Egerváry Research Group
on Combinatorial Optimization

Technical reportS

TR-2015-02. Published by the Egerváry Research Group, Pázmány P. sétány 1/C, H–1117, Budapest, Hungary. Web site: www.cs.elte.hu/egres. ISSN 1587–4451.

A succinct tree coding for greedy navigation

Zoltán Király and Sándor Kisfaludi-Bak

February 10, 2015

A succinct tree coding for greedy navigation

Zoltán Király * and Sándor Kisfaludi-Bak **

Abstract

We present a succinct tree coding that enables *greedy routing* in arbitrary connected graphs. The worst-case length of vertex addresses is at most $3 \log n + \log \log n + 3.59$ bits for n vertex graphs. We define a distance function with which greedy message forwarding is guaranteed to deliver messages between any pair of vertices in the graph.

1 Introduction

The need to assign codes to the vertices of a tree emerges naturally in many applications. One of these applications is the navigation or routing problem of graphs. The task is to deliver messages between any pair of vertices, using only local information and the label of the recipient. The most simple such algorithm is the greedy routing algorithm, where the labels correspond to points in some space which is equipped with a distance function f, and each vertex forwards the message to the neighbour that is the closest to the recipient according to f.

Finding a space in which to embed our graph such that the greedy routing algorithm is guaranteed to work is a non-trivial task: if the vertices are not placed properly in the target space, messages may get stranded if there is no neighbour of the current vertex v that is closer to the recipient than the v itself.

Several embedding methods were created ([5], [7]) that solved the issue of stranded messages. The downside of these methods is that they use $\Theta(n \log n)$ bits to encode

*Department of Computer Science and EGRES (MTA-ELTE), Eötvös University, Pázmány Péter sétány 1/C, Budapest, Hungary. Research was supported by grants (no. CNK 77780 and no. K 109240) from the National Development Agency of Hungary, based on a source from the Research and Technology Innovation Fund.

**Department of Computer Science, Eötvös University, Pázmány Péter sétány 1/C, Budapest, Hungary.

labels, which is no better than prescribing a forwarding place for each possible recipient. Reducing the label length is a main concern for applications as well. After this point, the papers in the literature are usually either concerned with the address lengths of the vertices or the efficiency of routing.

The paper of Eppstein and Goodrich [2] was one of the first attempts to give a *succinct* greedy embedding, i.e., an embedding requiring only $O(\log n)$ bits for each vertex, using abstract coordinates, but these coordinates could still be assigned to points on a hyperbolic plane. The shortcoming of this paper is that the messages get stranded (due to a flaw in the method).

Greedy embeddings do not need to rely on metric spaces. The paper of Zhang and Govindaiah [8] uses a semi-metric embedding (a metric space without triangle inequality), and guarantee $O(\log n)$ length address for 3-connected planar graphs, or any graph that has a spanning tree with bounded degree. Note that the final code length is more than $\Delta \log n$ because some separators need to be encoded. (Here Δ denotes the maximum degree of the chosen spanning tree.) When trying to use this embedding it is unclear whether there is a spanning tree with low maximal degree in an arbitrary connected graph, so this method still falls short of achieving the $O(\log n)$ address length in the most general sense.

Independently of our research, a paper by Althofer et al. [1] proves that there is a nearest common ancestor labelling for any tree with n vertices. It is straightforward to prove that this labelling can be used for greedy routing in our sense. They construct a method for representing the addresses using $3\lfloor \log n \rfloor$ bits. (Note that they also show a labelling with shorter addresses, but the coding and decoding process has no known polynomial algorithm.) While this bound is superior to ours in terms of maximum address length (we have a $\log \log n + 3$ additive term after $3 \log n$), this embedding uses close to $3\lfloor \log n \rfloor$ bits in practice, as it is demonstrated empirically in [4].

Our method is inspired by the embedding given by Eppstein and Goodrich [2], but our 'distance' function will not be a semi-metric: it will not be a symmetric function. It can be argued that the this is not a distance function, nevertheless we use this term because it describes the usage of the function. The main focus of the construction is to reduce code length as much as possible.

Our method is applicable for any connected simple graph G. It begins by choosing a spanning tree T of G. This tree is embedded into a binary tree B. The construction of B requires a subtree building procedures. We provide two sets of subtree building

procedures. The first set of subtree building procedures will provide a minimal code length (given our definitions), and the second set will be used for bounding the code length.

