Simpler, faster and shorter labels for distances in graphs

Stephen Alstrup [*] Cyril Gavoille [†] Esben Bistrup Halvorsen [‡] Holger Petersen [§]

April 20, 2015

Abstract

We consider how to assign *labels* to any undirected graph with n nodes such that, given the labels of two nodes and no other information regarding the graph, it is possible to determine the distance between the two nodes. The challenge in such a *distance labeling scheme* is primarily to minimize the maximum label lenght and secondarily to minimize the time needed to answer distance queries (decoding). Previous schemes have offered different trade-offs between label lengths and query time. This paper presents a simple algorithm with shorter labels and shorter query time than any previous solution, thereby improving the state-of-the-art with respect to both label length and query time in one single algorithm. Our solution addresses several open problems concerning label length and decoding time and is the first improvement of label length for more than three decades.

More specifically, we present a distance labeling scheme with labels of length $\frac{\log 3}{2}n + o(n)$ bits[1] and constant decoding time. This outperforms all existing results with respect to both size and decoding time, including Winkler's (Combinatorica 1983) decade-old result, which uses labels of size $(\log 3)n$ and $O(n/\log n)$ decoding time, and Gavoille et al. (SODA'01), which uses labels of size $11n + o(n)$ and $O(\log \log n)$ decoding time. In addition, our algorithm is simpler than the previous ones. In the case of integral edge weights of size at most W, we present almost matching upper and lower bounds for the label size ℓ: $\frac{1}{2}(n-1)\log\left\lceil \frac{W}{2} + 1\right\rceil \leq \ell \leq \frac{1}{2}n\log(2W+1) + O(\log n \cdot \log(nW))$. Furthermore, for r-additive approximation labeling schemes, where distances can be off by up to an additive constant r, we present both upper and lower bounds. In particular, we present an upper bound for 1-additive approximation schemes which, in the unweighted case, has the same size (ignoring second order terms) as an adjacency labeling scheme, namely $n/2$. We also give results for bipartite graphs as well as for exact and 1-additive distance oracles.

[*]Depart. of Computer Science, University of Copenhagen, Denmark. E-mail: stephen.alstrup.private@gmail.com.
[†]LaBRI - Université de Bordeaux, France. E-mail: gavoille@labri.fr.
[‡]Depart. of Computer Science, University of Copenhagen, Denmark. E-mail: esben@bistruphalvorsen.dk.
[§]E-mail: dr.holger.petersen@googlemail.com.
[1]Throughout the paper, all logarithms are in base 2.

1 Introduction

A *distance labeling scheme* for a given family of graphs assigns *labels* to the nodes of each graph from the family such that, given the labels of two nodes in the graph and no other information, it is possible to determine the shortest distance between the two nodes. The labels are assumed to be composed of bits. The main goal is to make the worst-case label size as small as possible while, as a subgoal, keeping query (decoding) time under control. The problem of finding implicit representations with small labels for specific families of graphs was first introduced by Breuer [13, 14], and efficient labeling schemes were introduced in [43, 51].

1.1 Distance labeling

For an undirected, unweighted graph, a naïve solution to the distance labeling problem is to let each label be a table with the $n-1$ distances to all the other nodes, giving labels of size around $n \log n$ bits. For graphs with bounded degree Δ it was shown [14] in the 1960s that labels of size $2n\Delta$ can be constructed such that two nodes are adjacent whenever the Hamming distance [41] of their labels is at most $4\Delta - 4$. In the 1970s, Graham and Pollak [38] proposed to label each node with symbols from $\{0, 1, *\}$, essentially representing nodes as corners in a "squashed cube", such that the distance between two nodes exactly equals the Hamming distance of their labels (the distance between $*$ and any other symbol is set to 0). They conjectured the smallest dimension of such a squashed cube (the so-called *Squashed cube conjecture*), and their conjecture was subsequently proven by Winkler [65] in the 1980s. This reduced the label size to $\lceil (n-1) \log 3 \rceil$, but the solution requires $O(n/\log n)$ query time to decode distances. Combining [43] and [50] gives a lower bound of $\lceil n/2 \rceil$ bits. A different distance labeling scheme of size of $11n + o(n)$ and with $O(\log \log n)$ decoding time was proposed in [36]. The article also raised it as open problem to find the right label size. Later in [63] the algorithm from [36] was modified, so that the decoding time was further reduced to $O(\log^* n)$ with slightly larger labels, although still of size $O(n)$. This article raised it as an open problem whether the query time can be reduced to constant time. Having distance labeling with short labels and simultaneous fast decoding time is a problem also addressed in text books such as [58]. Some of our are solutions are simple enough to replace material in text books.

Addressing the aforementioned open problems, we present a distance labeling scheme with labels of size $\frac{\log 3}{2}n + o(n)$ bits and with constant decoding time. See Table 1 and Figure 1 for an overview.

Space	**Decoding time**	**Year**	**Reference**
$(\log 3)n$	$O(n/\log n)$	1972/1983	[38, 65]
$11n$	$O(\log \log n)$	2001	[36]
$cn, c > 11$	$O(\log^* n)$	2011	[63]
$\frac{\log 3}{2}n$	$O(1)$	2015	this paper

Table 1: Unweighted undirected graphs. Space is listed presented without second order terms. A graphical presentation of the results is given in Figure 1

Distance labeling schemes for various families of graphs exist, e.g., for trees [5, 55], bounded tree-width [36], distance-hereditary [34], bounded clique-width [21], some non-positively curved plane [18], interval [35] and permutation graphs [10]. In [36] it is proved that distance labels require $\Theta(\log^2 n)$ bits for trees, $O(\sqrt{n} \log n)$ and $\Omega(n^{1/3})$ bits for planar graphs, and $\Omega(\sqrt{n})$ bits for bounded degree graphs. In an unweighted graph, two nodes are adjacent iff their distance is 1. Hence, lower bounds for adjacency labeling apply to distance labeling as well, and adjacency lower bounds can be achieved by reduction [43] to induced-universal graphs, e.g. giving $\frac{n}{2}$ and $\frac{n}{4}$ for general and bipartite graphs, respectively. An overview of adjacency labeling can be found in [7].

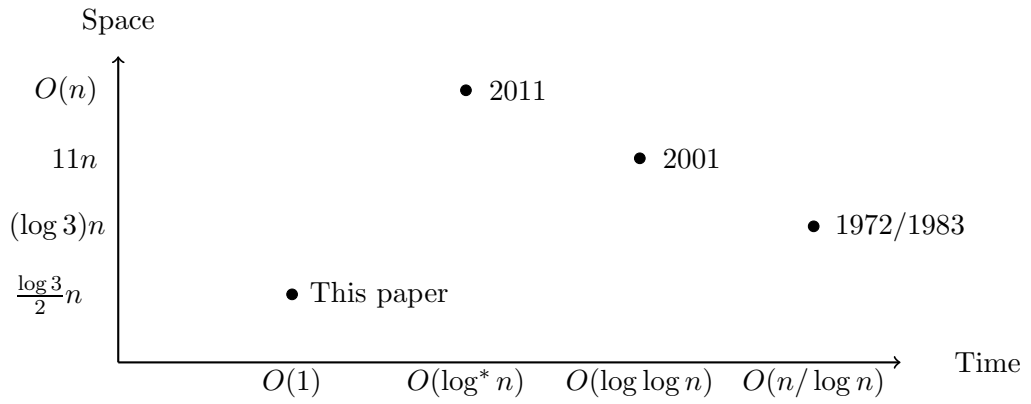

Figure 1: A graphical representation of the results from Table 1.

Various computability requirements are sometimes imposed on labeling schemes [2, 43, 45]. This paper assumes the RAM model and mentions the time needed for decoding in addition to the label size.

1.2 Overview of results

For weighted graphs we assume integral edge weights from $[1, W]$. Letting each node save the distance to all other nodes would require a scheme with labels of size $O(n \log(nW))$ bits. Let $\text{dist}_G(x, y)$ denote the shortest distance in G between nodes x and y. An r-additive approximation scheme returns a value $\text{dist}'_G(x, y)$, where $\text{dist}_G(x, y) \leq \text{dist}'_G(x, y) \leq \text{dist}_G(x, y) + r$.

Throughout this paper we will assume that $\log W = o(\log n)$ since otherwise the naïve solution mentioned above will be as good as our solution. Ignoring second order terms, we can for general weighted graphs and constant decoding time achieve upper and lower bounds for label length as stated in Table 2. For bipartite graphs we also show a lower bound of $\frac{1}{4}n \log \lfloor 2W/3 + 5/3 \rfloor$ and an upper bound of $\frac{1}{2}n$ whenever $W = 1$.

Problem	**Lower bound**	**Upper bound**
General graphs	$\frac{1}{2}(n-1)\log\lceil W/2+1\rceil$	$\frac{1}{2}n\log(2W+1)$

Table 2: General graphs with weights from $[1, W]$, where $\log W = o(\log n)$. The upper bound has an extra $o(n)$ term, and decoding takes constant time.

We present, as stated in Table 3, several trade-offs between decoding time, edge weight W, and space needed for the second order term.

