Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 29, no. 2, pp. 103–126 (2025)
DOI: 10.7155/jgaa.v29i2.3041

# Exact and Approximate Hierarchical Hub Labeling

*Justine Cauvi*[1,2] *Ruoying Li*[2] *Sabine Storandt*[2]

[1]École Normale Supérieure de Lyon, France
[2]Department of Computer and Information Science,
University of Konstanz, Konstanz, Germany

Submitted: June 2024 Accepted: March 2025 Published: May 2025

Article type: Regular paper Communicated by: R. Uehara, K. Yamanaka, and H.-C. Yen

**Abstract.** Hub Labeling (HL) is a state-of-the-art technique for accelerating shortest path computation in road networks. By utilizing precomputed node labels, it can answer distance queries in microseconds on continent-sized networks. The optimization goal is to get correct query results with a minimum number of labels. There is an $\mathcal{O}(\log n)$ approximation algorithm for the size of an HL with a running time of $\mathcal{O}(n^3 \log n)$. However, existing practical implementations rely mostly on heuristics for a special type of HL, so called Hierarchical HL (HHL). Deciding whether a graph admits a labeling of size at most $k$ is NP-hard for both HL and HHL. For HHL, an $\mathcal{O}(\sqrt{n} \log n)$ approximation algorithm (called w-HHL) is known. In this article, we devise an exact HHL algorithm for general graphs. We also show that the exact algorithm transfers to Hierarchical Landmark HL (HLHL), which is a generalization of HHL. Moreover, we prove that w-HHL provides a constant factor approximation on trees and investigate for the first time the practical performance of HHL approximation algorithms. We also compare the resulting label sizes to heuristic HHL and HLHL. Our experimental results offer novel insights and show that commonly used methods for HHL are noticeably outperformed by w-H(L)HL on general graphs as well as trees.

## 1 Introduction

Efficient shortest path planning plays a pivotal role in a multitude of applications, particularly in road networks but also in other types of graphs. In road networks, for instance, it is fundamental for navigation systems [19], transportation logistics [6], and urban planning [20]. Beyond road networks, shortest path computation is an important ingredient, e.g. in social network analysis [21], data package routing in computer networks [27], or VLSI design [23]. Oftentimes, multiple

*E-mail addresses:* justine.cauvi@ens-lyon.fr (Justine Cauvi) ruoying.li@inf.uni-konstanz.de (Ruoying Li) sabine.storandt@inf.uni-konstanz.de (Sabine Storandt)

shortest path queries need to be issued on the same input graph. To facilitate fast query answering, preprocessing based techniques are commonly applied.

Hub Labeling (HL) is a state-of-the-art technique which provides excellent query times on various types of graphs [9, 12, 11]. Given a weighted graph $G(V, E, c)$ with $c : E \to \mathbb{R}_+$, an HL assigns to each node a label $L : V \to 2^V$ such that the so called *cover property* is fulfilled, that is, for all connected pairs $s, t \in V$, $L(s) \cap L(t) \cap \pi(s, t) \neq \emptyset$ where $\pi(s, t)$ denotes the set of nodes on the shortest path from $s$ to $t$ in $G$. If all nodes $v \in V$ store the shortest path distance $c_v(w)$ to each node $w \in L(v)$, the cover property allows for computing the distance between nodes $s$ and $t$ via the formula: $\min_{v \in L(s) \cap L(t)}(c_s(v) + c_t(v))$. To compute the set $L(s) \cap L(t)$ efficiently, nodes in $L(v)$ are presorted by node ID. Then, the intersection of any two such sets can be computed in time linear in their sizes by a merge-like procedure. For convenience, we assume that all nodes contain themselves in their label set.

Thus, the performance of HL solely depends on the size of the respective labels. Let $L_{\text{avg}} := \frac{1}{n} \sum_{v \in V} |L(v)|$ denote the average label size. Then the space needed to store an HL is in $\mathcal{O}(nL_{\text{avg}})$ and the average query time over all node pairs is upper bounded by $2L_{\text{avg}}$. Accordingly, the natural optimization goal is to compute an HL with the average node label size as small as possible. Minimizing $L_{\text{avg}}$ is an NP-hard problem [4]. Hence efficient heuristics that produce moderate label sizes are used in practice. In particular, heuristics for a special type of HL, a so called Hierarchical HL, are most prominent.

**Definition 1 (Hierarchical Hub Labeling (HHL))** *A hub labeling $L$ is a hierarchical hub labeling if there exists a bijective ordering (rank) $r : V \to \{1, \ldots, n\}$ for $n = |V|$, such that for all $v, w \in V$, $w$ can only be added into the label set $L(v)$ if $r(w) \geq r(v)$.*

In other words, in an HHL, nodes have ranks and the label of a node is only allowed to contain nodes of higher rank. One of the main advantages of HHL is that it suffices to fix a ranking function and then the smallest possible HHL that respects this ranking can be computed in polynomial time. This labeling is called a canonical HHL [5].

**Definition 2 (Canonical HHL)** *Given a node ordering $r$, the node $w \in V$ is contained in the label set $L(v)$ of $v \in V$, if and only if $w$ has the highest ranking on some shortest path originating from node $v$.*

It is easy to prove that the canonical HHL is both necessary and sufficient to fulfill the cover property. However, determining the ranking function that minimizes $L_{\text{avg}}$ is still NP-hard [5]. There exists a graph family such that the optimal $L_{\text{avg}}$ of HHL is at least a factor of $\mathcal{O}(\sqrt{n})$ of the optimal $L_{\text{avg}}$ of HL [5]. A heuristic is to rank the nodes by degree [10]. While this approach works very fast, label sizes might become huge. More sophisticated approaches use other greedy rankings or are related to Contraction Hierarchy (CH) construction [2], which is another rank-based preprocessing method for faster shortest path computation [14]. However, these do not come with provable guarantees. There do exist approximation algorithms, but with quite large approximation factors of $\mathcal{O}(\sqrt{n} \log n)$ [5]. Using a nested dissection approach, a parametrized approximation factor of $\mathcal{O}(t \log n)$ for label sizes was shown [7], where $t$ denotes the treewidth of the network. This parameter is known to be small in road networks [18].

We will provide novel insights into exact and approximate HHL computation. Furthermore, we will also consider Landmark HL (LHL), which generalizes the HL approach. It also comes in a non-hierarchical and a hierachical variant (HLHL). We discuss how the methods we develop for HHL need to be modified to also cater for HLHL. Furthermore, we provide an extensive experimental study that compares existing and new algorithms for HHL and HLHL.

## 1.1 Related Work

HHL was proposed by Abraham et al. [2] and shown to perform well on road networks using heuristics inspired by contraction hierarchies (CH). It was demonstrated that HHL outperforms CH significantly with respect to query time. In [11], $\mathcal{O}(\log n)$ approximate solutions for HL were compared to heuristically obtained HHL solutions, showing that HHL labels are 20-50% larger but can also be computed much faster. The gap in label size is also far from the worst-case gap of $\Theta(\sqrt{n})$ shown for specifically constructed graphs [15].

Several scalable HHL heuristics have been investigated in the meantime, offering different trade-offs between preprocessing time and label size [3, 17]. The best known approximation algorithms for HHL, called w-HHL and g-HHL come with large approximation factors of $\mathcal{O}(\sqrt{n}\log n)$ [5]. It was also shown that the analysis is tight for g-HHL. For w-HHL, there exist example instances on which the produced labeling is larger than the optimum by a factor of $\Theta(\sqrt[3]{n})$.

There are several parametrized upper bounds for maximum (and thus average) label sizes in HL and HHL. For HL, an expected upper bound of $\mathcal{O}(\kappa \log n)$ was proven in [16], where $\kappa$ denotes the skeleton dimension of the network. This approach relies on a randomized preprocessing scheme. For HHL, labels of size $\mathcal{O}(h \log D)$ exist where $h$ is the highway dimension and $D$ the network diameter. Using polynomial-time preprocessing, the bound increases to $\mathcal{O}(h \log D \log n)$ [1]. However, the preprocessing is still too demanding to be used in practice even on small networks. A more practical approach that also comes with bounded label sizes was proposed in [7]. It is called nested dissection and was originally designed for CH computation. The approach relies on recursively dividing the graph using balanced node separators. The method produces HHL with label sizes in $\mathcal{O}(t \log n)$ where $t$ denotes the treewidth. Allowing only polynomial construction time, the label sizes are in $\mathcal{O}(t \log^2 n)$. On trees, where $t = 1$, the same strategy yields a 2-approximation for $L_{\text{avg}}$ [22]. There is also a PTAS on trees [4]. However, the complexity of minimizing label sizes of HHL on trees is still an open problem.

Landmark Hub Labeling (LHL) was introduced in [25]. It relies on a weaker definition of the cover property and thus allows for smaller label sizes than HL. A $\mathcal{O}(\log n)$-approximation algorithm with a running time of $\mathcal{O}(n^6)$ was proposed. For the hierarchial variant, however, so far no algorithm with an approximation guarantee is known. Parameterized upper bounds transfer from HHL to HLHL as every HHL is also a valid HLHL. The gaps between the different variants of LHL and HL have been shown theoretically by the authors in [8]. Of all the different distance labels, the optimal LHL has the smallest label size. Experimental evaluations on road networks and grid graphs show, though, that HLHL produced by sensible heuristics outperfoms HHL with respect to label size [25, 26].

## 1.2 Contribution

We investigate HHL algorithms with provable guarantees on general graphs and on trees. We are able to extend most results concerning HHL to HLHL as well. The following are our main contributions:

- We present an exact algorithm for HHL with a running time of $\mathcal{O}(n^2 2^n)$ that avoids iterating over all $n!$ possible ranking functions. While the algorithm does not scale to large networks, it can be used to assess the quality of heuristics on small instances. We also adapt this exact algorithm to compute an optimal labeling for HLHL, with running time of $\mathcal{O}(n^3 2^n)$ and space of $\mathcal{O}(n 2^n)$.