The paper is organized as follows. Section 2 introduces our basic setting, some notations, and states our main theorem (Theorem 2.1). Section 3 describes the embedding method, with the exception of the subtree building procedures for which some properties are prescribed. In Section 4, we offer the optimal subtree building procedures that fit the prescriptions of Section 3. Section 5 defines the distance function and proves that message delivery is guaranteed. Section 6 presents the alternative subtree building procedures. Using this second procedure we can prove our main theorem in Section 7. We also provide a construction that shows the sharpness of our main theorem. Finally, we summarize and pose some questions in Section 8.

2 Preliminaries

Let G be an arbitrary simple and connected graph. We denote by $N_G(v)$ the set of neighbours of $v \in V(G)$. Let f be an *embedding* into a set S, i.e., an injective function $f : V(G) \to S$, and let *dist* be a *distance* function: $dist : S \times S \to \mathbb{R}_{\geq 0}$. (Note that it is not required for *dist* to be a metric on S.) Using f, we can define the distance from u to v as $d(u, v) = dist(f(u), f(v))$.

Messages that are sent over the network have a specific sender $s \in V(G)$ and target $t \in V(G)$, and the message header contains $f(t) \in S$, that is, the *address* of t. The *greedy routing scheme* sends the message from v to the vertex $w \in N(v)$ for which $d(w, t)$ is minimal. (If v has multiple neighbours at the same distance to t, we choose a vertex among these arbitrarily.) The pair $(f, dist)$ defines a *guaranteed greedy embedding* if the greedy routing scheme can deliver any message.

We are now ready to state our main theorem.

Theorem 2.1. *For a connected graph G there is an assignment $f : V(G) \to \{0, 1\}^t = S$ and a function $d : S \times S \to \mathbb{R}_{\geq 0}$ such that (f, d) is a guaranteed greedy embedding with code length $t \leq \lfloor 3 \log n \rfloor + \lceil \log \lfloor 3 \log n \rfloor \rceil + 1 \leq 3 \log n + \log \log n + 3.59$.*

The following claims are easy to verify.

Claim 2.2. *If $(f, dist)$ is an embedding such that for any $v, t \in V(G)$, $v \neq t$ there is a vertex $w \in N(v)$ such that $d(w, t) < d(v, t)$ then $(f, dist)$ is a guaranteed greedy embedding.*

A greedy embedding that satisfies the above condition will be called *distance decreasing.*

Claim 2.3. *Let G' be a connected spanning subgraph of G. Then any distance decreasing greedy embedding of G' is a distance decreasing greedy embedding of G.*

Note that Claim 2.3 greatly simplifies our task: it is sufficient to give a distance decreasing greedy embedding of a spanning tree T of G. If we tried to deliver messages only in T, then the message would make very big detours compared to the shortest path in G. Fortunately, the routing considers all edges of G, so the number of hops taken by the routing is comparable to the shortest path in G.

To construct our embedding, we use various trees. Our trees are usually rooted and ordered, that is, for every vertex v there is an ordering defined on its children. Occasionally, we use partially ordered trees, where we only specify partial orderings on the children of each vertex. For binary trees, we call the first child as left child and the other as right child. If a vertex has only one child, then it will be regarded as a left child. For each vertex we define its *depth* (notation: $\mathrm{depth}(v)$) as its distance from the root. We define the *subtree* of a vertex v as the tree spanned by v and all its descendants.

For the rest of this section we discuss binary trees. We associate a bit sequence and a number with a vertex v of a rooted and ordered binary tree. (From now on we use the abbreviation ROBT for rooted ordered binary trees.) The bit sequence $\mathrm{code}(v)$ is defined recursively: for the root r let $\mathrm{code}(r)$ be the empty bit sequence, and for any other vertex v with parent p let

$$\mathrm{code}(v) = \begin{cases} \mathrm{code}(p)^\frown 0 & \text{if } v \text{ is the left child of } p \\ \mathrm{code}(p)^\frown 1 & \text{otherwise} \end{cases}$$

where $^\frown$ denotes the concatenation of two bit sequences.

Let $\mathcal{S}$ be the set of finite $0-1$ sequences. The code function defines a bijection between the vertices of the infinite binary tree B_∞ and $\mathcal{S}$. Vertices of a ROBT are referred to by their code, and vice versa.