Time	**Second order term**	**W**
N/A	$O(\log n \cdot \log(nW))$	Any value
$O(n)$	$O(\log^2 n)$	$O(1)$
$O(1)$	$O(\frac{n}{\log n}\log(2W+1)(\log\log n + \log W))$	$2^{o(\log n)}$

Table 3: Second order term for the upper bound for general graphs (in Table 2). The results also hold for the $n/2$ labels in the unweighted, bipartite case. It may be possible to relax the restriction $W = O(1)$ if the word "finite" in Lemma 2.2 below from [24] does not mean "constant".

We also show that, for any $k, D \geq 0$ with $\log k = o(\log n)$ and $D \leq 2(k+1)W - 1$, there exists a $(2kW + \left\lceil \frac{D}{2(k+1)W - D} \right\rceil)$-additive distance scheme using labels of size $\frac{1}{2(k+1)}n \log(2(k+1)W + 1 - D) + O(\log n \cdot \log(nW))$ bits.

Finally, we present lower bounds for approximation schemes. In particular, for $r < 2W$ we prove that labels of $\Omega(n \log(W/(r+1)))$ bits are required for an r-additive distance labeling scheme.

1.3 Approximate distance labeling schemes and oracles

Approximate distance labeling schemes are well studied; see e.g., [36, 39, 40, 55, 62]. For instance, graphs of doubling dimension [59] and planar graphs [60] both enjoy schemes with polylogarithmic label length which return approximate distances below a $1+\varepsilon$ factor of the exact distance. Approximate schemes that return a small additive error have also been investigated, e.g. in [17, 33, 48]. In [32], lower and upper bounds for r-additive schemes, $r \leq 2$, are given for chordal, AT, permutation and interval graphs. For general graphs the current best lower bound [32] for $r \geq 2$-additive scheme is $\Omega(\sqrt{n/r})$. For $r = 1$, one needs $\frac{1}{4}n$ bits since a 1-additive scheme can answer adjacency queries in bipartite graphs. Using our approximative result, we achieve, by setting $k = 0$ and $D = W = 1$, a 1-additive distance labeling scheme which, ignoring second order terms, has the same size (namely $\frac{1}{2}n$ bits) as an optimal adjacency labeling scheme. Somehow related, [11] studies labeling schemes that preserve exact distances between nodes with minimum distance P, giving an $O((n/P)\log^2 n)$ bit solution.

Approximate distance oracles introduced in [62] use a global table (not necessarily labels) from which approximate distance queries can be answered quickly. One can naïvely use the n labels in a labeling scheme as a distance oracle (but not vice versa). For unweighted graphs, we achieve constant query time for 1-additive distance oracles using $\frac{1}{2}n^2 + o(n^2)$ bits in total, matching (ignoring second order terms) the space needed to represent a graph. Other techniques only reduce space for r-additive errors for $r > 1$. For exact distances in weighted graphs, our solution achieves $\frac{1}{2}n^2 \log(2W+1) + o(n^2)$ bits for $\log W = o(\log n)$. This relaxes the requirement of $W = O(1)$ in [28] (and slightly improves the space usage in that paper).

1.4 Second order terms are important

Chung's solution in [19] gives labels of size $\log n + O(\log\log n)$ for adjacency labeling in trees, which was improved to $\log n + O(\log^* n)$ in [9] and in [12, 29, 30, 44] to $\log n + O(1)$ for various special cases. A recent STOC'15 paper [7] improves label size for adjacency in generel graphs from $n/2 + O(\log n)$ to $n/2 + O(1)$. Likewise, the second order term for ancestor relationship is improved in a sequence of STOC/SODA papers [2, 8, 4, 30, 31] (and [1]) to $\Theta(\log\log n)$, giving labels of size $\log n + \Theta(\log\log n)$.

Somewhat related, *succinct data structures* (see, e.g., [24, 26, 27, 52, 53]) focus on the space used in addition to the information theoretic lower bound, which is often a lower order term with respect to the overall space used.

1.5 Labeling schemes in various settings and applications

By using labeling schemes, it is possible to avoid costly access to large global tables, computing instead locally and distributed. Such properties are used, e.g., in XML search engines [2], network routing and distributed algorithms [22, 25, 61, 62], dynamic and parallel settings [20, 47], graph representations [43], and other applications [45, 46, 54, 55, 56]. From the SIGMOD, we see labeling schemes used in [3, 42] for shortest path queries and in [16] for reachability queries. Finally, we observe that compact 2-hop labeling (a specific distance labeling scheme) is central for computing exact distances on real-world networks with millions of arcs in real-time [23].

1.6 Outline of the paper

Section 3 illustrates some of our basic techniques. Sections 4 and 5 present our upper bounds for exact distance labeling schemes for general graphs. Section 6 presents upper bounds for approximate

distances. Our lower bounds are rather simple counting arguments with reduction to adjacency and have been placed in Appendix A.

2 Preliminaries

Trees. Given a rooted tree T and a node u of T, denote by T_u be the subtree of T consisting of all the descendants of u (including itself). The *depth* of u is the number of edges on the unique simple path from u to the root of T. For any rooted subtree A of T, denote by $\text{root}(A)$ the root of A, as the node of A with smallest depth. Denote by $A^* = A \setminus \{\text{root}(A)\}$ the forest obtained from A by removing its root. Denote by $|A|$ the number of nodes of A: hence, $|A^*|$ represents its number of edges. Denote by $\text{parent}_T(u)$ the parent of the node u in T. Let $T[u, v]$ denote the nodes on the simple path from u to v in T. The variants $T(u, v]$ and $T[u, v)$ denote the same path without the first and last node, respectively.

Graphs. Throughout we assume graphs to be connected. If a graph is not connected, we can add $O(\log n)$ bits to each label, indicating the connected component of the node, and then handle components separately. We denote by $\text{dist}_G(u, v)$ the minimum distance (counted with edge weights) of a path in G connecting the nodes u and v.

Representing numbers and accessing them. We will need to encode numbers with base different from 2 and sometimes compute prefix sums on a sequence of numbers. We apply some existing results:

Lemma 2.1 ([49])**.** *A table with n integral entries in $[-k, k]$ can be represented in a data structure of $O(n \log k)$ bits to support prefix sums in constant time.*

Lemma 2.2 ([24])**.** *A table with n elements from a finite alphabet σ can be represented in a data structure of $\lceil n \log |\sigma| \rceil$ bits, such that any element of the table can be read or written in constant time. The data structure requires $O(\log n)$ precomputed word constants.*

Lemma 2.3 (simple arithmetic coding)**.** *A table with n elements from an alphabet σ can be represented in a data structure of $\lceil n \log |\sigma| \rceil$ bits.*

3 Warm-up

This section presents, as a warm-up, a distance labeling scheme which does not achieve the strongest combination of label size and decoding time, but which uses some of the techniques that we will employ later to achieve our results. For nodes x, u, v, define

$$\delta_x(u, v) = \text{dist}_G(x, v) - \text{dist}_G(x, u).$$

Note that the triangle inequailty entails that

$$-\text{dist}_G(u, v) \leq \delta_x(u, v) \leq \text{dist}_G(u, v).$$

In particular, $\delta_x(u, v) \in [-W, W]$ whenever u, v are adjacent.

Given a a path $v_0, \ldots, v_t$ of nodes in G, the telescoping property of δ_x-values means that

$$\delta_x(v_0, v_t) = \sum_{i=1}^{t} \delta_x(v_{i-1}, v_i).$$

Since v_{i-1} and v_i are adjacent, we can encode the δ_x-values above as a table with t entries, in which each entry is a an element from the alphabet $[-W, W]$ with $2W + 1$ values. Using Lemma 2.3 we can

encode this table with $\lceil t \log(2W+1) \rceil$ bits. Note that we can compute $\mathrm{dist}_G(x, v_t)$ from $\mathrm{dist}_G(x, v_0)$ by adding a prefix sum of the sequence of δ_x-values:

$$\mathrm{dist}_G(x, v_t) = \mathrm{dist}_G(x, v_0) + \sum_{i=1}^{t} \delta_x(v_{i-1}, v_i).$$

The *Hamiltonian number* of G is the number $h(G)$ of edges of a Hamiltonian walk in G, i.e. a closed walk of minimal length (counted without weights) that visits every node in G. It is well-known that $n \leq h(G) \leq 2n-2$, the first inequality being an equality iff G is Hamiltonian, and the latter being an equality iff G is a tree (in which case the Hamiltonian walk is an Euler tour); see [15, 37].

Consider a Hamiltonian walk $v_0, \ldots, v_{h-1}$ of length $h = h(G)$. Given nodes x, y from G, we can find i, j such that $x = v_i$ and $y = v_j$. Without loss of generality we can assume that $i \leq j$. If $j \leq i + h/2$, we can compute $\mathrm{dist}_G(x, y)$ as the sum of at most $\lfloor h/2 \rfloor$ δ_x-values:

$$\mathrm{dist}_G(x, y) = \mathrm{dist}_G(v_i, v_j) = \sum_{k=i}^{j-1} \delta_x(v_k, v_{k+1}).$$

If, on the other hand, $j > i + h/2$, then we can compute $\mathrm{dist}_G(x, y)$ as the sum of at most $\lfloor h/2 \rfloor$ δ_y-values:

$$\mathrm{dist}_G(x, y) = \mathrm{dist}_G(v_j, v_i) = \sum_{k=j}^{i-1} \delta_y(v_k, v_{k+1}),$$

where we have counted indices modulo h in the last expression. This leads to the following distance labeling scheme. For each node x in G, assign a label $\ell(x)$ consisting of

- a number $i \in [0, h-1]$ such that $x = v_i$; and
- the $\lfloor h/2 \rfloor$ values $\delta_x(v_k, v_{k+1})$ for $k = i, \ldots, i + \lfloor h/2 \rfloor - 1 \pmod h$.