- We show that the approximation proof of w-HHL derived from the greedy SetCover algorithm as provided in [5] needs additional arguments for its correctness. We fill in the respective steps and prove that the approximation factor of $\mathcal{O}(\sqrt{n}\log n)$ for w-HHL is maintained.
- We prove that two known approximation algorithms for HHL on general graphs, namely w-HHL and g-HHL, produce constant-factor approximations on trees. By generalizing an LP-based approach from [4], we prove a more general result that nested dissection based on $\alpha$-balanced separators for $\alpha \in [1/2, 1)$ yields an approximation factor of $\frac{2\alpha}{1-\alpha}$. For w-HHL and g-HHL we show that $\alpha = 1/\sqrt{2}$.
- We present a 2-approximation of HLHL on complete binary trees.
- We implement and evaluate the performance of the approximation algorithms of HHL (g-HHL and w-HHL) on a diverse set of benchmarks. We assess their quality with respect to lower bounds and compare the outcomes to those of the commonly used H(L)HL heuristics on general graphs and trees. Despite their large theoretical approximation factors, it turns out that the approximation algorithms outperform the heuristics drastically.

## 2 Preliminaries

Throughout the paper, we assume to be given an undirected connected graph $G(V, E)$ with non-negative edge weights $c : E \to \mathbb{R}^+$ and unique shortest paths between all pairs of nodes. The latter property is a common assumption that might also be enforced via symbolic perturbation of the edge weights. We refer to the shortest path between two nodes $s, t \in V$ as $\pi(s, t) = s, \ldots, t$ and to its cost as $c(\pi(s, t))$ or $c_s(t) = c_t(s)$ for short.

The formal definition of HL and HHL was already provided in the introduction. It remains to define Landmark Hub Labeling (LHL). A label $v$ is called a hub for $s$ and $t$ if and only if $v$ lies on the shortest path between $s$ and $t$. Differing from a hub label, we introduce the concept of a *perfect landmark*.

**Definition 3 (Perfect Landmark)** *A node $l \in V$ is called a perfect landmark for a node pair $s, t \in V$ if $|c_s(l) - c_t(l)| = c_s(t)$.*

LHL is a generalization of HL that allows landmarks to fulfill the cover property.

**Definition 4 (Landmark Hub Labeling)** *A 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 perfect landmark for $s, t$.*

This weaker cover property allows for smaller labelings. However, it also results in a more intricate query answering algorithm. While for (H)HL, we know the upper bound

$$\text{UB} := \min_{v \in L(s) \cap L(t)} (c_s(v) + c_t(v))$$

to be tight, for LHL also the lower bound

$$\text{LB} := \max_{v \in L(s) \cap L(t)} |c_s(v) - c_t(v))|$$

could be the one equal to the correct shortest path distance. Clearly, if UB = LB, both are tight. But if LB < UB, one needs to check which of them is equal to $c_s(t)$. To determine whether the lower

bound implied by some landmark $l$ is tight, one extracts the shortest path from $s$ to $l$ (assuming w.l.o.g. $c_s(l) > c_t(l)$) up to distance LB and checks if $t$ is found at this distance. If that is not the case, UB is tight.

To perform the check more efficiently, it is beneficial for the LHL to fulfill an additional property, namely to be path-consistent (PC) [25].

**Definition 5 (Path-Consistent Labeling)** *In a* path-consistent *labeling, for each $w \in L(v)$, we also have $w \in L(u)$ for all $u$ on the shortest path from $v$ to $w$.*

As shown in [25], every canonical HHL is PC. A hierarchical LHL (HLHL) can be defined in the same manner as HHL by using a ranking function and only allowing node labels $L(v)$ to contain nodes of equal or higher importance than that of $v$. We use PC-LHL to refer to a path-consistent LHL and HPC-LHL denoted a hierarchical path-consistent LHL.

**Definition 6 (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 nodes of highest rank on each maximal shortest path that contains $v$.*

A maximal shortest path, denote as $\pi^+$, is one that cannot be extended by any edge without losing the shortest path property. As shown in [25], given a rank $r$, the canonical HPC-LHL has the smallest label size of all HPC-LHLs respecting $r$.

# 3 Exact Algorithms for HHL and HPC-LHL

We first design algorithms that produce exact hierarchical labelings, that is, labelings with smallest average label size.

## 3.1 HHL

There are $n!$ many node permutations, each of them defining a valid HHL. A naive algorithm for computing the optimal HHL with respect to maximum or average label size is to iterate over all these permutations, compute the canonical HHL for each one, and keep track of the one with smallest average label size. The respective running time amounts to $\mathcal{O}(n^3 n!)$. As the label $L(v)$ of a node $v$ depends on the exact ranking of all the nodes of rank higher than $v$, it appears to be difficult to improve that running time. However, to get the average label size in a canonical HHL, we can also sum over the inverse label sizes $L^{-1}(v)$ instead, where $L^{-1}(v) := \{w \in V \mid v \in L(w)\}$. Clearly, it yields $\sum_{v \in V} |L(v)| = \sum_{v \in V} |L^{-1}(v)|$.

We show next that considering the inverse label size has a crucial advantage in the design of an exact algorithm, as $L^{-1}(v)$ only depends on the set of nodes with a rank higher than $v$ but *not* on their particular order. Figure 1 illustrates an example of the inverse label set of a node $v$.

**Lemma 1** *For a node $v$, given $R(v) := \{w \in V \mid r(w) > r(v)\}$, the inverse label $L^{-1}(v)$ can be determined in $\mathcal{O}(n^2)$.*

**Proof:** The inverse label of node $v$ is the set of nodes $w$ for which $v$ is the node of maximum rank on the shortest path from $w$ to $v$. Thus, $L^{-1}(v)$ can be determined by running Dijkstra from $v$ to compute the shortest path tree, and then cutting off all subtrees rooted at a node $u \in R(v)$. □

Lemma 1 can be leveraged to develop a DP that returns the HHL with smallest possible average label size as follows: We assign a table with $n$ rows and $2^n$ columns, where row $v$ corresponds to

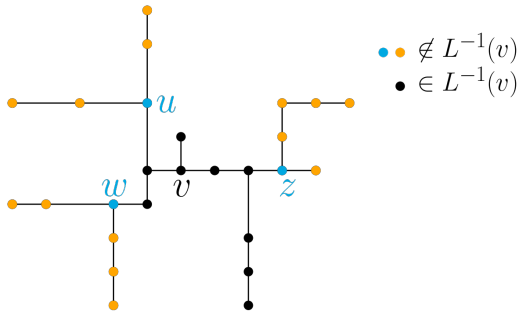



Figure 1: An example of the shortest path of node $v$ in the graph $G$. Let the nodes $u, w, z$ have the highest three ranks and the node $v$ has the fourth highest rank. For each blue and orange node, the highest-ranking node on the shortest path from them to node $v$ must be one of the nodes $u, w, z$. Then only the black nodes in the shortest path tree contain the node $v$ in their label set.

node $v \in V$ and each column corresponds to a set of the powerset $\mathcal{P}(V)$. The columns are sorted in ascending order of set size. For $v \in V$ and $S \in \mathcal{P}(V)$ we can define $m[v, S]$ recursively as follows:

1. $m[v, \emptyset] = 0$
2. $m[v, S] = \begin{cases} \min\{m[u, S \setminus \{v\}] \mid u \in V\} + |L^{-1}(v)|, \text{ with } R(v) = S \setminus \{v\} & \text{for } v \in S \\ \infty, & \text{otherwise} \end{cases}$

The entries in the first column, which corresponds to the empty set, are all filled with zeros. The other columns are filled in order using the following rules: Let the column set be $S$ and the row node $v$. If $v \notin S$, we fill the respective cell with $\infty$. Otherwise, we want to enter the smallest summed inverse label size of the nodes in $S$ assuming that the contained nodes have the $|S|$ highest ranks among all nodes and $v$ has the lowest rank among the nodes in $S$. To compute this value, we consider the smallest entry in column $S \setminus \{v\}$ and add to it the size of $L^{-1}(v)$ for $R(v) = S \setminus \{v\}$. Once the whole table is filled, the smallest entry in the last column (which corresponds to the whole node set) divided by $n$ indicates the optimal average label size. The respective ranking can be deduced via backtracking.

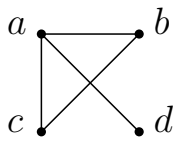



Figure 2: An example graph with four nodes and unit edge weight.

**Example.** Now we explain the DP with an example shown in Figure 2. To compute the optimal HHL of the four-node example graph, we create a table shown in Table 1 with four rows and sixteen columns. The columns are subsets of $\{a, b, c, d\}$ sorted by their size. Each row represents one of the nodes in the graph. The table is filled from left to right. The first column is filled with zeroes. For the rest of the cells, each cell $(v, S)$ with $v \notin S$ is filled with $\infty$, e.g. the cell $(a, \{b, c, d\})$. If $v \in S$, then the value can be calculated from the minimum of the column $S \setminus \{v\}$ plus the inverse label size of $v$, taking into account that the nodes in $S \setminus \{v\}$ have a higher rank than $v$. For example, to calculate the cell $(c, \{a, c, d\})$, we assume that $c$ has the third highest rank and $a, d$

| | $\emptyset$ | $a$ | $b$ | $c$ | $d$ | $a,b$ | $a,c$ | $a,d$ | $b,c$ | $b,d$ | $c,d$ | $a,b,c$ | $a,b,d$ | $a,c,d$ | $b,c,d$ | a,b, c,d |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a$ | 0 | 4 | $\infty$ | $\infty$ | $\infty$ | 7 | 7 | 7 | $\infty$ | $\infty$ | $\infty$ | 9 | 9 | 9 | $\infty$ | 10 |
| $b$ | 0 | $\infty$ | 4 | $\infty$ | $\infty$ | 6 | $\infty$ | $\infty$ | 7 | 7 | $\infty$ | 7 | 7 | $\infty$ | 9 | 8 |
| $c$ | 0 | $\infty$ | $\infty$ | 4 | $\infty$ | $\infty$ | 6 | $\infty$ | 7 | $\infty$ | 7 | 7 | $\infty$ | 7 | 9 | 8 |
| $d$ | 0 | $\infty$ | $\infty$ | $\infty$ | 4 | $\infty$ | $\infty$ | 5 | $\infty$ | 7 | 7 | $\infty$ | 7 | 7 | 9 | 8 |

Table 1: DP table. For example, the value in the cell $(d, \{a, d\})$ is the smallest summed inverse label size of $a$ and $d$ with the rank where $r(a) = 4$ and $r(d) = 3$. This result is followed by the sum of the minimum in the column $\{a\}$, which means that all nodes add $a$ to their label set, and then plus one label, since $d$ adds itself to its label set. Formerly, $L^{-1}(a) = \{a, b, c, d\}$ and $L^{-1}(d) = \{d\}$.

have higher ranks than $c$. It follows that the inverse label set size of $c$ is 2, where $L^{-1}(c) = \{b, c\}$. The minimum of column $\{a, d\}$ equals 5. Hence, the value of cell $(c, \{a, c, d\})$ is set to 7. After the whole table is filled, the minimum HHL size stored in the last column and the optimal rank can be read by backtracking. In Table 1, the blue color marks backtracking to one of the possible optimal ranks $r$, where $r(a) > r(d) > r(c) > r(b)$.