Another number that we associate with the vertex of a ROBT will be denoted by $\mathrm{diad}(v)$, and we use the same definition that Eppstein and Goodrich [2] does: for the root r let $\mathrm{diad}(r) = \frac{1}{2}$ and for a vertex v with parent p let

$$\mathrm{diad}(v) = \begin{cases} \mathrm{diad}(p) - 2^{-\mathrm{depth}(v)-1} & \text{if } v \text{ is the left child of } p \\ \mathrm{diad}(p) + 2^{-\mathrm{depth}(v)-1} & \text{if } v \text{ is the right child of } p \end{cases}$$

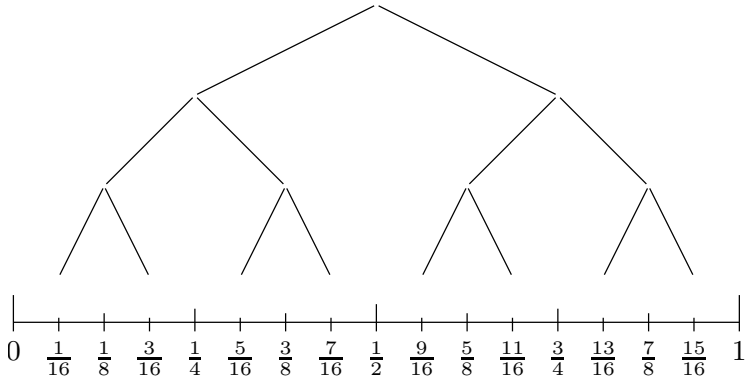

Figure 1: A rooted ordered binary tree; bijection with diadic rational numbers.

Observe that for any vertex v, $0 < \mathrm{diad}(v) < 1$ and the bit sequence $\mathrm{code}(v)^\frown 1$ is the sequence of binary fractions of $\mathrm{diad}(v)$. Note that diad defines an ordering on the vertices of the binary tree. We can think about a drawing of the tree in the plane as depicted in Figure 1. The order of the x coordinates of the vertices is the same as the ordering defined by diad. In the rest of the paper saying that u is on the left of v refers to this drawing: it is equivalent to $\mathrm{diad}(u) < \mathrm{diad}(v)$.

3 Embedding method

Our first goal is to embed a spanning tree into an infinite binary tree, and use the encodings described in Section 2 to make binary sequences. In order to decrease code length (the depth of the binary tree) we use *heavy paths* that are described below.

We begin by choosing an arbitrary spanning tree T of G, and we also a fix a root $r_T \in V(T)$. Let $\mathrm{des}(v)$ be the number of descendants of v: the number of vertices in the subtree defined by v, including v itself. (E.g., $\mathrm{des}(v) = 1$ for leaves and $\mathrm{des}(r_T) = |V(T)|$.)

Take an edge uv where u is the parent of v. The edge uv is *heavy* if $\mathrm{des}(v) \geq \mathrm{des}(u)/2$. Note that each vertex has at most one outgoing heavy edge. All other edges are *light*. We say that a child is heavy or light accordingly.

We introduce a partial ordering on the children of every vertex in T that has a heavy child: we simply require that the heavy edge is the first (leftmost) edge. The ordering of the light children is not fixed.

The heavy edges together form vertex disjoint *heavy paths*. For a vertex v on a heavy path P we define its child group as the set of light children: $\{w \in V(T) \mid vw \in$

$E(T),\ w \notin V(P)\}$.

By contracting the heavy paths we get the new tree Z. We denote the contraction by $\varphi : V(T) \to V(Z)$. We define another partial ordering on Z: for each vertex of Z, the ordering is based on the ancestor heavy path. Let u and v be vertices of T for which $\varphi(u) = \varphi(v) = w_Z$ and u is a descendant of v. Let $L(u)$ and $L(v)$ be the set of light children of u and v, respectively. For the children of w_Z, we constrain the ordering so that for any $x \in L(u)$ and $y \in L(v)$ the vertex $\varphi(x)$ is on the left of $\varphi(y)$ (The image of $L(u)$ is on the left of the image of $L(v)$). Figure 2 shows an example of this procedure, with correct left-to-right ordering.

For any vertex $v_Z \in V(Z)$ let $\varphi^{-1}(v_Z)$ be the set of vertices in T contracted to v_Z, or equivalently, the vertices on the heavy path corresponding to v_Z. We can think of each vertex in Z as the image of a (possibly 1-vertex) heavy path.

Next, we produce a binary tree B with root r_B that is essential in our embedding. This binary tree is an aggregate of smaller binary trees. It is constructed in a bottom-up manner, creating small binary trees for each heavy path (called *heavy path trees*, or in short HPTs), and a small binary tree for the light children of every vertex (called *light children trees*, or in short LCTs). For a vertex $v_Z \in V(Z)$, we produce a small ROBT denoted by $HPT(v_Z)$, whose leafs are assigned to the vertices on the heavy path $\varphi^{-1}(v_Z)$, with the same left-to-right ordering as in the heavy path. In case of single vertex heavy paths, $HPT(v_Z)$ is required to be the single vertex tree. For a vertex v_T of the tree T which has at least one light child, we make a small ROBT denoted by $LCT(v_T)$, whose leaves correspond to the light children of v_T. The method for building these small ROBTs will be described in Section 4.