From the above discussion it follows that the labels $\ell(x)$ and $\ell(y)$ for any two nodes x, y are sufficient to compute $\mathrm{dist}_G(x, y)$.

We can encode $\ell(x)$ with $\lceil \frac{1}{2} h \log(2W+1) \rceil + \lceil \log h \rceil$ bits using Lemma 2.3. If G is Hamiltonian, this immediately gives a labeling scheme of size $\lceil \frac{1}{2} n \log(2W+1) \rceil + \lceil \log n \rceil$. In the general case, we get size $\lceil (n-1) \log(2W+1) \rceil + \lceil \log n \rceil$, which for $W = 1$ matches Winkler's [65] result when disregarding second order terms. Theorem 4.1 in the next section shows that it is possible to obtain labels of size $\frac{1}{2} n \log(2W+1) + O(\log n \cdot \log(nW))$ even in the general case. Theorem 5.3 in the section that follows shows that we can obtain constant time decoding with $o(n)$ extra space.

4 A scheme of size $\frac{1}{2} n \log(2W+1)$

We now show how to construct a distance labeling scheme of size $\frac{1}{2} n \log(2W+1) + O(\log n \cdot \log(nW))$.

First, we recall the *heavy-light decomposition* of trees [57]. Let T be a rooted tree. The nodes of T are classified as either *heavy* or *light* as follows. The root r of T is light. For each non-leaf node v, pick one child w where $|T_w|$ is maximal among the children of v and classify it as heavy; classify the other children of v as light. The *apex* of a node v is the nearest light ancestor of v. By removing the edges between light nodes and their parents, T is divided into a collection of *heavy paths*. Any given node v has at most $\log n$ light ancestors (see [57]), so the path from the root to v goes through at most $\log n$ heavy paths.

Now, enumerate the nodes in T in a depth-first manner where heavy children are visited first. Denote the number of a node v by $\mathrm{dfs}(v)$. Note that nodes on a heavy path will have numbers in consecutive

order; in particular, the root node r will have number $\mathrm{dfs}(r) = 0$, and the nodes on its heavy path will have numbers $0, 1, \ldots$. Assign to each node v a label $\ell_T(v)$ consisting of the sequence of dfs-values of its first and last ancestor on each heavy path, ordered from the top of the tree and down to v. Note that the first ancestor on a heavy path will be the apex of that heavy path and will be light, whereas the last ancestor on a heavy path will be the parent of the apex of the subsequent heavy path. This construction is similar to the one used in [6] for nearest common ancestor (NCA) labeling schemes, although with larger sublabels. Indeed, the label $\ell_T(v)$ is a sequence of at most $2 \log n$ numbers from $[0, n[$. We can encode this sequence with $O(\log^2 n)$ bits.

Suppose that the node v has label $\ell_T(v) = (l_1, h_1, \ldots, l_t, h_t)$, where $l_1 = \mathrm{dfs}(r) = 0$ and $h_t = \mathrm{dfs}(v)$ and where l_i, h_i are the numbers of the first and last ancestor, respectively, on the i'th heavy path visited on the path from the root to v. Since nodes on heavy paths are consecutively enumerated, it follows that the nodes on the path from the root to v are enumerated

$$0 = l_1, \ldots, h_1, l_2, \ldots, h_2, \ldots, l_t, \ldots, h_t,$$

where duplicates may occur in the cases where $l_i = h_i$, which happens when the first and last ancestor on a heavy path coincide.

In addition to the label $\ell_T(v)$, we also store the label $\ell'_T(v)$ consisting of the sequence of distances $\mathrm{dist}_G(l_i, v)$ and $\mathrm{dist}_G(h_i, v)$. This label is a sequence of at most $2 \log n$ numbers smaller than nW, and hence we can encode $\ell'_T(v)$ with $O(\log n \cdot \log(nW))$ bits. Combined, $\ell_T(v)$ and $\ell'_T(v)$ can be encoded with $O(\log n \cdot \log(nW))$ bits.

Now consider a connected graph G with shortest-path tree T rooted at some node r. Using the above enumeration of nodes, we can construct a distance labeling scheme in the same manner as in Section 3, except that instead of using a Hamiltonian path, we use the dfs-enumeration of nodes in T from above, and we save only δ_x-value between nodes and their parents, using $\lceil \frac{1}{2} n \log(2W + 1) \rceil$ bits due to Lemma 2.3. More specifically, for each node x, we assign a label $\ell(x)$ consisting of

- the labels $\ell_T(x)$ and $\ell'_T(x)$ as described above; and
- the $\lfloor n/2 \rfloor$ values $\delta_x(\mathrm{parent}(v), v)$ for all v with $\mathrm{dfs}(x) < \mathrm{dfs}(v) \leq \mathrm{dfs}(x) + \lfloor n/2 \rfloor \pmod n$.

We can encode the above with $\frac{1}{2} n \log(2W + 1) + O(\log n \cdot \log(nW))$ bits.

Given nodes $x \neq y$, either $\ell(x)$ will contain $\delta_x(\mathrm{parent}(y), y)$ or $\ell(y)$ will contain $\delta_y(\mathrm{parent}(x), x)$. Without loss of generality, we may assume that $\ell(x)$ contains $\delta_x(\mathrm{parent}(y), y)$. Let z denote the nearest common ancestor of x and y. Note that z must be the last ancestor of either x or y on some heavy path, meaning that $\mathrm{dfs}(z)$ appears in either $\ell_T(x)$ or $\ell_T(y)$. By construction of depth-first-search, a node v on the path from (but not including) z to (and including) y will have a dfs-number $\mathrm{dfs}(v)$ that satisfies the requirements to be stored in $\ell(x)$. Thus, $\ell(x)$ must, in fact, contain δ_x-values for all nodes in $T_{(z,y]}$.

Next, note that, since T is a shortest-path tree, $\mathrm{dist}_G(x, z) = \mathrm{dist}_T(x, z)$. Now, if z appears in $\ell_T(x)$, we can obtain $\mathrm{dist}_T(x, z)$ directly from $\ell'_T(x)$; else, z must appear in $\ell_T(y)$, and we can then obtain $\mathrm{dist}_T(z, y)$ from $\ell'_T(y)$ and compute $\mathrm{dist}_T(x, z) = \mathrm{dist}_T(x, r) - \mathrm{dist}_T(r, y) + \mathrm{dist}_T(z, y)$. In either case, we can now compute the distance in G between x and y as

$$\mathrm{dist}_G(x, y) = \mathrm{dist}_G(x, z) + \sum_{v \in T(z,y]} \delta_x(\mathrm{parent}(v), v).$$

The label of x contains all the needed δ_x-values, and $\ell_T(x)$ and $\ell_T(y)$ combined allows us to determine the dfs-numbers of the nodes on $T(z, y]$, so that we know exactly which δ_x-values from x's label to pick out. Thus we have proved:

Theorem 4.1. *There exists a distance labeling scheme for graphs with label size* $\frac{1}{2} n \log(2W + 1) + O(\log n \cdot \log(nW))$.

This gives us the first row of Table 3. To obtain the second row, we encode the δ_x values with Lemma 2.2. Doing this we can access each value in constant time and simply traverse in $O(n)$ time the path from y to z, adding δ_x values along the way. Note, however, that Lemma 2.2 only applies for $W = O(1)$. Saving the δ_x-values in a prefix sum structure as described in Lemma 2.1, we can compute the sum using $\log n$ look-ups. The next section describes how we can avoid spending $O(\log n)$ time (or more) on this, while still keeping the same label size.

For unweighted ($W = 1$), bipartite graphs, δ_x-values between adjacent nodes can never be 0, which means that we only need to consider two rather than three possible values. Thus, we get label size $\frac{1}{2}n + O(\log^2 n)$ instead in this case. We shall give no further mention to this in the following.

5 Constant query time

Let T be any rooted spanning tree of the connected graph G with n nodes. We create an edge-partition $\mathcal{T} = \{T_1, T_2, \ldots\}$ of T into rooted subtrees, called *micro trees*. Each micro tree has at most β edges, and the number of micro trees is $|\mathcal{T}| = O(n/\beta)$. We later choose the value of β. For completeness we give a proof (Lemma B.1) in the appendix of the existence of such a construction. Observe that the collection $\{T_i^*\}_{i\geq 1}$ forms a partition of the nodes of T^*. As the parent relationship in T_i coincides with the one of T, we have $\text{parent}_{T_i}(u) = \text{parent}_T(u)$ for all $u \in T_i^*$.

For every node $u \in T^*$, we denote by $i(u)$ the unique index i such that $u \in T_i^*$. For a node u of T^* we let $\text{MicroRoot}(u) = \text{root}(T_{i(u)})$, and for $r = \text{root}(T)$ let $\text{MicroRoot}(r) = r$.