**Theorem 1** *The DP algorithm computes the minimum average label size of an HHL in $\mathcal{O}(n^2 2^n)$ time using $\mathcal{O}(n2^n)$ space.*

**Proof:** According to Lemma 1, computing $L^{-1}(v)$ takes $\mathcal{O}(n^2)$. Using this method to fill each of the $\mathcal{O}(n2^n)$ cells in the table would result in an overall running time of $\mathcal{O}(n^3 2^n)$. However, we reduce this running time by precomputing all shortest path trees in $\mathcal{O}(n^3)$, using $\mathcal{O}(n^2)$ space. Then, whenever we have to compute the inverse label size of a node $v$ with given set $R(v)$ of higher ranked nodes, we simply traverse the precomputed shortest path tree from $v$ and cap it at nodes in $R(v)$ instead of computing the whole tree from scratch. As the tree has a size of $\mathcal{O}(n)$, the traversal takes only linear time. Therefore, the overall running time is reduced to $\mathcal{O}(n^2 2^n + n^3) = \mathcal{O}(n^2 2^n)$. The space consumption is dominated by the number of table cells.

To show correctness, we prove by induction that the following loop invariant upholds: Once a table column with corresponding $S \in \mathcal{P}(V)$ is completely processed, the minimum value of this column equals to the optimal summed inverse label size for $S$, given that those have rank higher than the nodes in $V \setminus S$. Here we show it on a connected graph, for a graph with multiple connected components this proof works analogously for each component.

For columns that refer to a single node, that is, $S = \{v\}$, we compute $L^{-1}(v)$ with an empty set $R(v)$ and thus get $|L^{-1}(v)| = n$. Adding that to the zero obtained from the first column, which encodes $\emptyset$, the value of $n$ does not change. Clearly, this is the smallest possible inverse label size for any choice of a highest rank node. For a column $S$ with size $k + 1 > 1$, we assume that all columns with set sizes up to $k$ are processed correctly. Let $v^*$ be the lowest ranked node in $S$ in the optimal ordering of $S$, we compute $m[v^*, S]$ as $|L^{-1}(v^*)|$ with $R(v^*) = S \setminus \{v^*\}$ adding the minimum of the column $S \setminus \{v^*\}$. According to the induction hypothesis, the minimum value of the column $S \setminus \{v^*\}$ is the optimal inverse label size of the set $S \setminus \{v^*\}$. And for any ordering in $R(v^*)$, $L^{-1}(v^*)$ remains the same. Thus $m[v^*, S]$ is equal to the optimal summed inverse label size for $S$, given that they have a higher rank than the nodes in $V \setminus S$. Hence, the optimal summed inverse label size is the minimum value of the last column.

□

### 3.2 HPC-LHL

We now apply the same idea to compute the optimal HPC-LHL. In canonical HPC-LHL, for a given rank $r$, the inverse label size $|L^{-1}(v)|$ of a node $v$ also depends only on the set of nodes whose rank is higher than the rank of $v$.

**Lemma 2** *For a node $v$, given $R(v) \coloneqq \{w \in V \mid r(w) > r(v)\}$ and all maximal shortest paths in $G$, the inverse HPC-LHL label $L^{-1}(v)$ can be determined in $\mathcal{O}(n^3)$.*

**Proof:** Once the maximal shortest paths have been precomputed, an index of the paths can be created. And for each node, we can store a list of the indices of the maximal shortest paths that node lies on. For each node $w \in R(v)$, the maximal shortest paths that contain $w$ should be marked as covered, as $v$ cannot have the highest rank on those paths. At most $n^2$ paths per node in $R(v)$ can be marked as covered. This steps has a time complexity of $\mathcal{O}(n^3)$. For the remaining unmarked maximal shortest paths that contain $v$, it is necessary that all nodes contain $v$ in their label set, as $v$ has the highest rank on these paths. Consequently, the number of these nodes is equal to the inverse label size, $|L^{-1}(v)|$. At most $n$ nodes lie on one maximal shortest paths. Hence, in total, the time complexity is $\mathcal{O}(n^3)$.

□

Applying the same DP 1, while calculating the inverse label set according to Lemma 2 we can compute the optimal HPC-LHL.

**Theorem 2** *The DP algorithm 1 computes the minimum average label size of an HPC-LHL in $\mathcal{O}(n^4 2^n)$ time using $\mathcal{O}(n2^n)$ space.*

**Proof:** We precompute all maximal shortest paths in $\mathcal{O}(n^3)$, using $\mathcal{O}(n^3)$ space (see also [25]). According to Lemma 2, the computation of $L^{-1}(v)$ takes $\mathcal{O}(n^3)$. Therefore, the total running time is $\mathcal{O}(n^4 2^n + n^3) = \mathcal{O}(n^4 2^n)$. The space consumption is dominated by the number of table cells.

Since the inverse label set size of a node $v$ depends only on $R(v)$, the correctness can be proven as in Theorem 1. □

## 4 Analysis of w-HHL

The HHL approximation algorithms proposed in [5] all rely on formulating HHL as a SetCover problem and using a greedy algorithm to produce the respective cover. Correctness and approximation quality follow from a folklore Lemma about the greedy SetCover algorithm, which states that iteratively selecting the set whose coverage to cost ratio is at least $1/f(n)$ fraction of the maximum coverage to cost ratio results in a cover of cost within an $\mathcal{O}(f(n)\log n)$ factor of the optimal cost OPT. According to this Lemma, the authors in [5] concluded that w-HHL is a $\mathcal{O}(\sqrt{n}\log n)$-approximation. However, the Lemma applies only if the coverage to cost ratio of the selected elements is monotonously decreasing during the execution of the algorithm. This property is trivially fulfilled in case the sets have static costs. However, if the costs are prone to change, the property needs to be investigated.

We will show now that the property is indeed *not* met for w-HHL by providing an example in which the maximum coverage to cost ratio increases during the execution of the greedy algorithm. Recall that the w-HHL algorithm greedily selects the next node to assign the currently largest not yet assigned rank based on $\max_v w(v)$ where $w(v) := c_v/|L_v^{-1}|$ with $c_v$ denoting the number of shortest paths that would be newly covered by $v$.

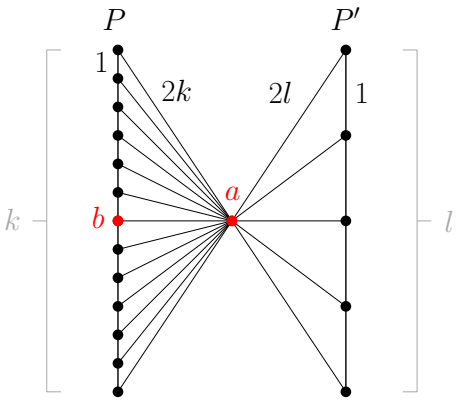



Figure 3: Schematic example graph for w-HHL analysis.

**Lemma 3** *Let $v_1, \ldots, v_n$ denote the order in which w-HHL selects the nodes. The sequence of corresponding weights $w(v_1), , \ldots, w(v_n)$ is not necessarily monotonously decreasing.*

**Proof:** Consider a graph constructed as follows: There are two paths $P$ and $P'$ with $k$ and $l$ nodes, respectively, and $k > l$. Edge costs along the two paths are all set to 1. Further, there is an additional node $a$ that is connected to all nodes in $P$ and $P'$, with edge costs of $2k$ to nodes in $P$ and edge costs of $2l$ to nodes in $P'$. We assume $k$ is odd and denote with $b$ the middle node of path $P$. See Figure 3 for an illustration.

Let us now consider weights $w(v)$ for all $v \in V$. We have $|L^{-1}(v)| = k + l + 1$ for all nodes and hence weight differences are induced solely by the number of newly covered shortest paths. For $a$, that number equals $c_a = kl + k + l + 1$, as all shortest paths from a node in $P$ to a node in $P'$ or to $a$ traverse $a$ and additionally $a$ covers the path which only contains itself. For $b$, we have $c_b = \left(\frac{k-1}{2}\right)^2 + k + l + 1$, which is the maximum attainable for any node on path $P$. For nodes $v$ on path $P'$ we have $c_v \leq \left(\frac{l}{2}\right)^2 + k + l + 1$. This term is smaller than $c_b$ due to $k > l$. If $l > \frac{k-1}{4}$, the maximum weight is assumed by node $a$. Accordingly, greedy w-HHL selects this node first.