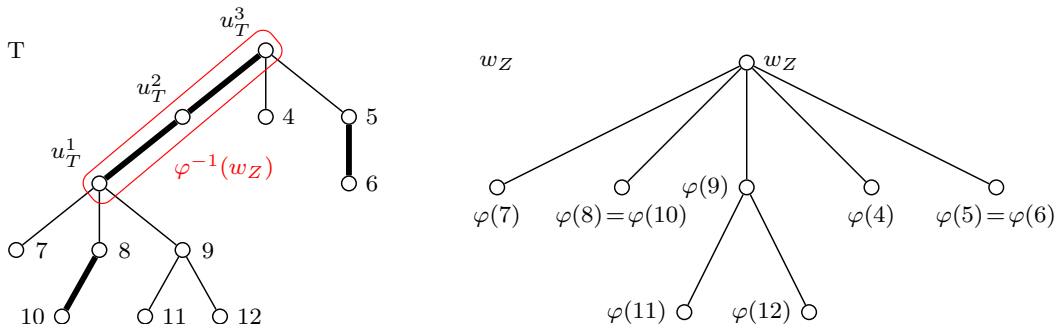

Figure 2: The first two steps of our embedding: finding the heavy paths in the spanning tree, and the tree Z obtained by contracting the heavy paths.

The algorithm processes the leaves of Z first, and later always picks a vertex whose

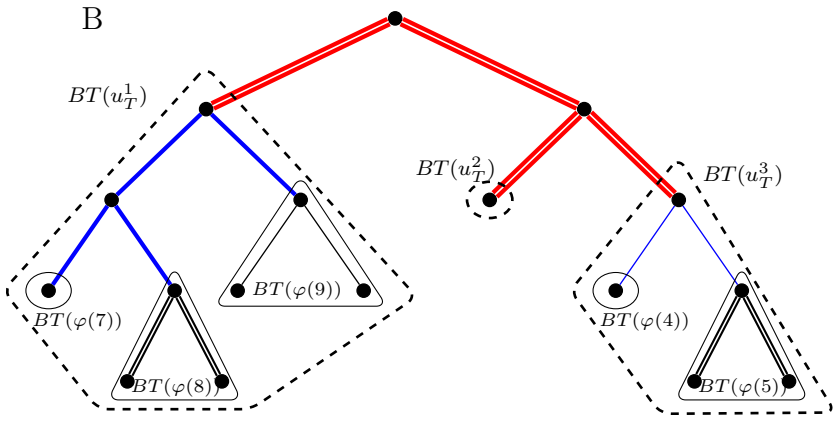

Figure 3: The resulting binary tree. We show the processing of the vertex w_z. The thick double edges represent $HPT(w_z)$, and the thick single edges are the edges of $LCT(u_T^1)$.

children have been previously processed. During this process B is constructed. Let v_Z be a leaf of Z. We produce the heavy path tree corresponding to the heavy path $\varphi^{-1}(v_Z)$. Let $BT(v_Z) = HPT(v_Z)$. Later, suppose we have assigned a binary tree $BT(v_Z)$ to all processed vertices v_Z of Z, and let w_Z be the next unprocessed vertex of Z (so w_Z is a vertex whose children have been processed).

First, create the light children tree $LCT(u_T)$ for each vertex u_T of the heavy path $\varphi^{-1}(w_Z)$. Let u_T^c be a light child of u_T. The vertex $\varphi(u_T^c) \in V(Z)$ is a child of v_Z, so it has been already processed, and has a binary tree $BT(\varphi(u_T^c))$ assigned to it. We glue $BT(\varphi(u_T^c))$ to the leaf of $LCT(u_T)$ that corresponds to (u_T^c) (i.e., the root of $BT(\varphi(u_T^c))$ will be the leaf). We repeat this procedure for every light child of u_T, resulting in a tree $BT(u_T)$.

Second, we create the heavy path tree $HPT(w_Z)$, which has a leaf associated with u_T, to which we attach $BT(u_T)$ for each $u_T \in \varphi^{-1}(w_Z)$. The resulting tree is assigned to w_Z (denoted by $BT(w_Z)$). See Figure 3 for an example of the processing.

After the above procedure terminates, all small trees are glued together into a final binary tree $B = BT(r_Z)$, the tree assigned to the root of Z by the algorithm. For a vertex $v \in G$, let $c_1(v)$ be the vertex of B that is the root of $HPT(\varphi(v))$ and let $c_2(v)$ be the leaf of $HPT(\varphi(v))$ that corresponds to v. See Figure4 for the values of c_1 and c_2 in our previous example.