Define the *macro tree* M to have node set $\{\text{MicroRoot}(u) \mid u \in G\}$ and an edge between $\text{MicroRoot}(u)$ and $\text{MicroRoot}(\text{MicroRoot}(u))$ for all $u \neq r$.

By construction, M has $O(n/\beta)$ nodes.

Our labeling scheme will compute the distance from x to y as

$$\text{dist}_G(x, y) = \text{dist}_G(x, r) + \delta_x(r, \text{MicroRoot}(y)) + \delta_x(\text{MicroRoot}(y), y).$$

The first addend, $\text{dist}_G(x, r)$, is saved as part of x's label using $\log n + \log W$ bits. The second addend can be computed as a sum of δ_x-values for nodes in the macro tree and is hence referred to as the *macro sum*. The third addend can be computed as a sum of δ_x-values for nodes inside y's micro tree and is hence referred to as the *micro sum*. The next two sections explain how to create data structures that allow us to compute these values in constant time.

5.1 Macro sum

Consider the macro tree M with $O(n/\beta)$ nodes. As mentioned in Section 3 there exists a Hamiltonian walk $v_0, \ldots, v_{h-1}$ of length $h = O(n/\beta)$, where we can assume that $v_0 = r$. Given nodes $x, y \in G$, consider a path in M along such a Hamiltonian walk from r to $\text{MicroRoot}(y)$. This is a subpath $v_0, \ldots, v_t$ of the Hamiltonian walk, where t is chosen such that $v_t = \text{MicroRoot}(y)$. Note that

$$\delta_x(r, \text{MicroRoot}(y)) = \delta_x(v_0, v_t) = \sum_{i=0}^{t-1} \delta_x(v_i, v_{i+1}).$$

Since each edge in M connects two nodes that belong to the same micro tree, and the distance within each micro tree is a most βW, we have that $\delta_x(v_i, v_{i+1}) \in [-\beta W, \beta W]$ for all i. Using Lemma 2.1 we can store these δ_x-values in a data structure, PreFix_x, of size $O((n/\beta)\log(2\beta W + 1)) = O(n \log(\beta W)/\beta)$ such that prefix sums can be computed in constant time. This data structure is stored in x's label. An index t with $v_t = \text{MicroRoot}(y)$ is stored in y's label using $O(\log(n/\beta))$ bits. These two pieces of information combined allow allows us to compute $\delta_x(r, \text{MicroRoot}(y))$ for all y.

Label summary: For a (pre-selected) Hamiltonian walk $v_0, \ldots, v_{h-1}$ in M, we store in the label of each node x a datastructure PreFix_x of size $O(n \log(\beta W)/\beta)$ such that prefix sums in the form

$\sum_{i=0}^{t-1} \delta_x(v_i, v_{i+1})$ can be computed in constant time. In addition, we store in the label of x an index $m(x)$ such that $v_{m(x)} = \text{MicroRoot}(x)$, which requires $O(\log(n/\beta))$ bits.

5.2 Micro sum

For any node $v \neq r$, define

$$\delta_x(v) = \delta_x(\text{parent}_T(v), v)$$

Note that, for a node $y \in T_i^*$, $\delta_x(\text{MicroRoot}(y), y)$ is the sum of the values $\delta_x(v_j)$ for all nodes $v_j \in T_i^*$ lying on the path from $\text{MicroRoot}(y)$ to y. Each of these δ_x-values is a number in $[-W, W]$.

For each i, order the nodes in T_i^* in any order. For each node x and index i, let $\delta_x(T_i^*) = (\delta_x(v_1), \ldots, \delta_x(v_{|T_i^*|}))$, where $v_1, \ldots, v_{|T_i^*|}$ is the ordered sequence of nodes from T_i^*. We will construct our labels such that x's label stores $\delta_x(T_i^*)$ for half of the total set of delta values (we will see how in the next section), and such that y's label stores information about for which j's the node v_j lies on the path between $\text{MicroRoot}(y)$ and y. With these two pieces of information, we can compute $\delta_x(\text{MicroRoot}(y), y)$ as described above.

We define $f(W) = 2W + 1$. The sequence $\delta_x(T_i^*)$ consists of $|T_i^*|$ values from $[-W, W]$ can be encoded with $|T_i^*| \lceil \log f(W) \rceil$ bits. To store this more compactly, we will use an injective function, as described in Lemma 2.3 that maps every sequence of t integers from $[-W, W]$ into a bit string of length $\lceil t \log f(W) \rceil$. Denote by $\text{code}(\delta_x(T_i^*))$ such an encoding of the sequence $\delta_x(T_i^*)$ to a bit string oflength $\lceil |T_i^*| \log f(W) \rceil \leq \lceil \beta \log f(W) \rceil$, as $|T_i^*| \leq \beta$

In order to decode the encoded version of $\delta_x(T_i^*)$ in constant time, we construct a tabulated inverse function code^{-1}. From the input and output sizes, we see that we need a table with $2^{\lceil \beta \log(f(W)) \rceil}$ entries, for each of the β possible micro tree sizes, and each result entry having $\beta \lceil \log f(W) \rceil$ bits, giving a total space of $\beta 2^{\lceil \beta \log f(W) \rceil} \beta \lceil \log f(W) \rceil$ bits.

Let $T_i = T_{i(y)}$. Let & be the bitwise AND operator. In node y's label we save the bit string $\text{mask}(y)$ such that $\text{mask}(y)$ & $\delta_x(T_i^*)$ gives an integer sequence S identical to $\delta_x(T_i^*)$, except that the integer $\delta_x(v)$ has been replaced by 0 for all v that are not an ancestor of y. Given S we can now compute the micro sum $\delta_x(\text{MicroRoot}(y), y)$ as the sum of integers in the sequence S. We will create a tabulated function that sums these integers, SumIntegers. SumIntegers is given a sequence of up to β values in $[-W, W]$, and the output is a number in $[-\beta W, \beta W]$. We can thus tabulate SumIntegers as a table with $\beta 2^{\beta \lceil \log f(W) \rceil}$ entries each of size $\lceil \log f(\beta W) \rceil$, giving a total space of $\beta 2^{\beta \lceil \log f(W) \rceil} \lceil \log f(\beta W) \rceil$.

Both functions, code^{-1} and SumIntegers, have been tabulated in the above. A lookup in a tabulated function can be done in constant time on the RAM as long as both input and output can be represented by $O(\log n)$ bits. We can achieve this by setting

$$\beta \leq \frac{c \log n}{\lceil \log f(W) \rceil}$$

for a constant c. To see this, note that the maximum of the four input and output values above is $\lceil \log \beta \rceil + \beta \lceil \log f(W) \rceil$. Using the above inequality then gives $\log \log n + c \log n = O(\log n)$.

The tables for the tabulated functions are the same for all nodes. Hence, in principle, assuming an upper bound for n is known, we could encode the two tables in global memory, not using space in the labels. However, as we will see, the tables take no more space than the prefix table PreFix_x, so we can just as well encode them into the labels. Doing that we use an additional $\beta 2^{\lceil \beta \log(f(W)) \rceil} \beta \lceil \log f(W) \rceil$ for the code^{-1} table and $\beta 2^{\beta \lceil \log f(W) \rceil} \lceil \log f(W\beta) \rceil$ for the SumIntegers table. Using that $W = o(n)$ and substituting β for the above expression then gives, after a few reductions, that the extra space used is no more than $O((\log n)^4 n^c)$ bits. Since the prefix table uses at least $O(\frac{n \log \log n}{\log n})$ bits, we see that the added space does not (asymptotically) change the total space usage, as long as we choose $c < 1$.

Label summary: We will construct the labels such that either x's label contains $\delta_x(T_i^*(y))$ or vice versa (we shall see how in the next section). Using the tabulated function code^{-1}, the bits in $\delta_x(T_i^*(y))$

can be extracted in constant time from x's label. Using mask(y) from y's label and the tabulated function SumIntegers, we can then compute $\delta_x(\text{MicroRoot}(y), y)$ in constant time. The total space used for all this is no more than $O(\frac{n \log\log n}{\log n})$.

5.3 Storing and extracting the deltas

Let the micro trees in $\mathcal{T}$ bee given in a specific order: $T_1, \ldots, T_{|\mathcal{T}|}$. Let $D(x) = \text{code}(\delta_x(T_1^*)) \cdots \text{code}(\delta_x(T_{|\mathcal{T}|}^*))$ denote the binary string composed of the concatenation of each string $\text{code}(\delta_x(T_i^*))$ in the order $i = 1, 2, \ldots, |\mathcal{T}|$.

Let $L = |D(x)|$ be the length in bits of $D(x)$. Let $p_i \in [0, L)$ be the position in the string $D(x)$ where the substring $\text{code}(\delta_x(T_i^*))$ starts. E.g., $D(x)[0] = D(x)[p_1]$ is the first bit of $\text{code}(\delta_x(T_1^*))$, $D(x)[p_2]$ the first bit of $\text{code}(\delta_x(T_2^*))$, and so on. According to Lemma 2.3 we have $p_i = \sum_{j<i} \left\lceil |T_j^*| \log f(W) \right\rceil$. Observe that the position p_i only depends on i and W and not on x.