Now, we consider the impact of $a$ being chosen before $b$ on the weight of $b$. The number of shortest paths that are now covered by $b$ is reduced to $c'_b = \left(\frac{k-1}{2}\right)^2 + k$ and the inverse search space size to $|L^{-1}(b')| = k$. The new weight is hence equal to $w'_b = \frac{k^2+2k+1}{4k}$. For $l < \frac{k}{3} - 2$, we get $w'_b > w_a$. For sufficiently large $k$, we can always choose an $l$ strictly between $\frac{k-1}{4}$ and $\frac{k}{3} - 2$. Accordingly, the maximal value among the so far not selected nodes increases over the course of the algorithm. □

Why could this pose a problem for the approximation quality of the w-HHL algorithm? A crucial ingredient in proving the approximation factor of $\mathcal{O}(f(n) \log n)$ is that after each round of greedy selection, the current solution can be extended to a full cover at cost at most the optimal cover cost OPT of the original instance. But if the weights of the sets may increase, it could happen that at some point during the execution of the greedy algorithm the currently available collection of sets with their corresponding weights does not allow for a completion to a full cover at cost at most OPT. This would invalidate the analysis. However, we will show next that for HHL in general, any partial solution can always be extended to a full solution using at most OPT additional labels.

**Lemma 4** *Let p-HHL be a partial HHL in which the p highest node ranks are already assigned. For any $1 \leq p \leq n-1$, each p-HHL can be completed to a full HHL at cost at most OPT for the respective 0-HHL.*

**Proof:** We observe that for any node $v \in V$, its inverse search space size $L^{-1}(v)$ in a given HHL solely depends on the nodes of higher rank than $v$. Let now $v_1, v_2, \ldots, v_n$ denote the optimal node ordering for a min-sized HHL and consider a $p$-HHL for some $p \in \{1, \ldots, n\}$. Let $v$ be the node with smallest index $i$ in the optimal ordering that was not already assigned a rank in the $p$-HHL. If we now assign the next highest rank to $v$ to get a $p+1$-HHL, it yields that the inverse search space size of $v$ in this labeling is at most as large as in the optimal HHL. Namely, all nodes of higher rank than $v$ in the optimal ordering also have a higher rank in the $p$-HHL. The argument can then be repeated until the partial HHL is completed to a full HHL. The total cost may not exceed OPT as we sum up only over a subset of the nodes and for those we have smaller or equal inverse search space sizes as in the optimal full solution. □

Thus, in any round of the greedy algorithm, there always exists a node $v$ with weight $w(v) \geq \frac{C}{\sqrt{n} \cdot \mathrm{OPT}}$ where $C$ denotes the total number of shortest paths that still need to be covered. This now allows to apply the standard greedy analysis, which shows that the approximation factor of w-HHL is indeed in $\mathcal{O}(\sqrt{n} \log n)$.

# 5 Approximation on Trees

We now turn our focus to tree graphs $T(V, E)$. No exact polytime algorithm is known for computing the HHL with minimum average label size on trees. However, it was shown in [4], that there is a (rather complicated) PTAS, and a simple 2-approximation algorithm that relies on a balanced separator decomposition. In this section, we show that g-HHL and w-HHL admit constant-factor approximations on trees and provide further structural insights into close-to-optimal HHL and HLHL solutions on tree graphs.

## 5.1 Upper Bound for $\alpha$-Balanced Separator Algorithm

The algorithm proposed by Peleg [22] uses nested dissection on trees as decribed in the last section. It relies on using a balanced separator on each level of the decomposition tree. We will now generalize this approach to using $\alpha$-balanced separators for $\alpha \in (0, 1)$. Such a separator $S$ demands that all components in $T[V \setminus S]$ have size at most $\alpha n$.

It was proven in [4] that the optimal HL on trees is hierarchical. They provide a primal and dual formulation as shown in Table 2 of HL as linear programs to show that applying Peleg's algorithm based on balanced separators with $\alpha = 1/2$ yields a solution that is a 2-approximation compared to the optimal HL on trees. First, we recall their approach and then extend their proof to cater for general $\alpha$-balanced separators.

In the formulation of the primal linear program, $I$ is the set of all (unordered) pairs of vertices. Note that for all vertices $u$, the pair $\{u, u\}$ is included as well. The variables $x_{uv}$ indicate whether $v$ is contained in the label set $H_u$ of $u$. For every node $w$ on the shortest path $P_{uv}$ from $u$ to $v$, the variable $y_{uvw}$ is lower or equal to both $x_{uw}$ and $x_{vw}$. For all pairs $\{u, v\} \in I$, the sum over all $y_{uvw}$ for all nodes $w$ on $P_{uv}$ must be at least one. This implies that any shortest path must be covered by at least one node. The goal of the primal linear program is to determine the assignments for the variables $x$ and $y$, minimizing the sum over the $x$ variables. This is exactly equivalent to minimizing the hub label size.

(**PRIMAL-LP**)

$$\min : \sum_{u\in V}\sum_{v\in V} x_{uv}$$

$$\begin{aligned}
\text{s.t.} : \sum_{w\in P_{uv}} y_{uvw} &\geq 1, && \forall\{u,v\}\in I\\
x_{uw} &\geq y_{uvw}, && \forall\{u,v\}\in I, \forall w\in P_{uv}\\
x_{vw} &\geq y_{vuw}, && \forall\{u,v\}\in I, \forall w\in P_{uv}\\
x_{uv} &\geq 0, && \forall\{u,v\}\in V\times V\\
y_{uvw} &\geq 0, && \forall\{u,v\}\in I, \forall w\in P_{uv}
\end{aligned}$$

(**DUAL-LP**)
variables $\alpha_{uv}$ and $b_{uvw}$ for $w \in P_{uv}$

$$\max : \sum_{\{u,v\}\in I} a_{uv}$$

$$\begin{aligned}
\text{s.t.} : a_{uv} &\leq b_{uvw} + b_{vuw}, && \forall\{u,v\}\in I, u\neq v\\
& && \forall w\in P_{uv}\\
a_{uu} &\leq b_{uuu}, && \forall u\in V\\
\sum_{v:w\in P_{uv}} b_{uvw} &\leq 1, && \forall(u,w)\in V\times V\\
a_{uv} &\geq 0, && \forall\{u,v\}\in I\\
b_{uvw} &\geq 0, && \forall\{u,v\}\in I, \forall w\in P_{uv}\\
b_{vuw} &\geq 0, && \forall\{u,v\}\in I, \forall w\in P_{uv}
\end{aligned}$$

Table 2: Primal and Dual LPs for HL on trees [4].

In the dual linear program, $a_{uv}$ is a variable of the unordered pair $\{u,v\}\in I$, however variables $b_{uvw}$ and $b_{vuw}$ are different variables, where $(u,v)\in V\times V$ is an ordered pair.

The authors in [4] derive the constant approximation factor of Peleg's algorithm by comparing the contribution from each iteration of the heuristic algorithm to the value of DUAL-LP. We adjust the variables assignment in the DUAL-LP associated with parameter $\alpha$ such that the constraints are still fulfilled, then apply the analogous upper bound analysis.

**Theorem 3** *The $\alpha$-balanced separator algorithm is a $\frac{2\alpha}{1-\alpha}$-approximation algorithm on trees for $\alpha\in[1/2,1)$.*

**Proof:** We apply the $\alpha$-balanced separator algorithm and construct a fractional solution for the dual linear program. Let $T'$ be the current tree of size $n'$ in one iteration of the algorithm, and let $r$ be a $\alpha$-balanced separator. Since trees are free of cycles, one vertex is sufficient to cut a tree into subtrees of sizes at most $\alpha n'$. Furthermore, such a node $r$ exists for every tree. Let $A = 1/\alpha n'$ and $B = 1/2\alpha n'$. Let $T_1,\ldots,T_a$ be the connected components of $T'$ after removal of node $r$.

We assign values for the variables $a_{u,v}$, $b_{uvw}$ and $b_{vuw}$ for node pairs $\{u,v\}$, where $r$ is contained on the (unique) path from $u$ to $v$. In the previous iterations, nodes $u$ and $v$ were in the same subtree. Hence, no previous $\alpha$-balanced separator is contained in the shortest $u$-$v$ path. After removing $r$, they lie in different connected components. Hence, the variables $a_{uv}$, $b_{uvw}$ and $b_{vuw}$ are assigned exactly once during execution of the algorithm.

The rules of assignment are as follows:

We verify that these assignments are feasible solutions for the dual linear program. As shown in Table 3 and Figure 4, for all nodes pairs $(u,v)\in V\times V\setminus\{(r,r)\}$ it follows $a_{uv} = A = b_{uvw}+b_{vuw} = 2B$ and $a_{rr} = b_{rrr}$, so both the first and the second constraints from DUAL-LP are satisfied.

For the third constraint, consider nodes $u\in T_i$, $w\in T_j$ and $v\notin T'$, where $i\neq j$. If $w$ is not on the shortest $u$-$v$ path, then $b_{uvw}$ will not be assigned. If $w$ lies on the shortest $u$-$v$ path, then there must be a balanced separator $r'$ in some previous iteration of the algorithm such that $w\in P_{ur'}$. In this case, the variable $b_{uvw}$ receives the value zero. Therefore, for nodes $u\in T_i$, $w\in T_j$, $i\neq j$,

| | |
|---|---|
| $a_{uv} = A$, | $\forall u \in T_i, v \in T_j \wedge i \neq j$ |
| $a_{ur} = A$, | $\forall u \in T' \setminus \{r\}$ |
| $a_{rr} = b_{rrr} = B$ | |
| $b_{uvw} = A$ and $b_{vuw} = 0$, | $\forall w \in P_{rv} \setminus \{r\} : \forall u \in T_i, v \in T_j \wedge i \neq j$ |
| $b_{uvw} = 0$ and $b_{vuw} = A$, | $\forall w \in P_{ur} \setminus \{r\} : \forall u \in T_i, v \in T_j \wedge i \neq j$ |
| $b_{uvr} = b_{vur} = B$, | $\forall u \in T_i, v \in T_j \wedge i \neq j$ |
| $b_{urr} = b_{rur} = B$, | $\forall u \in T \setminus \{r\}$ |

Table 3: The assignments of variables in the dual linear program.