Notice that $c_1(v)$ is always an ancestor of $c_2(v)$, and for single-vertex heavy paths, $c_1(v) = c_2(v)$ holds. Moreover, the function c_1 can be thought of as a function from

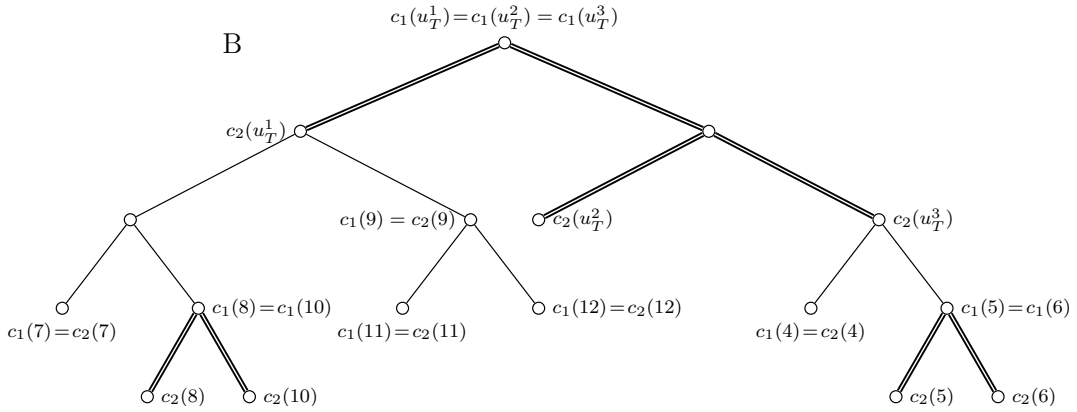

Figure 4: The values of c_1 and c_2.

$V(Z)$ to B, and this function keeps ancestor-descendant relations between vertices of Z, and for $v, w \in V(T)$ who are on the same heavy path, $\mathrm{diad}(c_2(v)) < \mathrm{diad}(c_2(w))$ if and only if v is a descendant of w.

Finally, we need to encode the vertices $c_1(v)$ and $c_2(v)$ in a single bit sequence. The bit sequences $\mathrm{code}(c_1(v))$ and $\mathrm{code}(c_2(v))$ are easy to encode, we use the following standard procedure:

Lemma 3.1. *Let s_2 be a bit sequence of length k with a starting slice s_1. Then s_1 and s_2 can be encoded in a machine word of length at least $k + \lceil \log k \rceil + 1$.*

Proof. Given the bit sequence s_1 and s_2, we use the lower $k + \lceil \log k \rceil + 1$ bits of the machine word. We set the first of these bits to 1, then follow it with $\lceil \log k \rceil$ bits on which we write the length of s_1. Note that the length of s_1 is at most k, so its length can be represented on $\lceil \log k \rceil$ bits. Finally, we write s_2 on the remaining k bits. To encode such a sequence, we need to determine k. First, we find the highest bit that is set, that gives us the value $f(k) = k + \lceil \log k \rceil + 1$. It is straightforward to check that for any $k \geq 1$ the value $\lceil \log f(k) \rceil$ is either $\lceil \log k \rceil$ or $\lceil \log k \rceil + 1$. This also gives us the value of k. □

4 Optimal depth binary trees

We are looking for an embedding that provides a minimum depth binary tree B, given our definition of heavy edges, our ordering restrictions and our requirements on the heavy path trees (HPTs) and light children trees (LCTs).

First, we define the LCT building method. Let v_T be a vertex of T that has at least one light child. We redefine z_i to be the depth of $BT(\varphi(c_i))$ where $c_i \in V(T)$ is a light child of v_T. These depths are available to us by our construction in 3. The LCTs will be constructed by a slight modification of Huffman's coding algorithm [3]. For the unordered sequence $\mathcal{Z} = \{z_1, z_2, \ldots, z_k\}$, we choose the two smallest values (z_i and z_j), and build a tree of 3 vertices, where the two children of the root correspond to z_i and z_j. In $\mathcal{Z}$, we delete z_i and z_j, and add $1 + \max\{z_i, z_j\}$. We continue taking the two smallest value from $\mathcal{Z}$ and adding the new element (their maximum plus 1), until there is only one element left in $\mathcal{Z}$, and its value is the depth of the optimal subtree. In the meantime we also gave the construction for the optimal ROBT.

We now turn to the HPT building method. We use a dynamic programming algorithm which is a modification of the naïve implementation of Knuth's algorithm for binary search trees [6].We require an initial depth for each heavy path vertex: let $z_i = \text{depth}(BT(u_T^i))$ where u_T^i is a vertex of the heavy path. Again, these depth are available to us by our construction in Section 3.

