. In compliance with our theoretical results, the LHL sizes are in general significantly smaller than the HL sizes. In 95 out of 100 instances, the smallest label size is returned from one of the (H)LHL algorithms. The relative difference of HL to LHL size is up to a factor of 34.01 (on a graph with a density of 15). For instances with a density less than 2, w-HLHL computes on average smaller label sizes than w-LHL, which is somewhat surprising. For instances with higher density, w-LHL performs best overall. The PC-LHL constructed with our augmentation algorithm is smaller than w-HL in 51 out of 100 instances but we still observe that enforcing this property increases the label size significantly compared to the other LHL variants. Thus, our new query answering routine described in Section 6, which does no longer rely on the PC-property, is very useful to also keep the label size small.

Figure 7 and Table 3 show the results on the OSM benchmark set. We observe similar trends as for the PACE instances. The LHL algorithms consistently produce smaller label sizes. In fact, w-HLHL computes the smallest labeling in all instances. On average, each

[1] https://www.openstreetmap.org
[2] https://pacechallenge.org/2020/td

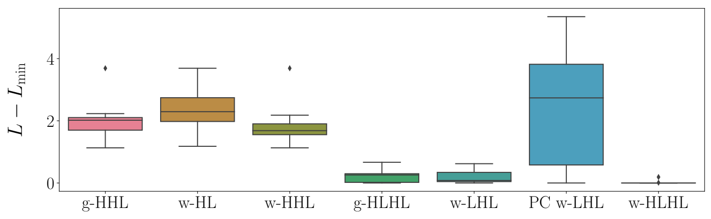

Figure 7 Label size comparison on the OSM benchmark set. The depicted value is the difference between the average label size and the minimal label size among the solutions of all algorithms.

node has at least two more labels in HLs than in LHLs. While this does not sound like a lot, we remark that with every decrease of the average label size by 1 we save space proportional to the size of the graph. The relative difference is up to a factor of 3, even on these rather small instances. We expect this gap to grow further on larger instances. Again, enforcing the PC property leads to larger label sizes and thus should be avoided. Alternatively, there might be other construction algorithms that take the PC property directly into account and thus produce smaller PC-LHL.

Regarding the construction time, the w-HLHL algorithm without lazy updates could not compute the result within 20 minutes on the four largest road network instances, while the engineered one finished in time on all of them. On the large PACE instances, the lazy variant was up to a factor of 3.75 faster.

8 Conclusions & Future Work

We provided new structural insights into LHL, including label size gaps towards the respective HL variants, hardness results, and improved query routines. There are several directions for future work. Our experimental results indicate that w-HLHL performs better than w-LHL on graphs of low density. It would be interesting to provide a theoretical explanation for this. From a practical perspective, there is a clear demand for LHL construction algorithms that scale well with the graph size. Our focus was on approximation algorithms to have a somewhat fair comparison between label sizes. However, most algorithms used in practice are fast heuristics. For LHL, so far only heuristics for the path-consistent variant were investigated [17]. With our alternative method to store labels, there is no longer the need to ensure the PC property, which should allow for easier construction and smaller label sizes. Moreover, there is room for further improving the space consumption. The method for compressing HL described in [9] relies on storing labels as trees and reducing space by identifying common tree structures. This approach should be transferable to LHL, where we store inverse label trees instead. Furthermore, our LCA-based query routine allows for stopping early if a perfect landmark is identified for the query node pair s, t. Thus, assigning node IDs in an LHL such that perfect landmarks are identified early for many node pairs could decrease the query time in practice. This is not possible in an HL query, where always all labels in $L(s)$ and $L(t)$ need to be inspected to ensure that the tightest upper distance bound is identified.

References

1 Haris Angelidakis, Yury Makarychev, and Vsevolod Oparin. Algorithmic and hardness results for the hub labeling problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1442–1461. SIAM, 2017.

2 Maxim Babenko, Andrew V Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller. On the complexity of hub labeling. In *International Symposium on Mathematical Foundations of Computer Science*, pages 62–74. Springer, 2015.

3 Jaime Barceló, Hanna Grzybowska, and Sara Pardo. Vehicle routing and scheduling models, simulation and city logistics. *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies*, pages 163–195, 2007.

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

5 Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.

6 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

7 Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Robust exact distance queries on massive networks. *Microsoft Research, USA, Tech. Rep*, 2, 2014.

8 Daniel Delling, Andrew V Goldberg, Ruslan Savchenko, and Renato F Werneck. Hub labels: Theory and practice. In *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 259–270. Springer, 2014.