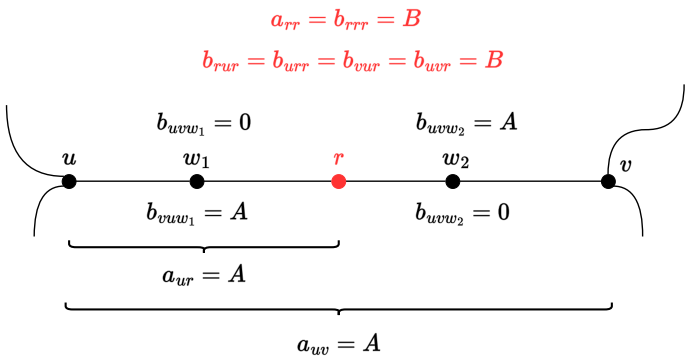



Figure 4: An illustration of the assignments for DUAL-LP variables.

all nodes $v$ whose $b_{uvw}$ are non-zero lie in $T_j$. For $u \in T_i \cup \{r\}$ and $w \in T_j$, where $i \neq j$, it follows

$$\sum_{v|w \in P_{uv}} b_{uvw} \leq |T_j| \cdot A \leq \alpha n' \cdot \frac{1}{\alpha n'} = 1. \tag{1}$$

For $u \in T_i \cup \{r\}$ and $w = r$, it follows that,

$$\sum_{v|w \in P_{uv}} b_{uvw} = \sum_{v|r \in P_{uv}} b_{uvr} = \sum_{v|r \in P_{uv}} B \leq n' \cdot \frac{1}{2\alpha n'} \leq 1. \tag{2}$$

Hence, the assignments for $a$ and $b$ lead to a feasible solution of DUAL-LP. Furthermore, we show that the assignments lead to a resulting value that is at least $\frac{1-\alpha}{2\alpha}$ of the maximal value. This proves that the algorithm is a $\frac{2\alpha}{1-\alpha}$-approximation algorithm.

During one iteration, we add $r$ into the hub labels of all nodes in $T'$. This increases the cost of hub labeling by $n'$. Now consider the contribution $C$ of the assignment in this iteration, which is the sum of all $a_{uv}$ such that $r$ lies on the shortest $u$-$v$ path, i.e., $u$ and $v$ are in different subtrees after removal of $r$. Let $t_1, \ldots, t_s$ be the subtree sizes. It follows that

$$C = \sum_{i=1}^{s-1} \sum_{j=i+1}^{s} \left( \sum_{u \in T_i, v \in T_j} a_{uv} \right) + \sum_{i=1}^{s} \sum_{u \in T_i} a_{ur} + a_{rr} \tag{3}$$

$$= A \sum_{i=1}^{s-1} \sum_{j=i+1}^{s} t_i t_j + A(n' - 1) + B \tag{4}$$

$$= \frac{A}{2} \sum_{i=1}^{s} t_i \left( \sum_{j=1, i \neq j}^{s} t_j \right) + A(n' - 1) + B. \tag{5}$$

Considering that the term $\sum_{j=1,i\neq j}^{s} t_j = n' - 1 - t_i \geq n' - 1 - \alpha n'$, it follows that

$$\sum_{i=1}^{s} t_i \left( \sum_{j=1, i \neq j}^{s} t_j \right) \geq \sum_{i=1}^{s} t_i (n' - 1 - \alpha n') = (n' - 1)(n' - 1 - \alpha n'). \tag{6}$$

Moreover,

$$C \geq \frac{(n' - 1)(n' - 1 - \alpha n')}{2\alpha n'} + \frac{n' - 1}{\alpha n'} + \frac{1}{2\alpha n'} = \frac{(n')^2(1 - \alpha)}{2\alpha n'} = \frac{1 - \alpha}{2\alpha} n'. \tag{7}$$

Hence, the $\alpha$-balanced separator algorithm is a $\frac{2\alpha}{1-\alpha}$-approximation algorithm on trees. □

## 5.2 Properties of g-HHL and w-HHL on Trees

Both g-HHL and w-HHL are greedy algorithms for HHL which assign the node ranks from highest to lowest. g-HHL always selects next the node that is contained in the largest number of shortest paths that do not already contain a node of higher rank. w-HHL also uses that shortest path count but divides it by the inverse label size.

To show that g-HHL and w-HHL are $\frac{2}{\sqrt{2}-1}$-approximations on trees, we introduce the algorithms using the center graph. The center graph is defined on an undirected graph as follows (analogous to [5]):

**Definition 7 (Center Graph)** *A center graph $G_v = (V, E_v)$ of node $v$ in graph $G = (V, E)$ contains all the vertices in $G$, and there is an edge $\{u, w\} \in E_v$ if and only if there is a shortest path between $u$ and $w$ via $v$.*

According to the definition, g-HHL selects the node whose center graph has the most edges in each iteration and assigns it the next highest rank; instead,w-HHL selects the node whose central graph has the highest density (number of edges divided by the number of non-isolated nodes), which corresponds to the highest weight, as described in Section 4.

It is trivial to show that g-HHL and w-HHL return the same label sizes on trees. Namely, for any node $v \in T$, there is exactly one shortest path from $v$ to any other node in $T$. Therefore, the center graph of each node in $T$ does not contain isolated vertices. Let $v \in T$ be the node whose center graph has the highest number of edges. It follows immediately that the center graph of $v$ has also the highest density among all center graphs.

For the same tree, both g-HHL and w-HHL assign $v$ the next highest rank, every node in $T$ contains $v$ in its label set. After the removal of node $v$, $T$ decomposes into subtrees of smaller size. The shortest path pairs covered by $v$ are separated into different subtrees. Hence, recursively g-HHL and w-HHL select the same node in each subtree. Hence, the resulting label sizes of both algorithms are the same.

To prove that g-HHL and w-HHL have a constant approximation factor on trees, we show that a node $v \in V$ maximizing the number of edges in its center graph is actually an $\alpha$-balanced separator of the tree $T$ for some $\alpha \in [1/2, 1)$. Then, the assertion readily follows from Theorem 3.

**Lemma 5** *Let $T = (V, E)$ be a tree with $n$ nodes. The vertex $v \in V$ whose center graph $G_v$ of $T$ maximizes the number of edges is a $\frac{1}{\sqrt{2}}$-balanced separator.*

**Proof:** Let $v$ be the node whose center graph $G_v$ maximizes the number of edges. Let $T_m$ be the largest subtree rooted at a neighbor of $v$. Our goal is to find an upper bound on the size $t$ of $T_m$. In the worst-case scenario, $T_m$ has as many nodes as possible, and the center graph $G_v$ still has the most edges. This is achieved if the number of edges contributed from $T \backslash T_m$ is as large as possible, i.e., if the nodes in $T \backslash T_m$ form a clique in $G_v$. Therefore, $T \backslash T_m$ forms a star in $T$ whose center is $v$. In this case, these nodes contribute $\frac{(n-t)(n-t-1)}{2}$ edges in $G_v$. If $T_m$ is much larger than $T \backslash T_m$, then there exists some node $u$ on $T_m$ whose center graph $G_u$ has more edges than $G_v$. In order to make the center graph $G_u$ have as few edges as possible (so that $T_m$ can be as large as possible), $T_m$ must be a path. Hence, $u$ is the balanced separator of $T$ whose center graph has the most edges among the nodes on $T_m$. Since $G_v$ has at least so many edges as in $G_u$, it follows that $\frac{(n-t-1)(n-t-2)}{2} + t(n-t-1) + n - 1 \geq \left(\frac{n-1}{2}\right)^2 + n - 1$. Accordingly, the balance ratio of separator $v$ is $\frac{t}{n} \leq \frac{1}{\sqrt{2}}$. □

Based on Section 5.2 and Section 5.1, we obtain the following theorem.

**Theorem 4** *g-HHL is a $\frac{2}{\sqrt{2}-1}$-approximation algorithm for HL on trees.*

Furthermore, the center graph of any node on the tree is a complete $k$-partite graph, where $k$ is the number of neighbors from node $v$ plus one. It was shown in [15] that both in a regular graph and a regular bipartite graph (i.e., degrees in each part are uniform), the density of the graph is larger or equal to the density of any subgraph. We now show that this also holds for the complete $k$-partite graph.

**Lemma 6** *The density of a complete $k$-partite graph $G$ is larger or equal to the density of any subgraph of $G$.*

**Proof:** We prove it by induction.

**Base case:** According to [15], the statement holds for any complete bipartite graph.

**Inductive step:** Let $G = (\bigcup_{i=1}^{k+1} V_i, E)$ be a complete $k+1$-partite graph, and $G_S = (S, E_S)$ be any subgraph of $G$. Denote $n' = |\bigcup_{i=1}^{k} V_i|$, $n_{k+1} = |V_{k+1}|$, $s' = |\bigcup_{i=1}^{k} V_i \cap S|$, $s_{k+1} = |S \cap V_{k+1}|$. Furthermore, define $m'$ and $m'_S$ as the number of edges of the induced subgraph of $G$ and $G_S$ on $\bigcup_{i=1}^{k} V_i$ and $\bigcup_{i=1}^{k} V_i \cap S$. Since $G$ is complete $k+1$-partite, the number of edges in $G$ is equal to $m' + n' n_{k+1}$, and the subgraph $G_S$ contains at most $m'_S + s' s_{k+1}$ edges. Therefore the density

difference between the graph and its subgraph is at least

$$\begin{aligned}
&\frac{m' + n'n_{k+1}}{n' + n_{k+1}} - \frac{m'_S + s's_{k+1}}{s' + s_{k+1}} \\
=&\frac{(m' + n'n_{k+1})(s' + s_{k+1}) - (m'_S + s's_{k+1})(n' + n_{k+1})}{(n' + n_{k+1})(s' + s_{k+1})} \\
=&\frac{(m's' - m'_S n') + n_{k+1}(n's' - m'_S) - s_{k+1}(n's' - m') + n_{k+1}s_{k+1}(n' - s')}{(n' + n_{k+1})(s' + s_{k+1})}.
\end{aligned}$$

It remains to show that this expression is not negative. Followed by inductive hypothesis, the first term $m's' - m'_S n'$ is larger or equal to zero. It is also trivial that $n_{k+1}s_{k+1}(n' - s') \geq 0$ holds.