We use dynamic programming to determine the optimal depth. If we are given a sequence $z_1, z_2, \ldots, z_k$, we are looking for a ROBT that has each z_i assigned to a leaf v_i, and

$$\max_{i \in \{1,\ldots,k\}} \{\text{depth}(v_i) + z_i\} \qquad (*)$$

(the depth of the whole subtree) is minimal, while the leaves $v_1, v_2, \ldots, v_k$ are ordered from left to right. Let Odt_i^j denote the optimal depth tree for the sequence $z_i, z_{i+1}, \ldots, z_j$, and let od_i^j be the optimal subtree depth (we take the maximum between i and j in equation $*$), for each $1 \leq i \leq j \leq k$. We can set Odt_i^i to be the 1-vertex tree and $od_i^i = 0$. The dynamic programming will compute the values od_i^j in increasing order of $j - i$ using the following simple formula:

$$od_i^j = \min_{i \leq t < j} \max\{1 + od_i^t, 1 + od_{t+1}^j\}.$$

Let m be an index for which this formula takes its minimum value, and let r be the root of Odt_i^j, to which we add two children. The tree Odt_i^j can be constructed by attaching the tree Odt_i^m by its root to the left child of r and attaching the tree Odt_{m+1}^j by its root to the right child of r. It is routine to prove that this construction gives a minimum depth tree.

5 Computing the 'distance'

Recall that the function named code defined in Section 2 maps the vertices of the rooted binary tree B into $0-1$ sequences. Let $d_B : V(B) \times V(B) \to \mathbb{N}$ denote the (hop) distance of two vertices in B. Furthermore, let $a \wedge_W b$ denote the lowest common ancestor of a and b in the rooted tree W.

Our asymmetric distance function $d : V(T) \times V(T) \to \mathbb{R}_{\geq 0}$ is the sum of two functions: $d = d_1 + d_2$, where the summands are defined in the following way:

$$d_1(a,b) = \begin{cases} d_B(c_1(a), c_1(b)) & \text{if } c_1(a) \wedge_B c_1(b) \in \{c_1(a), c_1(b)\} \\ d_B(c_1(a), c_1(b)) + d_B(r_B, c_1(a) \wedge_B c_1(b)) & \text{otherwise} \end{cases}$$

$$d_2(a,b) = \begin{cases} \dfrac{\mathrm{diad}(c_2(a))}{2} & \text{if } a \wedge_T b = a \\ 1 - \dfrac{\mathrm{diad}(c_2(a))}{2} & \text{otherwise} \end{cases}$$

In order to make this distance function suitable for greedy routing, we need to be able to compute it using only the values of c_1 and c_2. It is easy to see that all cases of both d_1 and d_2 are defined in terms of c_1 and c_2, so the problem lies in distinguishing between the cases. Fortunately we can easily find the vertex $x \wedge_B y$ for any $x, y \in \mathcal{S}$: we just need to take the longest common prefix of the two sequences. This solves the problem of distinguishing the cases in the definition of d_1.

We now turn to distinguishing between the cases of d_2. If a is an ancestor of b in T, then there are two possible scenarios. Notice that the path P_{ab} (the unique path from a to b in T) starts with a light edge if and only if $c_1(b)$ is a descendant of $c_2(a)$ in B. This can be checked by verifying $c_2(a) \wedge_B c_1(b) = c_2(a)$.

Otherwise, let a' be the vertex on the heavy path of a that is incident to the first light edge $a'v$ on P_{ab}. We claim that a is an ancestor of b in T if and only if $c_1(a)$ is an ancestor of $c_1(b)$ in B and b is on the left of a, that is, $\mathrm{diad}(c_2(b)) < \mathrm{diad}(c_2(a))$.

Vertex a is an ancestor of vertex b if and only if $\varphi(a)$ is an ancestor of $\varphi(b)$ in Z or $\varphi(a) = \varphi(b)$ and a' is a descendant of a on the heavy path. As we noted earlier, the function c_1 preserves ancestor-descendant relations of heavy paths, so the requirement that $c_1(a)$ is an ancestor of $c_1(b)$ is equivalent to saying $\varphi(a)$ is an ancestor of $\varphi(b)$ in Z. It is easy to verify (using the definition of diad) that the second requirement $(\mathrm{diad}(c_2(b)) < \mathrm{diad}(c_2(a)))$ is satisfied if and only if $a' \neq a$ is a descendant of a.

Theorem 5.1. *Our embedding is greedy, that is, using the distance function $d(v,t)$ for the greedy forwarding algorithm (where the recipient is t) will always deliver the message in a finite number of steps.*

Before proving the theorem, we would like to revisit a previous approach to illustrate why the seemingly complicated distance function is needed.