9 Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hub label compression. In *International symposium on experimental algorithms*, pages 18–29. Springer, 2013.

10 PC Gilmore. Families of sets with faithful graph representation. *IBM Research Note NC*, 184, 1962.

11 András Gyárfás and Jenö Lehel. A helly-type problem in trees. *Combinatorial Theory and its applications*, 4:571–584, 1970.

12 Kartik Lakhotia, Qing Dong, Rajgopal Kannan, and Viktor Prasanna. Planting trees for scalable and efficient canonical hub labeling. *arXiv preprint arXiv:1907.00140*, 2019.

13 Sara Nazari, M Reza Meybodi, M Ali Salehigh, and Sara Taghipour. An advanced algorithm for finding shortest path in car navigation system. In *2008 First International Conference on Intelligent Networks and Intelligent Systems*, pages 671–674. IEEE, 2008.

14 Kali Prasad Nepal and Dongjoo Park. Solving the median shortest path problem in the planning and design of urban transportation networks using a vector labeling algorithm. *Transportation Planning and Technology*, 28(2):113–133, 2005.

15 Sven Peyer, Dieter Rautenbach, and Jens Vygen. A generalization of dijkstra's shortest path algorithm with applications to vlsi routing. *Journal of Discrete Algorithms*, 7(4):377–390, 2009.

16 Sabine Storandt. Algorithms for landmark hub labeling. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

17 Sabine Storandt. Scalable landmark hub labeling for optimal and bounded suboptimal pathfinding. In Kate Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 6788–6795. International Joint Conferences on Artificial Intelligence Organization, August 2024. Main Track. doi:10.24963/ijcai.2024/750.

A Detailed Experimental Data

Table 2 This table presents a random sample of 30 results, chosen from a total of 100, which describe the average label sizes $|L|$ and running times t(s) of the output produced by each algorithm tested on the PACE challenge benchmark set. The largest label size among them is indicated in italic font, while the smallest label size among them, which is denoted as $L_{\min}$ in Figure 6, is presented in bold font. In column w-LHL, the values of PC represent the number of labels that have increased when the results of w-LHL are transformed into PC w-LHL.

	g-HHL		**w-HL**		**w-HHL**		**g-HLHL**		**w-LHL**			**w-HLHL**	
	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	PC	$\lvert L\rvert$	t(s)
e003	*4.9*	≤ 1	*4.9*	≤ 1	*4.9*	≤ 1	**4.1**	≤ 1	**4.1**	≤ 1	0.2	**4.1**	≤ 1
e017	*5.5*	≤ 1	5	≤ 1	5.3	≤ 1	4.8	≤ 1	4.9	≤ 1	0.8	**4.8**	≤ 1
e027	*5.4*	≤ 1	4.8	≤ 1	**4.7**	≤ 1	5	≤ 1	**4.7**	≤ 1	0.7	**4.7**	≤ 1
e029	*5.9*	≤ 1	5.8	≤ 1	5.8	≤ 1	5	≤ 1	**4.9**	≤ 1	0.1	**4.9**	≤ 1
e031	*4.6*	≤ 1	4.5	≤ 1	4.4	≤ 1	4.2	≤ 1	4.1	≤ 1	0.3	**4**	≤ 1
e033	*5.6*	≤ 1	5.4	≤ 1	5.1	≤ 1	5.4	≤ 1	**4.9**	≤ 1	0.3	5.2	≤ 1
e043	*6.8*	≤ 1	6.2	≤ 1	6.6	≤ 1	5.9	≤ 1	**5.6**	≤ 1	0.6	5.8	≤ 1
e047	*9.7*	≤ 1	8.4	≤ 1	8.8	≤ 1	7.7	≤ 1	**7**	≤ 1	0.9	7.2	≤ 1
e059	*14.8*	≤ 1	11.9	≤ 1	12.9	≤ 1	8.3	≤ 1	**7.6**	≤ 1	0.1	8	≤ 1
e065	*6.2*	≤ 1	5.9	≤ 1	5.8	≤ 1	5.6	≤ 1	5.4	≤ 1	0.8	**5.3**	≤ 1
e071	*10.2*	≤ 1	6.9	≤ 1	7.2	≤ 1	8.5	≤ 1	**6.7**	≤ 1	0.4	7.2	≤ 1
e073	*7.8*	≤ 1	7.2	≤ 1	7.5	≤ 1	7.3	≤ 1	**6.7**	≤ 1	1	6.9	≤ 1
e075	*11.4*	≤ 1	9.8	≤ 1	10.9	≤ 1	9.4	≤ 1	**8.8**	≤ 1	2.1	9.1	≤ 1
e083	*10.1*	≤ 1	9.1	≤ 1	9.1	≤ 1	8.8	≤ 1	**8.3**	≤ 1	1.8	8.6	≤ 1
e085	*8.4*	≤ 1	8.3	≤ 1	8	≤ 1	6.9	≤ 1	7.2	≤ 1	1.4	**6.9**	≤ 1
e087	5.8	≤ 1	*6*	≤ 1	5.8	≤ 1	5.9	≤ 1	**5.5**	≤ 1	0.2	5.9	≤ 1
e089	*5.4*	≤ 1	5.3	≤ 1	5.3	≤ 1	5.1	≤ 1	**4.8**	≤ 1	0.2	5.1	≤ 1
e111	*9.2*	≤ 1	8.1	≤ 1	8.5	≤ 1	7.5	≤ 1	**6.8**	1	0.6	7	≤ 1
e117	*9.9*	≤ 1	9.4	1	9.4	≤ 1	9.3	≤ 1	**9**	1	1.8	**9**	≤ 1
e133	*7.5*	1	*7.5*	1	7.4	≤ 1	6.7	≤ 1	**6.4**	1	0.7	6.6	≤ 1
e137	*33.8*	1	33.6	2	33.7	1	7.1	1	**6.4**	2	0.3	6.7	1
e143	8.4	1	*8.6*	1	8.2	≤ 1	**7.6**	≤ 1	**7.6**	1	0.9	**7.6**	≤ 1
e151	5	≤ 1	*5.5*	1	5	≤ 1	4.5	≤ 1	**4.3**	1	0.2	4.5	≤ 1
e153	7.9	1	*8.1*	1	7.9	≤ 1	7.1	1	**6.9**	2	0.9	**6.9**	1
e159	*8.5*	1	*8.5*	1	8.2	1	**7.3**	1	7.5	2	0.7	**7.3**	1
e173	8.5	2	*9.1*	2	8.4	1	7.2	1	7.4	3	1.1	**7.1**	1
e183	*54.9*	19	21.2	8	53.4	2	51.4	6	**20.8**	12	0.3	50.4	6
e185	*9.6*	5	9.5	4	9.5	2	7.9	2	7.9	6	1.1	**7.8**	2
e189	5.3	4	*5.4*	4	5.3	3	4.6	3	4.7	5	0.3	**4.6**	3
e193	*12.9*	26	12.8	13	12.6	6	11.2	7	**10.8**	20	1.2	11.1	7