We denote by $a(y)$ and $a'(y)$ the starting and ending positions of the substring $\text{code}(\delta_x(T_{i(y)}^*))$ in $D(x)$. More precisely, $a(y) = p_{i(y)}$ and $a'(y) = p_{i(y)+1} - 1$, so that $|\text{code}(\delta_x(T_{i(y)}^*))| = a'(y) - a(y) + 1$. For each node y we use $O(\log n)$ bits to store $a(y)$ and $a'(y)$ in its label.

For a node x we will only save approximate half of $D(x)$, in a table $H(x)$. $H(x)$ will start with $\text{code}(\delta_x(T_{i(x)}^*))$ and the code for the following micro trees in the given circular order until $H(x)$ in total has at least $n/2$ δ_x values, but as few a possible. In other words $H(x) = \text{code}(\delta_x(T_{i(x)}^*)) \cdots \text{code}(\delta_x(T_{j(x)}^*))$ where the indexes $i(x), i(x)+1, \ldots, j(x)$ may wrap to 1 after reaching the largest index $|\mathcal{T}|$ if $j(x) < i(x)$. Let $b(x) = p_{j(x)+1}$.

In a node x's label we save $a(x), a'(x), b(x)$ and L using $O(\log n)$ bits. Having those values we know which δ_x values from $D(x)$ are saved in x's label as well as the position of them in $H(x)$. Furthermore we know the position of the δ_x-values of x's own micro tree in $D(x)$. We will need to extract at most $\lceil \beta \log f(W) \rceil = O(\log n)$ consecutive bits from $H(x)$ in one query. On the word-RAM this can be done in constant time.

Proposition 5.1. *Let x, y be two nodes of G. Then,*

(i) $|H(x)| = \frac{1}{2} n \log f(W) + O(\frac{n}{\log n} \log f(W))$*; and*
(ii) *either* $\text{code}(\delta_x(T_{i(y)}^*))$ *is part of* $H(x)$ *or* $\text{code}(\delta_y(T_{i(x)}^*))$ *is part of* $H(y)$.

Proof. Let $\mathcal{T}'$ be the subset of $\mathcal{T}$ encoded in $H(x)$. We have:

$$\begin{aligned}
|H(x)| &= \sum_{T_i \in \mathcal{T}'} \lceil |T_i^*| \log f(W) \rceil < \sum_{T_i \in \mathcal{T}'} (|T_i^*| \log f(W) + 1) \\
&< \frac{1}{2} n \log f(W) + |\mathcal{T}| + \lceil \beta \log f(W) \rceil < \frac{1}{2} n \log f(W) + O(n/\beta + \beta \log f(W)) \\
&< \frac{1}{2} n \log f(W) + O(\frac{n}{\log n} \log f(W))
\end{aligned}$$

which proves (i). Part (ii) follows from the fact that x saves at least half of the δ_x's in a cyclic order. If y not is include here, x must be included in the δ_y-values saved by y. □

5.4 Summary

The label of x is composed of the follows items.

1. The values $a(x)$, $a'(x)$, mask(x), $m(x)$, $\text{dist}_G(x, r)$, L and $b(x)$: $O(\log n)$.
2. A prefix table, PreFix_x, for the values in the macro tree: $O(\frac{n}{\log n}((\log f(W))^2 + \log\log n \log f(W)))$.

3. The table $H(x)$: $\frac{1}{2}n\log f(W) + O(\frac{n}{\log n}\log f(W))$.

4. Global tables, code^{-1} and SumIntegers of size $O(\frac{n\log\log n}{\log n})$.

Note that L and the global tables are common to all the nodes. In addition we may need to use $O(\log n)$ bits to save the start position in the label for the above constant number of sublabels.

Lemma 5.2. *Every label has length at most* $\frac{1}{2}n\log f(W) + O(\frac{n}{\log n}(\log^2 W + \log\log n \log f(W)))$ *bits.*

Let $\textsc{Decode}(\ell(x,G),\ell(y,G))$ denote the distance returned by the decoder given the labels of x and of y in G. It is defined by:

$\textsc{Decode}(\ell(x,G),\ell(y,G))$:

1. If $(a(x) \le a(y) < b(x)) \vee (b(x) < a(x) \le a(y)) \vee (a(y) < b(x) < a(x))$ then $s = a(y) - a(x) \pmod L$ and $e = a'(y) - a(x) \pmod L$
2. Else return $\textsc{Decode}(\ell(y,G),\ell(x,G))$
3. MacroSum $= \text{PreFix}_x(m(y))$
4. $S = \text{code}^{-1}(H_x[s,\ldots,e])$ & mask(y)
5. MicroSum $=$ SumIntegers(S)
6. Return $\text{dist}_G(x,r) + \text{MicroSum} + \text{MacroSum}$

Theorem 5.3. *There exists a distance labeling scheme for graphs with edge weights in* $[1,W]$ *using labels of length* $\frac{1}{2}n\log(2W+1) + O(\frac{n}{\log n}\log(2W+1)(\log W + \log\log n))$ *bits and constant decoding time.*

6 Approximate distances

By considering only a subset of nodes from G and using the previous techniques, it is possible to create an approximation scheme where the label size is determined by a smaller number of nodes but with larger weights between adjacent nodes. We leave the details for Appendix C.1 and present here only the result.

Theorem 6.1. *There exists a* $(2kW)$*-additive distance labeling scheme for graphs with* n *nodes and edge weights in* $[1,W]$ *using labels of size* $\frac{1}{2(k+1)}n\log(2(k+1)W+1) + O(\log n \cdot \log(nW))$.

Another way to achieve an approximation scheme is to use a smaller set of weights while keeping the accumulated error under control. This leads to the following result whose proof can be seen in Appendix C.2.

Theorem 6.2. *For any* $D \le 2W-1$ *there exists a* $\left\lceil \frac{D}{2W-D} \right\rceil$*-additive distance labeling scheme for graphs with* n *nodes and edge weights in* $[1,W]$ *using labels of size* $\frac{1}{2}n\log(2W+1-D) + O(\log n \cdot \log(nW))$.

One instance of Theorem 6.2 is $D = W$, which gives a 1-additive distance labeling scheme of size $\frac{1}{2}n\log(W+1) + o(n)$. For $D = 2W-1$ we get a $(2W-1)$-additive distance labeling scheme of size $\frac{1}{2}n + o(n)$. For constant r the above technique also applies to our constant time decoding results. For unweighted graphs this implies that we can have labels of size $\frac{1}{2}n + o(n)$ with a 1-additive error and constant decoding time.

By combining the above two theorems, we obtain the theorem below; see Appendix C.3.

Theorem 6.3. *For any* $k \ge 0$ *and* $D \le 2(k+1)W - 1$ *there exists a* $(2kW + \left\lceil \frac{D}{2(k+1)W-D} \right\rceil)$*-additive distance labeling scheme for graphs with* n *nodes and edge weights in* $[1,W]$ *using labels of size* $\frac{1}{2(k+1)}n\log(2(k+1)W+1-D) + O(\log n \cdot \log(nW))$ *bits.*

References

[1] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, and T. Rauhe. Compact labeling scheme for ancestor queries. *SIAM J. Comput.*, 35(6):1295–1309, 2006.

[2] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. of the 12th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 547–556, 2001.

[3] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *ACM International Conference on Management of Data (SIGMOD)*, pages 349–360, 2013.

[4] S. Alstrup, P. Bille, and T. Rauhe. Labeling schemes for small distances in trees. In *Proc. of the 14th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 689–698, 2003.

[5] S. Alstrup, P. Bille, and T. Rauhe. Labeling schemes for small distances in trees. *SIAM J. Discrete Math.*, 19(2):448–462, 2005. See also SODA'03.

[6] S. Alstrup, E. B. Halvorsen, and K. G. Larsen. Near-optimal labeling schemes for nearest common ancestors. In *Proc. of the 25th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 972–982, 2014.

[7] S. Alstrup, H. Kaplan, M. Thorup, and U. Zwick. Adjacency labeling schemes and induced-universal graphs. In *Proc. of the 47th Annual ACM Symp. on Theory of Computing (STOC)*, 2015. To appear.

[8] S. Alstrup and T. Rauhe. Improved labeling schemes for ancestor queries. In *Proc. of the 13th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2002.

[9] S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In *In Proc. 43rd annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 53–62, 2002.

[10] F. Bazzaro and C. Gavoille. Localized and compact data-structure for comparability graphs. *Discrete Mathematics*, 309(11):3465–3484, June 2009.

[11] B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. *SIAM J. Discrete Math.*, 19(4):1029–1055, 2005. See also SODA'03.

[12] N. Bonichon, C. Gavoille, and A. Labourel. Short labels by traversal and jumping. In *Structural Information and Communication Complexity*, pages 143–156. Springer, 2006.

[13] M. A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT–12:148–153, 1966.

[14] M. A. Breuer and J. Folkman. An unexpected result on coding vertices of a graph. *J. of Mathemathical analysis and applications*, 20:583–600, 1967.

[15] G. Chartrand, T. Thomas, P. Zhang, and Varaporn Saenpholphat. A new look at Hamiltonian walks. *Bull. Inst. Combin. Appl.*, 42:37–52, 2004.

[16] J. Cheng, S. Huang, H. Wu, and A. Wai-Chee Fu. TF-label: a topological-folding labeling scheme for reachability querying in a large graph. In *ACM International Conference on Management of Data (SIGMOD)*, pages 193–204, 2013.