5.1 Revisiting some other approaches

Aside from minor differences[1], the initial distance function of [2] is

$$d_{EG}(a,b) = d_B(c_1(a), c_1(b)) + |\operatorname{diad}(c_2(a)) - \operatorname{diad}(c_2(b))|.$$

Later they define a hyperbolic embedding using (almost the same) representation of each vertex in $\mathcal{S} \times \mathcal{S}$, but their proof heavily rely on the greediness of their tree embedding with distance d_{EG}. However their reasoning contains flaws: there are multiple examples of graphs for which the message cannot be delivered using this distance function. (In fact, in a random tree of at least 100 vertices the message will get stranded with high probability.) Consider e.g., the graph depicted in Figure 5.

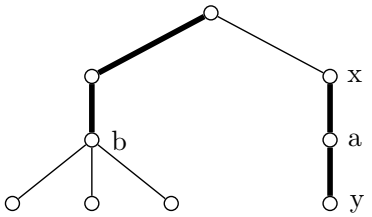

Figure 5: The message cannot get from a to b.

We are sending a message from a to b. It follows from the construction that $\operatorname{diad}(c_2(b)) < \operatorname{diad}(c_2(y)) < \operatorname{diad}(c_2(a)) < \operatorname{diad}(c_2(x))$. Since $c_2(a) = c_2(x) = c_2(y)$, the distances $d_{EG}(y,b) < d_{EG}(a,b) < d_{EG}(x,b)$, so the message will be forwarded to y instead of x, and reach a local minimum there.

[1]The autocraticness of the binary trees and the dummy vertices required in [2] do not have any influence on the issue we bring up here. We do not introduce the full setup, but kindly direct the reader to the original paper.

5.2 Proof of Theorem 5.1

Proof. We are delivering a message from vertex a to b ($b \neq a$) in G. By Claim 2.2 and Claim 2.3, it is sufficient to prove that in each step we can choose a neighbour $v \in N_T(a)$ for which $d(v, b) < d(a, b)$. For our proof, we need the following claim.

Claim 5.2. *Let $H = (v_1, v_2, \ldots, v_k)$ be a heavy path in T with topmost vertex v_k, and let b be a descendant of v_t in T (where $1 \leq t \leq k$). Then the following inequalities hold:*

$$d_2(v_1, b) > d_2(v_2, b) > \cdots > d_2(v_t, b)$$

and

$$d_2(v_t, b) < d_2(v_{t+1}, b) < \cdots < d_2(v_k, b).$$

Proof of Claim 5.2. The inequalities for v_i and v_{i+1} are an immediate consequence of the definition of d_2 if $i \leq t-2$ or $i \geq t$. If $i = t-1$, then $d_2(v_{t-1}, b) = \frac{1-\mathrm{diad}(c_2(v_{t-1}))}{2}$ and $d_2(v_t, b) = \frac{\mathrm{diad}(c_2(v_t))}{2}$. By the definition of diad, $0 < \mathrm{diad}(x) < 1$ for any $x \in \mathcal{S}$, so $d_2(v_{t-1}, b) > \frac{1}{2} > d_2(v_t, b)$. □

Let P_{ab} be the unique path from a to b in T, and let a^* be the second vertex (the vertex next to a). We show that $d(a^*, b) < d(a, b)$.

Case 1. $c_1(a) = c_1(b)$. In this case $d_1(a, b) = d_1(a^*, b) = 0$, and by applying Claim 5.2 we get that $d_2(a^*, b) < d_2(a, b)$, so $d(a^*, b) < d(a, b)$ follows.

Case 2. $c_1(a)$ and $c_1(b)$ are ancestor and descendant (in some order). It follows that $\varphi(a)$ and $\varphi(b)$ are ancestor and descendant in the same order. If edge aa^* is light, then $d_1(a^*, b) \leq d_1(a, b) - 1$ and $|d_2(a^*, b) - d_2(a, b)| < 1$ according to definitions, thus $d(a^*, b) < d(a, b)$.

If edge aa^* is heavy, then $d_1(a, b) = d_1(a^*, b)$, so we need to show that $d_2(a^*, b) < d_2(a, b)$. If a^* is a child of a, then b is a descendant of a in T. It also follows that $\mathrm{diad}(a^*) < \mathrm{diad}(a)$, thus $d_2(a^*, b) < d_2(a, b)$. If b is an ancestor of a^*, then b is also an ancestor of a as well, so $d_2(a, b) = \frac{1-\mathrm{diad}(c_2(a))}{2}$ and $d_2(a^*, b) = \frac{1-\mathrm{diad}(c_2(a^*))}{2}$. Since $\mathrm{diad}(a) < \mathrm{diad}(a^*)$, we get that $d_2(a^*, b) < d_2(a, b)$. Otherwise we apply Claim 5.2 and get that $d_2(a^*, b) < d_2(a, b)$.