Furthermore, we know $m'_S \leq 1/2(s' - 1)s'$, it follows that $n's' - m'_S > 0$. We can consider the expresion $n_{k+1}(n's' - m'_S) - s_{k+1}(n's' - m')$ as a linear function of $s_{k+1}$, since the other variables are independent from $s_{k+1}$. If $s_{k+1} = 0$, then the function is positive. If $s_{k+1} = n_{k+1}$, it is not negative as well. Therefore, for all $s_{k+1} \in [0, n_{k+1}]$, the function is larger or equal to zero. Hence, the density of $G$ is larger or equal to the density of any subgraph of $G$. □

## 5.3 LHL Approximation on Binary Trees

In this section we show a 2-approximation on binary trees. First, we derive a lower bound and then show that the resulting label size of the approximation is a factor of 2 of the lower bound. In what follows, the label size of a labeling $L$ is defined as $\sum_{v \in V} |L(v)|$. We denote by $T_i$ the complete binary tree of height $i$ for $i \geq 0$.

### 5.3.1 Lower Bound

We provide a lower bound for the optimal size of a LHL on $T_i$, denote by $\mathrm{OPT}_i$. For that purpose, we will consider labelings on $T_i$ that cover only specific pairs of nodes, that is pairs of nodes $(u, v)$ such that $u$ is not an ancestor of $v$ and $v$ is not an ancestor of $u$ (denote by $(u, v) \in N$). We denote by $\mathrm{OPT}_i^N$ the optimal size of such a labeling. We trivially have that $\mathrm{OPT}_i \geq \mathrm{OPT}_i^N$ as every LHL on $T_i$ always covers all the pairs in $N$.

An important remark is that to cover $(u, v) \in N$ we can only use a label in the subtree rooted at the lowest common ancestor of $u, v$. Denote by $T_{l,i}$ (respectively $T_{r,i}$) the left (respectively right) subtree of the root and $\mathrm{OPT}_i^N[T_{l,i}]$ the size of the labeling $L$ associated to $\mathrm{OPT}_i^N$ restricted to $T_{l,i}$, i.e. we only count $u \in L(v)$ for $u, v \in T_{l,i}$. We have that $\mathrm{OPT}_i^N[T_{r,i}] \geq \mathrm{OPT}_{i-1}^N$ and $\mathrm{OPT}_i^N[T_{l,i}] \geq \mathrm{OPT}_{i-1}^N$ due to the previous remark. Moreover, to cover the pairs between $T_{l,i}$ and $T_{r,i}$, at least $|T_{l,i}| = |T_{r,i}| = 2^i - 1$ nodes have to contain a node in their label that is not in the same subtree. Indeed, if there are less than $|T_{l,i}|$ such nodes, we can find a node $v_l \in T_{l,i}$ with its label containing only nodes in $T_{l,i}$ and a node $v_r \in T_{r,i}$ with its label containing only nodes in $T_{r,i}$ and the pair $(v_l, v_r)$ is not covered. Finally, the root has to contain itself in its label.

Thus, $\mathrm{OPT}_i^N \geq 2 \cdot \mathrm{OPT}_{i-1}^N + 2^i$, and as $\mathrm{OPT}_0^N \geq 1$, this gives us the following lower bound for LHL on $T_i$:

$$\mathrm{OPT}_i \geq \mathrm{OPT}_i^N \geq 2^i(i + 1) = \mathrm{LB}_i$$

The same proof can be adapted to derive a lower bound for general trees. With $\mathrm{OPT}^N(T)$ the optimal size of a labeling for the pairs in $N$ of the tree $T$ rooted in some $r$, and $T_1, \ldots, T_k$ the

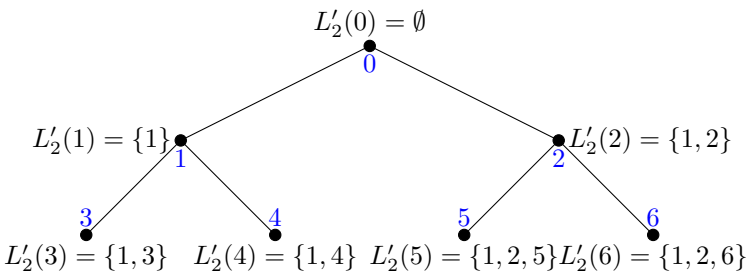



Figure 5: An illustration of labeling $L'_2$.

subtrees rooted at the children of $r$, we have:

$$\mathrm{OPT}(T) \geq \mathrm{OPT}^N(T) \geq \sum_{i=1}^{k} \mathrm{OPT}^N(T_i) + |T| - \max\{|T_i| \mid i = 1, \ldots, k\}$$

### 5.3.2 LHL Approximation

We now introduce a construction that yields a 2-approximation compared to the optimal LHL on complete binary trees $T_i$, $i \geq 1$. We build the labeling recursively. We denote the root by 0, the left child by 1 and the right child by 2. Consider the following labeling $L'_i$ for $T_i$ (note that $L'_i$ is not yet a valid LHL):

- For $i = 1$, $L'_1(0) = \emptyset$, $L'_1(1) = \{1\}$ and $L'_1(2) = \{1, 2\}$.
- For $i = 2$, we consider the labeling $L'_2$ shown in Figure 5.
- For $i \geq 3$, we recursively build the labeling $L'_{i-1}$ for the left and right subtree of $T_i$. We then add 1 to the label set of each node of $T_i$ except the root. Furthermore, we add 2 to the label set of itself.

We can now consider the labeling $L_i$ for $T_i$ where we add 0 and 1 to the label set of the root in $L'_i$. $L_i$ is an LHL for $T_i$. We first prove by induction that labeling $L_i$ satisfies the landmark hub cover property for any pair of nodes of $T_i$ except those containing the root. It is true for $i = 1$ and $i = 2$. Let $i \geq 3$ and assume that it is true for $i - 1$. Consider a pair of nodes of $T_i$ that does not contain 0, 1 or 2. If both nodes are in the same subtree, the landmark hub property is satisfied by induction hypothesis. If they are not in the same subtree, 1 is in each label and so the hub cover property is satisfied. Moreover, pairs of nodes that contain at least one of the children of the root satisfy either the hub or landmark cover property due to 1 in both labels. Finally, adding 1 to the root label set enables to cover the pair of nodes containing the root.

It is a hierarchical LHL by assigning the rank as such: the nodes at a high level (root at the highest level of the tree) have a higher rank and on the same level, the nodes on the left have a higher rank.

This labeling is not optimal. For instance, the labeling in Figure 6 for $i = 3$ has size 37 while our heuristic provides us with a label size of 43. However, the label size of the resulting LHL is within a factor of 2 of the optimum.

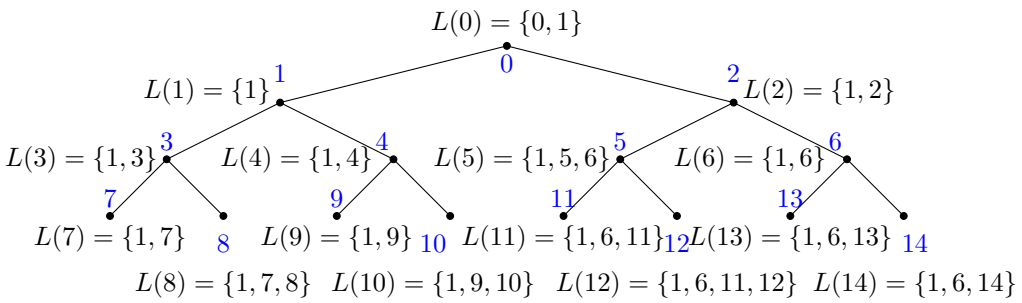



Figure 6: A better labeling for $T_3$.

**Observation 1** *The above construction is a 2-approximation algorithm.*

**Proof:** Let us denote by $s_i$ the size of the labeling $L_i$ and $s'_i$ the size of $L'_i$. We have: $s'_1 = 3$, $s'_2 = 13$ and for $i \geq 3$,

$$\begin{aligned} s'_i &= 2s'_{i-1} + 2^{i+1} - 1 \\ &= (i-2)2^{i+1} + 3 \cdot 2^i + 1 \end{aligned}$$

Finally, $s_i = s'_i + 2$ for $i \geq 1$. Using the lower bound $\mathrm{LB}_i = 2^i(i+1)$ of Section 5.3.1, we have $s_i \leq 2 \cdot \mathrm{LB}_i$ for $i \geq 1$. □

# 6 Experimental Evaluation

This section evaluates the performance of g-HHL, w-HHL and two heuristics for HHL: contraction Hierarchies based heuristic and balanced separator based heuristic. Furthermore, we apply the exact algorithm for HHL on trees and investigate lower and upper bounds. Finally, we implement and evaluate the performance of a simple HLHL heuristic on trees.

We implemented all algorithms in C++ and executed on a single core of an AMD Ryzen 7 3700X processor (clocked at 3.6 GHz) with 128 GB main memory, which had an L3 cache size of 16384K. The operating system was ubuntu 18.04. KaHIP 2.10 (used for computing balanced separators) was compiled using Clang 11.0.0 with OpenMPI 1.4.1.

## 6.1 Results on General Graphs

We consider two prominently used heuristics for HHL, CH based, and the balanced separator based heuristics.

- **CH Based Heuristic.** Methods for Contraction Hierarchies (CH) construct node rankings that can be used for HHL [15]. We always compute the canonical HHL associated with the CH node ranking.
- **Balanced Separator Based Heuristic.** The algorithm based on balanced separators computes a preferably small 1/2-balanced separator $S$ in each connected component and assigns them the next highest rankings. We use KaFFPa in our implementation (a tool

included in KaHIP[24]), which can find small balanced separators in large graphs efficiently in a heuristic fashion. We again compute the canoncial HHL with respect to the obtained node ranking.