[17] V. D. Chepoi, F. F. Dragan, B. Estellon, M. Habib, and Y. Vaxès. Diameters, centers, and approximating trees of delta-hyperbolic geodesic spaces and graphs. In *24st Annual ACM Symp. on Computational Geometry*, pages 59–68, 2008.

[18] V. D. Chepoi, F. F. Dragan, and Y. Vaxès. Distance and routing labeling schemes for non-positively curved plane graphs. *J. of Algorithms*, 61(2):60–88, 2006.

[19] F. R. K. Chung. Universal graphs and induced-universal graphs. *J. of Graph Theory*, 14(4):443–454, 1990.

[20] E. Cohen, H. Kaplan, and T. Milo. Labeling dynamic XML trees. *SIAM J. Comput.*, 39(5):2048–2074, February 2010.

[21] B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131:129–150, 2003.

[22] L. J. Cowen. Compact routing with minimum stretch. *J. of Algorithms*, 38:170–183, 2001.

[23] D. Delling, A. V. Goldberg, R. Savchenko, and R. Foncesa Werneck. Hub labels: Theory and practice. In *13th International Symp. on Experimental Algorithms*, pages 259–270, 2014.

[24] Y. Dodis, M. Pătraşcu, and M. Thorup. Changing base without losing space. In *Proc. of the 42nd Annual ACM Symp. on Theory of Computing (STOC)*, pages 593–602, 2010.

[25] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. *J. of Algorithms*, 46(2):97–114, 2003.

[26] A. Farzan and J. I. Munro. Succinct encoding of arbitrary graphs. *Theoretical Computer Science*, 513:38–52, 2013.

[27] A. Farzan and J. I. Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014.

[28] P. Ferraginaud, I. Nitto, and R. Venturini. On compact representations of all-pairs-shortest-path-distance matrices. *Theor. Comput. Sci.*, 411(34-36):3293–3300, July 2010.

[29] P. Fraigniaud and A. Korman. On randomized representations of graphs using short labels. In *Proc. of the 21st Annual Symp. on Parallelism in Algorithms and Architectures*, pages 131–137, 2009.

[30] P. Fraigniaud and A. Korman. Compact ancestry labeling schemes for XML trees. In *Proc. of the 21st annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 458–466, 2010.

[31] P. Fraigniaud and A. Korman. An optimal ancestry scheme and small universal posets. In *Proc. of the 42nd ACM Symp. on Theory of computing (STOC)*, pages 611–620, 2010.

[32] C. Gavoille, M. Katz, N. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *Proc. of the 9th annual European Symp. on Algorithms*, pages 476–488, 2001.

[33] C. Gavoille and O. Ly. Distance labeling in hyperbolic graphs. In *16th Annual International Symp. on Algorithms and Computation*, pages 1071–1079, 2005.

[34] C. Gavoille and C. Paul. Distance labeling scheme and split decomposition. *Discrete Mathematics*, 273(1-3):115–130, 2003.

[35] C. Gavoille and C. Paul. Optimal distance labeling for interval graphs and related graphs families. *SIAM J. on Discrete Mathematics*, 22(3):1239–1258, July 2008.

[36] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *J. of Algorithms*, 53(1):85 – 112, 2004. See also SODA'01.

[37] S. Goodman and S. Hedetniemi. On the hamiltonian completion problem. In *Proc. 1973 Capital Conf. on Graph Theory and Combinatorics*, pages 263–272, 1974.

[38] R. L. Graham and H. O. Pollak. On embedding graphs in squashed cubes. In *Lecture Notes in Mathematics*, volume 303 of *Proc. of a conference held at Western Michigan University*. Springer-Verlag, 1972.

[39] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symp. on Foundations of Computer Science (FOCS)*, pages 534–543, 2003.

[40] A. Gupta, A. Kumar, and R. Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM J. on Computing*, 34(2):453–474, 2005. See also FOCS'01.

[41] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 26(2):147–160, 1950.

[42] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *ACM International Conference on Management of Data (SIGMOD)*, pages 445–456, May 2012.

[43] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. Disc. Math.*, pages 596–603, 1992. See also STOC'88.

[44] H. Kaplan, T. Milo, and R. Shabo. A comparison of labeling schemes for ancestor queries. In *Proc. of the 13th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2002.

[45] M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004. See also SODA'02.

[46] A. Korman. Labeling schemes for vertex connectivity. *ACM Trans. Algorithms*, 6(2):39:1–39:10, April 2010.

[47] A. Korman and D. Peleg. Labeling schemes for weighted dynamic trees. *Inf. Comput.*, 205(12):1721–1740, 2007.

[48] R. Krauthgamer and J. R. Lee. Algorithms on negatively curved spaces. In *47th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 119–132, 2006.

[49] V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theor. Comput. Sci.*, 387(3):332–347, November 2007.

[50] J. W. Moon. On minimal n-universal graphs. *Proc. of the Glasgow Mathematical Association*, 7(1):32–33, 1965.

[51] J. H. Müller. *Local structure in graph classes*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 1988. Order No: GAX88-11342.

[52] J. I. Munro, R. Raman, V. Raman, and S. Srinivasa Rao. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, 2012.

[53] M. Pătraşcu. Succincter. In *Proc. 49th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 305–313, 2008.

[54] D. Peleg. Informative labeling schemes for graphs. In *In Proc. 25th Symp. on Mathematical Foundations of Computer Science*, pages 579–588. Springer-Verlag, 2000.

[55] D. Peleg. Proximity-preserving labeling schemes. *J. Graph Theory*, 33(3):167–176, March 2000.

[56] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The computer J.*, 28:5–8, 1985.

[57] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. of Computer and System Sciences*, 26(3):362 – 391, 1983.

[58] J. P. Spinrad. *Efficient Graph Representations*, volume 19 of *Fields Institute Monographs*. AMS, 2003.

[59] K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proc. of the 36th Annual ACM Symp. on Theory of Computing (STOC)*, pages 281–290, 2004.

[60] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, November 2004. See also FOCS'01.

[61] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. of the 13th Annual ACM Symp. on Parallel Algorithms and Architectures*, SPAA '01, pages 1–10. ACM, 2001.

[62] M. Thorup and U. Zwick. Approximate distance oracles. *J. of the ACM*, 52(1):1–24, 2005. See also STOC'01.

[63] O. Weimann and D. Peleg. A note on exact distance labeling. *Inf. Process. Lett.*, 111(14):671–673, 2011.

[64] R. Wenger. Extremal graphs with no C^4's, C^6's, or C^{10}'s. *J. of Combinatorial Theory, Series B*, 52(1):113–116, 1991.

[65] P. M. Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983.

A Lower bounds

Our lower bound technique can be seen as a generalization of the classical counting argument for adjacency labeling schemes. Indeed, for $r = 0$ and $W = 1$, our formula yields $(n-1)/2$ bits, which is exactly the number of bits needed for adjacency. The lower bound we develop here is well-suited for small additive errors r. In particular, when $r < 2W$ we prove that labels of $\Omega(n \log(W/(r+1)))$ bits are required for an r-additive distance labeling scheme.

Given an unweighted graph B and an integer $W \geq 1$, denote by $\mathcal{F}_W(B)$ be the family of all subgraphs of B whose edges are weighted by values taken from $[1, W]$.

Theorem A.1. *Let B be an unweighted graph with n vertices, m edges and girth at least g, and let r, W be integers such that $r \in [0, (g-2)W)$. Then, every r-additive approximate distance labeling scheme for $\mathcal{F}_W(B)$ requires a total label length of at least $m \log(k+1)$, and thus a label of at least $(m/n)\log(k+1)$ bits, where*

$$k \;=\; \left\lfloor \frac{g-2}{g-1} \cdot \left(\frac{W}{r+1} + 1 \right) \right\rfloor \;.$$

Proof. An r-approximate distance matrix for a weighted graph G with vertex-set $[1, n]$ is an $n \times n$ matrix M such that $\text{dist}_G(x, y) \leq M[x, y] \leq \text{dist}_G(x, y) + r$ for all vertices x, y of G.

The basic idea of our lower bound technique is to show that $\mathcal{F}_W(B)$ contains a large set $\mathcal{G}$ of weighted graphs for which no two graphs can have the same r-approximate distance matrix. A crude observation is that an r-approximate distance matrix for each graph of $\mathcal{G}$ can be generated from the ordered list of all the labels provided by any r-additive approximate labeling scheme for $\mathcal{G}$. So, it turns out that, by a simple counting argument, the total label length must be at least $\log|\mathcal{G}|$. In particular, the labeling scheme must assign, for some vertex of some graph of $\mathcal{G}$, a label of at least $(\log|\mathcal{G}|)/n$ bits. We now construct such a set $\mathcal{G}$ with $|\mathcal{G}| = (k+1)^m$.

For the shake of the presentation, define $W_i = W - (k-i-1)(r+1)$ for $i = 0, \ldots, k$. Note that the W_is increase with i, and more precisely that $W_{i+1} = W_i + r + 1$. Moreover, we observe that:

Claim A.2. *The following hold: $k \geq 1$, $W_0, \ldots, W_{k-1} \in [1, W]$, and $W_k \leq (g-1)W_0$.*

Before we give a formal proof of Claim A.2 (which is a basic calculation), we explain how to derive our lower bound.