Table 3 This table presents the average label sizes $|L|$ and running times t(s) of the output produced by each algorithm tested on the benchmark set OSM. The largest label size among them is indicated in italic font, while the smallest label size among them, which is denoted as $L_{\min}$ in Figure 7, is presented in bold font. In column w-LHL, the values of PC represent the number of labels that have increased when the results of w-LHL are transformed into PC w-LHL.

	g-HHL		**w-HL**		**w-HHL**		**g-HLHL**		**w-LHL**			**w-HLHL**	
	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	$\lvert L\rvert$	t(s)	PC	$\lvert L\rvert$	t(s)
1	*5.7*	≤ 1	*5.7*	≤ 1	*5.7*	≤ 1	**2.0**	≤ 1	**2.0**	1	0	**2.0**	≤ 1
2	4.7	≤ 1	*4.8*	≤ 1	4.7	≤ 1	3.6	≤ 1	3.6	≤ 1	0.3	**3.5**	≤ 1
3	5.3	≤ 1	*5.5*	≤ 1	5.4	≤ 1	**4.0**	≤ 1	**4.0**	1	0.7	4.4	≤ 1
4	*5.2*	≤ 1	5.1	≤ 1	5.1	≤ 1	**2.9**	≤ 1	3.0	1	0.3	3.0	≤ 1
5	9.5	72	*9.9*	136	9.4	68	8.1	84	8.5	285	4.7	**7.9**	82
6	10.3	74	*10.4*	148	10.2	69	**8.5**	84	8.9	315	4.2	8.7	84
7	9.4	73	*9.5*	146	9.3	68	7.7	82	7.5	311	2.8	**7.4**	83
8	8.9	74	*9.2*	161	8.7	71	7.1	87	7.1	363	2.4	**6.9**	88
9	12.3	484	*13.6*	848	12.0	437	10.8	557				**10.5**	523
10	12.4	494	*13.2*	899	12.3	445	11.0	587				**10.3**	532
11	11.4	475	*11.9*	856	11.0	438	9.7	541				**9.6**	530
12	10.0	489	*10.4*	1019	10.0	461	8.1	584				**7.8**	572