To evaluate the resulting label size quality of the heuristics, we compare the results with the label size lower bound from [9]. The authors introduce the notion of efficiency of a pair of nodes and show that it can be used to derive a general lower bound for HL on arbitrary graphs. For $u, v \in V$, let $h_u(v)$ be the number of vertices $x$ whose shortest path from $u$ to $x$ contains $v$, and let $P_{uv}$ be the set of shortest paths from $u$ to $v$. Then the efficiency of each node pair $(u, v)$ is defined as: $\text{eff}(u, v) = \max_{p \in P_{uv}} \max_{w \in p} \min \{h_u(w), h_v(w)\}$.

For the optimal average hub labeling size $L_{\text{avg}}$ for the graph $G = (V, E)$ holds that

$$n \cdot L_{\text{avg}} \geq \sum_{(s,t):P_{st} \neq \emptyset} \frac{1}{\text{eff}(s,t)}. \tag{8}$$

Note that this bound works well for dense graphs, however, it is less tight on sparser graphs. For example, on paths or binary trees, the lower bound above states $n \cdot L_{\text{avg}} \in \Omega(n)$ (where $n$ is the number of nodes), though the optimal hub labeling on these graphs is in $\Omega(n \log n)$.

### 6.1.1 Benchmark Sets

The first benchmark set contains the instances from the PACE challenge 2020[1], which were provided for the treedepth decomposition challenge. The second benchmark set contains the instances from the PACE challenge 2019 [13], which were provided for the vertex cover challenge. The last benchmark set consists of road networks extracted from OpenStreetMap[2] (OSM). Table 4 presents an overview over all benchmark sets. Benchmark sets from the PACE challenge contain a variety of graph densities, while the OSM benchmark set is comparably sparse.

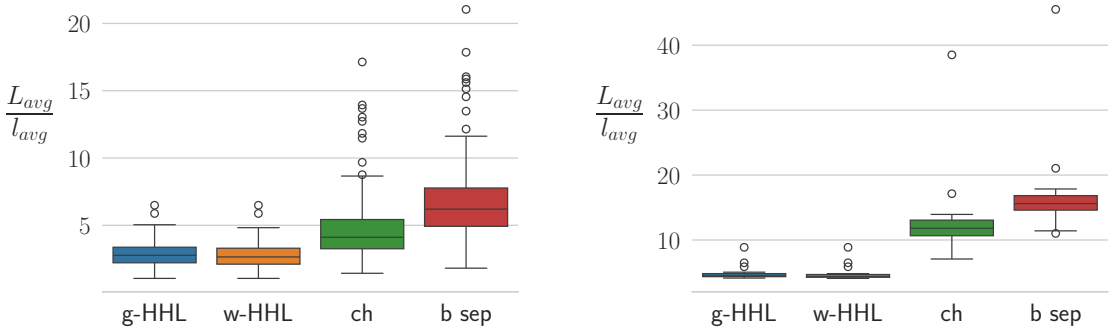
The average label sizes $L_{\text{avg}}$ of the different approaches, compared to the lower bound $l_{\text{avg}}$ given by efficiency, are shown in Figure 7. We can observe that g-HHL and w-HHL compute results of comparable quality. No algorithm could find a label size smaller than four times the efficiency lower bound on the road networks. Recall that the efficiency lower bound is less accurate on sparse graphs. In all PACE instances, the label sizes from both algorithms are within the factor of 7 from the efficiency lower bound, while the average value of $L_{\text{avg}}/l_{\text{avg}}$ is around three on the PACE instances and still lower than 5 for the OSM instances. The algorithm based on CH almost always computes worse results than the previously mentioned algorithms. It performs poorly on the OSM

[1] PACE challenge
[2] OSM

| Benchmark Sets | #Instances | $\lvert V \rvert$ | $\lvert E \rvert$ | $\overline{d}$ |
|---|---|---|---|---|
| PACE 2020 exact | 100 | [10, 491] | [15, 4.100] | [2, 65] |
| PACE 2020 heuristic | 40 | [105, 7.813] | [143, 366.239] | [2, 208] |
| PACE 2019 | 30 | [153, 4.579] | [625, 126.163] | [3, 164] |
| OSM | 15 | [100,4000] | [99,4336] | 2 |

Table 4: Number of nodes, edges and average degree of the graphs in the used benchmark sets.

road networks. In particular, the average ratio to the lower bound is around twelve, and even the best-case instances have a larger label set than the worst-case instances of g-HHL and w-HHL.




(a) PACE instances (b) OSM

Figure 7: Comparison of the resulting average label sizes $L_{\text{avg}}$ and the efficiency lower bound $l_{\text{avg}}$.

The balanced separator based algorithm computes the worst results on average in all instances. On the PACE instances, its average ratio is around six compared to the lower bound, while all other algorithms find smaller hub labels on average. On the road networks, the balanced separator based algorithm could, in the best-performed instances, find a solution within the lower bound by a factor of around 13. It is slightly worse than the worst results of g-HHL and w-HHL. The average ratio of the solution quality of the balanced separator based algorithm compared to the lower bound is around 15, about 3 times as large as g-HHL and w-HHL.

To summarize, we observe that g-HHL and w-HHL compute hub labels of similar size on all inputs. On average, the CH-based algorithm computes solutions smaller than the balanced separator based algorithm but slightly worse hub labels than the previous algorithms. The last algorithm based on balanced separators computes the worst solutions in all instances.

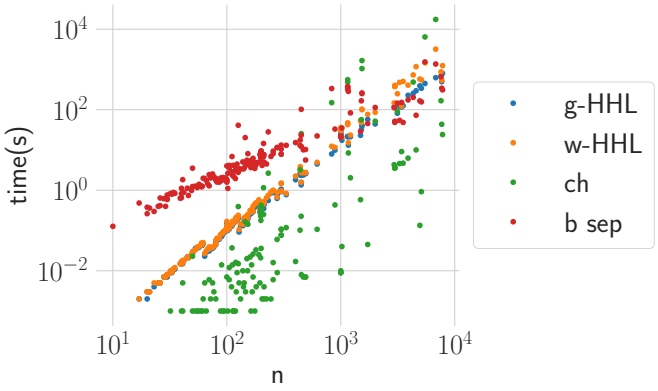



Figure 8: Running time comparison.

The running times of all algorithms are shown in Figure 8. Both g-HHL and w-HHL have very

| Benchmark Set | #Instances | $\lvert V\rvert$ | $\max(\alpha(v, T'))$ |
|---|---|---|---|
| PACE 2020 Heur | 64 | $[100, 1.319.677]$ | 0.66 |
| Random Trees | 66 | $[100, 448.000]$ | 0.64 |

Table 5: Number of instances, number of nodes, and the maximum balance ratio during the execution of g-HHL.

similar running times. In almost all instances, the CH-based approach is the fastest algorithm. However, on some of the dense graphs, the running times of CH are even the longest. This results from the fact that CH is originally designed to work on large and sparse road networks. The last algorithm based on balanced separators is time-consuming in small instances. This is due to the computation of a balanced separator involving the execution of KaHIP, which takes considerable time. However, this approach can be used on large instances since its asymptotical growth is slower than that of all other algorithms.

## 6.2 Results on Trees

To evaluate the performance of Peleg's algorithm and g-HHL on trees, we generated two benchmark sets. The first benchmark set includes extracted shortest path trees from the PACE 2020 heuristic benchmark set. The second one comprises our self-generated random trees, which are constructed from a single node by iteratively adding a new node to a random previous node.

Since both g-HHL and w-HHL behave the same on trees following the results from Section 5.2, it is sufficient to only compare Peleg's algorithm to g-HHL.

We applied Peleg's algorithm and g-HHL to our two benchmark sets and obtained the resulting label sizes $L_p$ and $L_g$. Furthermore, we also measured the balance ratio $\alpha(v, T')$ of each node $v$ to which g-HHL assigned the highest rank in the subtree $T'$ during the execution. Recall that the balance ratio of a node $v$ is definded as the largest connected component size after removing the node $v$ and its adjacent edges divided by $n$, where $n$ is the number of nodes in $T'$.

The table presents an overview over the benchmark sets and the maximum balance ratios. Figure 9 illustrates the resulting label sizes of g-HHL $L_g$ compared to the label size of Peleg's algorithm $L_p$.

The maximum balance ratio $\alpha(v, T')$ in the first benchmark set is 0.64 and 0.66 in the second benchmark set, both were significantly lower than $1/\sqrt{2} \approx 0.707$ from the worst case shown in Section 5. Furthermore, there are only two instances in which $L_g$ is larger than $L_p$, one from each benchmark set. In both instances, the resulting label sizes of g-HHL exceeded the results from Peleg's algorithm by less than 1%. The approximation factor from Theorem 3 depends on the balance ratio. Since g-HHL and w-HHL do not aim to cover the shortest paths by the node that separates the tree in the most balanced way, the approximation factors via Theorem 3 are larger than Peleg's algorithm. However, our results demonstrate that the label sizes of g-HHL are in most cases lower or equal to $L_p$, and are significantly closer to the optimum than the factor of $\frac{2}{\sqrt{2}-1}$ from Theorem 4.