Consider the set $\mathcal{C}$ of all edge-colorings of B into $k+1$ colors. More precisely, an edge-coloring $c \in \mathcal{C}$ is simply a function $c : E(B) \to [0, k]$ mapping to each edge e of B some integer $c(e) \in [0, k]$. Clearly, $|\mathcal{C}| = (k+1)^m$ since each of the m edges of B can receive $k+1$ distinct values.

With each coloring $c \in \mathcal{C}$, we associate a weighted graph G with edge-weight function w obtained from graph B by testing the color of each edge xy of B. If $c(xy) = k$, the edge is deleted. And, if $c(xy) = i < k$, we keep xy in the graph and set $w(xy) = W_i$. We denote by $\mathcal{G}$ the family of graphs constructed by this process from all the colorings of $\mathcal{C}$. It is clear that, given B, one can recover from G and w the initial coloring c (just scan all the possible edges of B, check if they exist in G and look at their weights). In other words the construction is bijective and thus $|\mathcal{G}| = |\mathcal{C}| = (k+1)^m$.

By construction, each graph of $\mathcal{G}$ is a subgraph of B. Moreover, by Claim A.2, each weight is some integer $W_i \in [1, W]$ as $i \in [0, k-1]$ (edges of color k have been removed). In other words, $\mathcal{G} \subseteq \mathcal{F}_W(B)$. It remains to prove that any two graphs of $\mathcal{G}$ cannot have the same r-approximate distance matrix. The intuition is that two graphs of $\mathcal{G}$ differ only when there is an edge xy in B whose color is different in the two graphs. Because of the choice of the edge-weights, the distance between x and y in the two graphs must, as we shall see, differ by at least $r+1$.

Let G, G' be two distinct weighted graphs of $\mathcal{G}$. Denote by w, w' their respective edge-weighting functions, and by M, M' any r-approximate distance matrices for G and G' respectively. As we will show, if G and G' are different, there must exist an edge xy of B, and two colors $i, j \in [0, k]$, $i < j$,

such that $\text{dist}_G(x,y) \leq W_i$ and $\text{dist}_{G'}(x,y) \geq W_j$ (the case $\text{dist}_{G'}(x,y) \leq W_i$ and $\text{dist}_G(x,y) \geq W_j$ is symmetric). For this purpose, we consider two cases:

(i) The graphs G, G' are different because there is an edge xy of B which is in G but not in G'. We have $xy \in E(G)$, which implies that $\text{dist}_G(x,y) \leq w(xy) \leq W_{k-1}$, since the color of xy in G is $< k$. Further, $xy \notin E(G')$ implies that $\text{dist}_{G'}(x,y) \geq (g-1)W_0$, since any path from x to y in G' contains at least $g-1$ edges (G' is a subgraph of B which has girth at least g), and the minimum weight assigned to any edge is W_0. (Note that this holds, in particular, when x and y are unconnected and $\text{dist}_G(x,y) = \infty$.) Thus from Claim A.2, $\text{dist}_{G'}(x,y) \geq W_k$. So the claim holds for $i = k-1$ and $j = k$.

(ii) The graphs G, G' are different because there is an edge xy in G and G' with different weights. Assuming that $w(xy) < w'(xy)$, there must exist i, j such that $w(xy) = W_i$ and $w'(xy) = W_j$. Note that $i < j < k$. We have $\text{dist}_G(x,y) = W_i$ and $\text{dist}_{G'}(x,y) = W_j$ since we have seen in the previous case that every path from x to y and excluding the edge xy has cost at least $(g-1)W_0$. By Claim A.2, $W_i < W_j < W_k \leq (g-1)W_0$.

In both cases we have found $i, j \in [0,k]$, $i < j$, such that $\text{dist}_G(x,y) \leq W_i$ and $\text{dist}_{G'}(x,y) \geq W_j$. Now, by definition of M and M', $M[x,y] \leq \text{dist}_G(x,y) + r \leq W_i + r$ and $W_j \leq \text{dist}_{G'}(x,y) \leq M'[x,y]$. Since $W_j \geq W_{i+1} = W_i + r + 1$, we conclude that $M[x,y] < M'[x,y]$ proving that no two different graphs of $\mathcal{G}$ can have the same r-approximate distance matrix.

To complete the proof, it remains to prove Claim A.2. Let us first show that $k \geq 1$ (which is required since in the proof we use for instance that $W_0 \leq W_{k-1}$). Recall that $r \leq (g-2)W - 1$,

$$k \;=\; \left\lfloor \frac{g-2}{g-1}\left(\frac{W}{r+1}+1\right)\right\rfloor \;\geq\; \left\lfloor \frac{g-2}{g-1}\left(\frac{W}{(g-2)W}+1\right)\right\rfloor \;=\; \left\lfloor \frac{g-2}{g-1}\left(\frac{1}{g-2}+1\right)\right\rfloor \;=\; 1.$$

Let us show that $W_0 \geq 1$. Since $W_0 = W - (k-1)(r+1)$, it suffices to check that $(k-1)(r+1) < W$. We have,

$$(k-1)(r+1) \;=\; \left(\left\lfloor \frac{g-2}{g-1}\left(\frac{W}{r+1}+1\right)\right\rfloor - 1\right)\cdot(r+1) \;\leq\; \frac{g-2}{g-1}\cdot\frac{W}{r+1}\cdot(r+1) \;<\; W$$

since the girth g is always at least three.

Now we have $W_0, \ldots, W_{k-1} \in [1, W]$, since the W_i's are non-decreasing, $W_0 \geq 1$, and $W_{k-1} = W - (k-(k-1)-1)(r+1) = W$.

Let us show that $W_k \leq (g-1)W_0$. We have $W_k = W + r + 1$ and $W_0 = W - (k-1)(r+1)$. Therefore,

$$\begin{aligned}
W_k \leq (g-1)W_0 \;&\Leftrightarrow\; W + r + 1 \leq (g-1)(W - (k-1)(r+1)) \\
&\Leftrightarrow\; r + 1 + (g-1)(k-1)(r+1) \leq (g-2)W \\
&\Leftrightarrow\; (g-1)(k-1) \leq (g-2)\frac{W}{r+1} - 1 \\
&\Leftrightarrow\; k \leq \frac{g-2}{g-1}\cdot\frac{W}{r+1} - \frac{1}{g-1} + 1 \;=\; \frac{g-2}{g-1}\cdot\left(\frac{W}{r+1}+1\right).
\end{aligned}$$

The latter equation is true by the choice of k. This completes the proof of Claim A.2 and of Theorem A.1. □

A collection of corollaries to Theorem A.1 can be seen in Table 4.

The case $r \geq 2$ and $W = 1$ is out of the range of our lower bound, as long as we choose for B a graph with $m = \Theta(n^2)$ edges. Our lower bound still applies for $r = 2, 3$ and $W = 1$, but using girth-6 graphs B that are known to exists with $m = \Theta(n^{3/2})$ edges. There are several constructions, based on finite projective geometries, of graphs with $\Omega(n^{3/2})$ edges and girth at least 6 (see for instance [64]). So, Theorem A.1 can also prove the $\Omega(\sqrt{n}\,)$ lower bound for $r = 2, 3$ and $W = 1$. The case of larger r can be captured by the more general lower bound of [32], that uses a subdivision technique, and shows that $\Omega(\sqrt{n/(r+1)}\,)$ bit labels are required for any $r \geq 2$.

Graphs	$r = 0, W \geq 1$	$r = 1, W \geq 2$	$r = 0, W = 1$	$r = (g-2)W - 1$
General	$\frac{1}{2}(n-1)\log\lceil\frac{W}{2}+1\rceil$	$\frac{1}{2}(n-1)\log\lfloor\frac{W}{4}+\frac{3}{2}\rfloor$	$\frac{1}{2}(n-1)$	
Bipartite	$\frac{1}{4}n\log\lfloor\frac{2W}{3}+\frac{5}{3}\rfloor$	$\frac{1}{4}n\log\lfloor\frac{W}{3}+\frac{5}{3}\rfloor$	$\frac{1}{4}n$	

Table 4: Lower bounds derived from Theorem A.1. For "general graphs" we use the family $\mathcal{F}_W(K_n)$, where K_n denotes the complete graph on n vertices, so $m = n(n-1)/2$ and $g = 3$. For "Bipartite graphs" we use the family $\mathcal{F}_W(K_{n/2,n/2})$, where $K_{n/2,n/2}$ denotes the complete bipartite graph on n vertices (assuming n even) so $m = n^2/4$ and $g = 4$. Note that the case $r = W = 1$ and the case $r = 2W - 1$ is captured by the last column of the last line, and so the lower bound is $n/4$.

B Constructing micro trees

Lemma B.1. *Let k be a positive integer. Every m-edge tree has an edge partition into at most $\lceil m/k \rceil$ trees of at most $2k$ edges.*

Proof. Consider a tree T with m edges. If T has fewer than k edges, then the partition is T itself and we are done.