Furthermore, we implemented the exact algorithm for hierarchical hub labeling and applied it on 20000 randomly generated small trees with node sizes lower or equal to 20. Comparing the hub label sizes resulting from g-HHL and Peleg's algorithm to the optimal solution in the same instances, we observed a lower bound of 1.005 and 1.023. The maximum balance ratio $\alpha(v, T')$ of

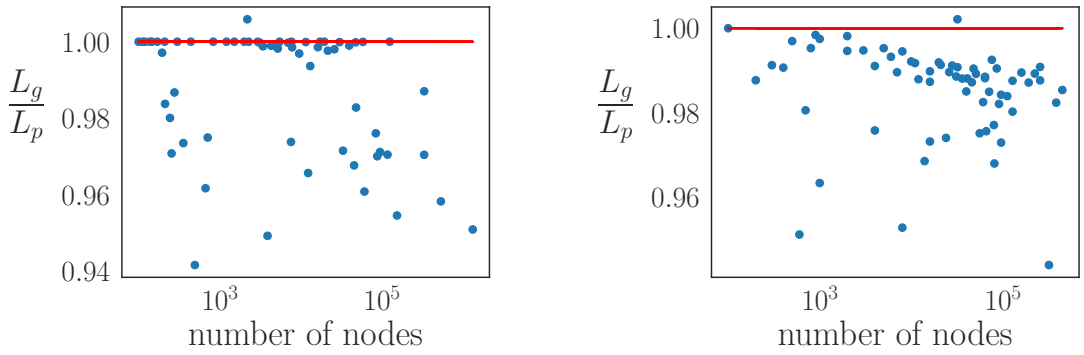



Figure 9: Label size comparison between g-HHL and Peleg's algorithm. Left: Label size comparison on PACE2020 heuristic public benchmark set. Right: Label size comparison on random tree benchmark set.

the exact algorithm never exceed 2/3. Based on this observation, we implemented an algorithm that tries to assign every 2/3-balanced separator with the next highest rank in each iteration and returns the HHL with the smallest label size, which has a running time in $\mathcal{O}(n^{\log n})$. After applying this algorithm to randomly generated trees with node sizes up to 2000, we observed a slightly larger lower bound of 1.034 for the approximation factor of g-HHL. It would be interesting to further narrow the gap between this lower bound and the proven upper bound in future work.

### 6.3 HLHL on Trees

A simple heuristic to compute hierarchical LHL on trees is to contract all degree two chains in the tree and apply g-HHL on the contracted tree, then complete the label set for the contracted nodes. First, we contract each degree two chain in the tree $T$ into a single node on the chain, called a representation. Then we compute the g-HHL for the contracted tree $T'$. Finally, the label set of any contracted node $v$ is the label set of its representation plus $v$. The nodes in $T'$ are ranked higher than the contracted nodes, hence the labelling is hierarchical. For the non-contracted nodes, their hub label is a valid HHL. Furthermore, the representation lies on all maximal shortest paths from any other node to any node in the same degree two chain. By inheriting the label set from the representation, all shortest paths of contracted nodes are also covered. Therefore, the resulting labeling is a valid hierarchical LHL.

We now evaluate the performance of the HLHL heuristic against g-HHL and Peleg's algorithm. Our benchmark sets are the self-generated random trees from the last section and 13 shortest path trees from OSM with sizes ranging from 100 to 10000 nodes. As shown in Figure 10, in the OSM benchmark set the LHL heuristic always computes the smaller labeling than g-HHL and Peleg's algorithm. Since the shortest path trees from OSM contain many degree two chains, in most contracted graphs there are only half as many nodes as in the original graph. However, in the random tree benchmark set, the LHL heuristic computes a smaller labeling than HHL in only 5 out of 66 cases. On this benchmark set less than 30 percent of nodes can be contracted.

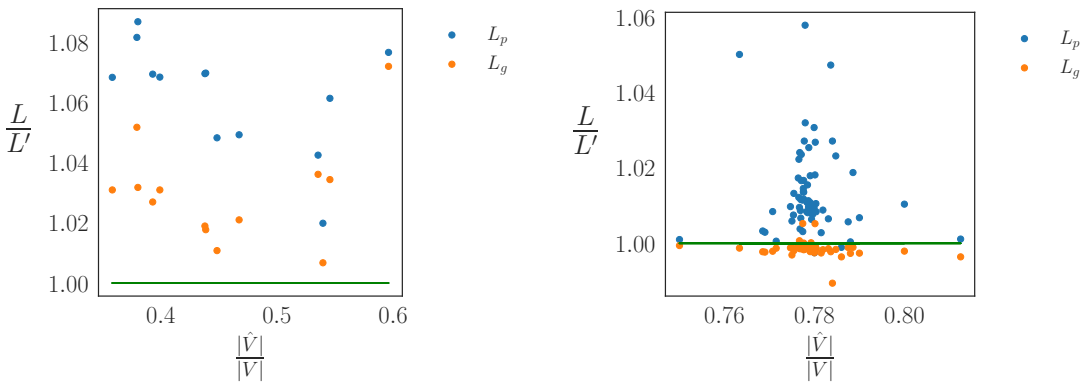



Figure 10: Label size $L_g$ of g-HHL divided by HLHL heuristic label size $L'$, and label size $L_p$ of Peleg's algorithm divided by $L'$. Left: Label size comparison on the OSM shortest path tree benchmark set. Right: Label size comparison on the random tree benchmark set. The x-axis shows the ratio of the number of nodes $|\hat{V}|$ in the contracted graph to the number of nodes $|V|$ in the original graph.

# 7 Conclusions and Future Work

We hproposed novel algorithms and bounds for HHL, which complement and improve existing techniques. Our evaluation indicates that g-HHL and w-HHL perform remarkably well on trees as well as on general graphs. Improving their scalability to make them applicable to larger inputs may be an interesting direction for future work. Also, their approximation factor on trees could potentially be shown to be even smaller than $\frac{2}{\sqrt{2}-1}$. It was proven in [4] that the node with the highest rank in an optimal HHL on trees is a $5/6$-balanced separator. Based on our experiments, it may be possible that optimal solutions adhere to even smaller balance ratios, potentially down to $2/3$.

On general graphs, lower bounds for the approximation factors were investigated in [5]. These bounds show that g-HHL and w-HHL do not provide polylogarithmic approximation factors. However, there are also no approximation hardness results known that would imply that no such algorithm can exist. Thus, it might be worthwhile to search for alternative algorithms that produce a close-to-optimal HHL in polytime.

We have furthermore provided an experimental comparison between HHL and HLHL on different benchmarks and have shown that HLHL indeed provides smaller label sizes on average on shortest path trees of road networks. It would be interesting to further study the theoretical relationship between HL and LHL, in their hierarchical and non-hierarchical variant, on different types of graph classes.

# References

[1] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension and provably efficient shortest path algorithms. *Journal of the ACM (JACM)*, 63(5):1–26, 2016. doi:10.1145/2985473.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *European Symposium on Algorithms*, pages 24–35. Springer, 2012. doi:10.1007/978-3-642-33090-2_4.

[3] T. Akiba, Y. Iwata, K.-i. Kawarabayashi, and Y. Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *2014 Proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 147–154. SIAM, 2014. doi:10.1137/1.9781611973198.14.

[4] H. Angelidakis, Y. Makarychev, and V. 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, Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.94.

[5] M. Babenko, A. V. Goldberg, H. Kaplan, R. Savchenko, and M. Weller. On the complexity of hub labeling. In *Mathematical Foundations of Computer Science 2015: 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II 40*, pages 62–74. Springer, 2015. doi:10.1007/978-3-662-48054-0_6.

[6] J. Barceló, H. Grzybowska, and S. Pardo. Vehicle routing and scheduling models, simulation and city logistics. *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies*, pages 163–195, 2007. doi:10.1007/978-0-387-71722-7_8.

[7] R. Bauer, T. Columbus, I. Rutter, and D. Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016. doi:10.1016/j.tcs.2016.07.003.

[8] J. Cauvi, R. Li, and S. Storandt. Landmark hub labeling: Improved bounds and faster query answering. In *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.ATMOS.2024.1.

[9] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003. doi:10.1137/S0097539702403098.

[10] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Robust exact distance queries on massive networks. *Microsoft Research, USA, Tech. Rep*, 2, 2014.

[11] D. Delling, A. V. Goldberg, R. Savchenko, and R. 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. doi:10.1007/978-3-319-07959-2_22.

[12] J. Du, B. Shen, and M. A. Cheema. Ultrafast euclidean shortest path computation using hub labeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12417–12426, 2023. doi:10.1609/aaai.v37i10.26463.

[13] M. A. Dzulfikar, J. K. Fichte, and M. Hecher. Pace2019: Track 1 - vertex cover instances, July 2019. doi:10.5281/zenodo.3368306.

[14] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

[15] A. V. Goldberg, I. Razenshteyn, and R. Savchenko. Separating hierarchical and general hub labelings. In *Mathematical Foundations of Computer Science 2013*, pages 469–479. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40313-2_42.

[16] A. Kosowski and L. 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. doi:10.1137/1.9781611974782.95.

[17] K. Lakhotia, Q. Dong, R. Kannan, and V. Prasanna. Planting trees for scalable and efficient canonical hub labeling. *arXiv preprint arXiv:1907.00140*, 2019. doi:10.14778/3372716.3372722.

[18] S. Maniu, P. Senellart, and S. Jog. An experimental study of the treewidth of real-world graph data. In *22nd International Conference on Database Theory (ICDT 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[19] S. Nazari, M. R. Meybodi, M. A. Salehigh, and S. 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. doi:10.1109/ICINIS.2008.147.

[20] K. P. Nepal and D. 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. doi:10.1080/03081060500053509.

[21] T. Opsahl, F. Agneessens, and J. Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social networks*, 32(3):245–251, 2010. doi:10.1016/j.socnet.2010.03.006.

[22] D. Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000. doi:10.1002/(SICI)1097-0118(200003)33:3<167::AID-JGT7>3.0.CO;2-5.

[23] S. Peyer, D. Rautenbach, and J. Vygen. A generalization of dijkstra's shortest path algorithm with applications to vlsi routing. *Journal of Discrete Algorithms*, 7(4):377–390, 2009. doi:10.1016/j.jda.2007.08.003.

[24] P. Sanders and C. Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *LNCS*, pages 164–175. Springer, 2013. doi:10.1007/978-3-642-38527-8_16.

[25] S. Storandt. Algorithms for landmark hub labeling. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.5.

[26] S. Storandt. Scalable landmark hub labeling for optimal and bounded suboptimal pathfinding. In *33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*, 2024.

[27] N. Wang and J. Li. Shortest path routing with risk control for compromised wireless sensor networks. *IEEE Access*, 7:19303–19311, 2019. doi:10.1109/ACCESS.2019.2897339.