Otherwise, we will construct a subtree A of T with at least k and at most $2k$ edges such that $T - A$ is still connected. (By $T - A$ we mean the forest induced by all the edges in $E(T) - E(A)$.) Once such an A is constructed, we can repeat the process on the remaining tree $T - A$ until having a tree with less than k edges. Since each subtree A has at least k edges, the process stop after we have constructed at most $\lceil m/k \rceil$ trees. □

C Approximate distances

C.1 Approximation using fewer nodes

Lemma C.1. *Given a graph G with n nodes, edge weights in $[1, W]$ and a rooted spanning tree T, we can, for integers $k \geq 0$, construct a tree $T(k)$ whose node set is a subset of T and with the following properties.*

- $|T(k)| \leq 1 + \frac{n}{k+1}$.
- *For any node $v \in T(k)^*$, $\text{dist}_G(v, parent_{T(k)}(v)) \leq (k+1)W$.*
- *For any node $w \in G$, there exists a node $v \in T(k)$ with $\text{dist}_G(v, w) \leq kW$.*

Proof. Partition the nodes in T into $k+1$ equivalence classes according to their depth in T modulo $k+1$. One of these equivalence classes must contain $\lfloor n/(k+1) \rfloor$ or fewer nodes. Select such a subset of nodes and denote it $T(k)$. Also include the root of T in $T(k)$, giving that $|T(k)| \leq 1 + n/(k+1)$. In $T(k)$ construct an edge between two nodes iff no other nodes from $T(k)$ are on the simple path between the nodes in T. Then, for any $v \in T(k)^*$, $\text{dist}_G(v, \text{parent}_{T(k)}(v)) \leq (k+1)W$ since the number of edges between v and $\text{parent}_{T(k)}(v)$ in T is at most $k+1$. Similarly, for any $w \in T$, its nearest ancestor v, which also in $T(k)$ (could be w itself) is at most k edges up in T, giving $\text{dist}_G(v, w) \leq kW$ □

Roughly speaking, the approximation scheme presented here applies the previous techniques to create an exact distance labeling scheme for $T(k)$, which has fewer nodes but larger weights between adjacent nodes. For a node x that is not in $T(k)$, we find its nearest ancestor x' in $T(k)$ and give x the same label as x'. We then approximate $\text{dist}_G(x, y)$ by $2kW + \text{dist}_G(x', y')$. This will at most give us an error in $[0, 4kW]$, meaning that we now have a $(4kW)$-additive labeling scheme with labels of size

$\frac{n}{2(k+1)}\log(2(k+1)W+1)$, ignoring second order terms. It is possible to optimize this approach and obtain a $(2kW)$-additive scheme, by only using approximate distance for either x or y to their nearest ancestors x' or y'. Here, we show how to do it for the heavy path approach. A similar result holds for the micro tree approach.

As described in Section 4, the label $\ell(x)$ includes the sublabels $\ell_T(x)$ and $\ell'_T(x)$ using $O(\log n \cdot \log(nW))$ bits. Our new label for approximate distance will include those sublabels as well. For a node v in T let v' be its nearest ancestor in $T(k)$. In x's label we also save $\ell_T(x')$ and $\ell'_T(x')$. In addition, we include a label $\ell''_T(x)$ containing the distances from x to w' for all w that appear in $\ell(x)$; this label also uses $O(\log n \cdot \log(nW))$ bits.

Now. if x or x' is an ancestor to y in T, we compute $\text{dist}_G(x,y)$ as kW plus the distance in T from x or x' to y. This will at most give an additive error of $2kW$. Similarly for y and y' and x. Those computation can be done as explained in Section 4 using the labels defined so far. If we cannot compute the distance in this way, then, consider the nearest common ancestor z of x and y in T. The node z' must then be the nearest common ancestor of x' and y' in $T(k)$. Let $n_k = |T(k)|$. We will save $\lfloor n_k/2 \rfloor$ of the values $\delta_x(\text{parent}_{T(k)}(v), v)$ for all v from $T(k)$ with $\text{dfs}(x') < \text{dfs}(v) \leq \text{dfs}(x') + \lfloor n_k/2 \rfloor \pmod{n_k}$. As $\delta_x(\text{parent}_{T(k)}(v), v) \in [-(k+1)W, (k+1)W]$ we can encode all these δ-values using $\lceil \frac{1}{2} n_k \log f((k+1)W) \rceil$ bits. We can now compute $\text{dist}_G(x,y) = kW + \text{dist}_G(x,z') + \sum_{v \in T(k)(z',y']} \delta_x(\text{parent}_{T(k)}(v), v)$, where $\text{dist}_G(x,z')$ can be computed from $\ell''_T(x)$. This will at most give an additive error of $2kW$. We have now proved the following theorem.

Theorem C.2. *There exists a $(2kW)$-additive distance labeling scheme for graphs with n nodes and edge weights in $[1,W]$ with labels of size $\frac{1}{2(k+1)} n \log f((k+1)W) + O(\log n \cdot \log(nW))$.*

C.2 Approximation using fewer weights

With edge weights in $[1,W]$ we have been encoding $\frac{1}{2}n$ numbers from the interval $I = [-W, W]$ of size $|I| = 2W+1$ using $\frac{1}{2} n \log(2W+1)$ bits. The theorem below uses an approximation technique where we use integers from a smaller set $I' \subseteq I$, which will reduce the space consumption but introduce an error when computing δ_x-values. As we shall see, the error can be capped even when we are summing many δ_x-values.

Theorem C.3. *For any $D \leq 2W-1$ there exists a $\left\lceil \frac{D}{2W-D} \right\rceil$-additive distance labeling scheme for graphs with edge weights in $[1,W]$ using labels of size $\frac{1}{2} n \log(2W+1-D) + O(\log n \cdot \log(nW))$ bits.*

Proof. Let us create a subset $I' \subseteq I$ with $|I'| = |I| - D$. In I' we always include the maximum and minimum from I, and hence we require $D \leq |I| - 2 = 2W-1$. In addition we minimize the maximum number Q of consecutive numbers from $I - I'$. Hence, for $i_1 \in I'$ (excluding maximum) there exists a number $i_2 \in I'$ such that $i_2 \leq i_1 + Q + 1$. Since $|I'| = |I| - D = 2W + 1 - D$, we have $2W - D$ pair of neighbors in I', where we by "neighbors" mean two numbers in I' with no other number from I' between them. By equally spreading the D missing numbers between the $2W - D$ pairs, we can obtain $Q = \left\lceil \frac{D}{2W-D} \right\rceil$. By substituting values in I' for values in I, we can now encode t values from I with $\lceil t \log(2W+1-D) \rceil$ bits. This will introduce an error, but in the case of δ_x-values, the accumulated error can be kept below Q as described below.

Let T be a tree and consider the δ_x-values $\delta_x(v) = \delta_x(\text{parent}(v), v)$ for nodes $v \in T^*$. Each δ_x-value is a number in $I = [-W, W]$. We will visit the nodes top down starting from (but not including) the root r and assigning to each node y a new approximate value: $\tilde{\delta}_x(y) \in I'$. (For the root r we implicitly associate the value 0. Implicitly, since it may not be a value in I'.) Recall that $\delta_x(r,y) = \sum_{v \in T[y,r)} \delta_x(v)$, and define $\tilde{\delta}_x(r,y) = \sum_{v \in T[y,r)} \tilde{\delta}_x(v)$. We will assign the values such that $\delta_x(r,y) \leq \tilde{\delta}_x(r,y) \leq \delta_x(r,y) + Q$.

For $y \in T^*$, let $A(y) = \tilde{\delta}_x(r,y) - \delta_x(r,y)$. We prove by induction that $A(y) \in [0,Q]$. So assume inductively that $A(\text{parent}(y)) \in [0,Q]$. If $\delta_x(y) \in I'$, we can define $\tilde{\delta}_x(y) = \delta_x(y)$, and we then have

$A(y) = A(\text{parent}(y)) \in [0, Q]$ as desired. If this is not the case, let $i_1, i_2 \in I'$ be the largest and smallest numbers from I', respectively, with $i_1 < \delta_x(y) < i_2$. By assumption, $i_2 - i_1 \leq Q + 1$. If $A(\text{parent}(y)) + i_1 - \delta_x(y) \geq 0$, we can set $\tilde{\delta}_x(y) = i_1$ and obtain $A(y) \in [0, Q]$. If not, then we must have $A(\text{parent}(y)) + i_2 - \delta_x(y) < i_2 - i_1 \leq Q + 1$, so we can set $\tilde{\delta}_x(y) = i_2$ and obtain $A(y) \in [0, Q]$. This concludes the theorem.

Above we have been changing all δ_x-values top-down from the root. In the constant time solution, we could instead change the values top-down for each micro tree T_i, keeping exact distances to the root and in the macro tree. □

C.3 Final approximation

We can combine the above two approaches by, for a $k \geq 0$, first using Appendix C.1 to obtain a $(2kW)$-additive distance labeling scheme with labels of size $\frac{n}{2(k+1)} \log f((k+1)W) + O(\log n \cdot \log(nW))$. The approximate scheme will use edge weights in $[1, (k+1)W]$ to which then can apply the technique from Appendix C.2, finally getting:

Theorem C.4. *For any $k \geq 0$ and $D \leq 2(k+1)W - 1$ there exists a $(2kW + \left\lceil \frac{D}{2(k+1)W - D} \right\rceil)$-additive distance labeling scheme for graphs with n nodes and edge weights in $[1, W]$ using labels of size $\frac{n}{2(k+1)} \log(2(k+1)W + 1 - D) + O(\log n \cdot \log(nW))$ bits.*