Case 3. $c_1(a)$ and $c_1(b)$ are on different branches in B (none is descendant of the other). It is easy to check that in this case a and b are on different branches in T as well, so a^* is the parent of a.

If $c_1(a^*) = c_1(a)$, then $d_1(a^*, b) = d_1(a, b)$. We know that $d_2(a^*, b) = \frac{1-\mathrm{diad}(c_2(a^*))}{2}$ and $d_2(a, b) = \frac{1-\mathrm{diad}(c_2(a))}{2}$, and since a^* is the parent of a, we get $d_2(a^*, b) < d_2(a, b)$.

From now on, we suppose that $c_1(a^*) \neq c_1(a)$. If $c_1(a^*)$ is an ancestor of $c_1(b)$ in B, then observe that – as $c_1(a)$ and $c_1(b)$ are both descendants of $c_1(a^*)$ – the vertex $c_1(a) \wedge_B c_1(b)$ is either $c_1(a^*)$ or its descendant[2].

We have

$$\begin{aligned} d_B(c_1(a^*), c_1(b)) =& d_B(c_1(a^*), c_1(a) \wedge_B c_1(b)) + d_B(c_1(a) \wedge_B c_1(b), c_1(b)) \\ \leq& d_B(r_B, c_1(a) \wedge_B c_1(b)) + d_B(c_1(a), c_1(b)) - 1 \end{aligned}$$

because a^* is on a path from r_B to $c_1(a) \wedge_B c_1(b)$ and $c_1(a) \neq c_1(a) \wedge_B c_1(b)$. Using $|d_2(a^*, b) - d_2(a, b)| < 1$ we proved $d(a^*, b) < d(a, b)$ for this sub-case.

If $c_1(a^*)$ is not an ancestor of $c_1(b)$, then $d_1(a^*, b) \leq d_1(a, b) - 1$ by the definition of d_1, so by using $|d_2(a^*, b) - d_2(a, b)| < 1$ we have proved the final sub case. □

6 Alternative ROBT building method

We describe a second ROBT building procedure. Although this procedure yields a binary tree of suboptimal depth, it is easier to upper bound the depth of its resulting tree, and it has the same worst-case performance as the optimal building method.

Weight based `ROBT` procedure:

For both the HPTs and LCTs we define a weight sequence. Each individual weight in this sequence will be assigned to a leaf of the HPT or LCT. We fix an ordering on the light children for every vertex (recall that this was not fixed previously): let the left-to right ordering of the children be decreasing in the number of descendants, i.e., for child u and v if $\text{des}(u) > \text{des}(v)$ then u is on the left of v. (We break ties arbitrarily.) When building our LCTs and HPTs, we preserve the ordering, that is, the ordering of the assigned weights of the leaves from left to right will be the same as the ordering of the original weight sequence.

We can assign a weight sequence to each vertex of an ROBT: the sequence assigned to vertex v consists of the weights assigned to the descendant leaves of v, and the ordering is the same as the left-to-right ordering on those leaves. This assignment maps the root of the ROBT to the full initial weight sequence.

Let $v_0, v_1, \ldots v_k$ be the vertices of a heavy path traversed upward, so the topmost vertex is v_k. Let the weight sequence for the the root of the HPT be $(\text{des}(v_0), \text{des}(v_1) -$

[2]If $c_1(a) \wedge_B c_1(b)$ is a descendant of $c_1(a^*)$, then $d_B(c_1(a^*), c_1(b)) + 1 \leq d_B(c_1(a), c_1(b))$ is not necessarily true. This is the reason why we had two separate cases in the definition of d_1.

$\mathrm{des}(v_0), \mathrm{des}(v_2) - \mathrm{des}(v_1), \ldots, \mathrm{des}(v_k) - \mathrm{des}(v_{k-1}))$, or in other words, the i-th weight is the total sum of its child group's descendants plus 1. For a vertex with light children $\{u_1, u_2, \ldots u_k\}$ let the weights of the LCT root be $(\mathrm{des}(u_1), \mathrm{des}(u_2), \ldots, \mathrm{des}(u_k))$ (so in this case, the weights are a non-increasing sequence of integers).

Let $\underline{w} = (w_1, w_2, \ldots, w_k)$ be a weight sequence ($w_i \in \mathbb{N}^+$). Let $s_i = \sum_{j=1}^{i} w_j$ (for $i = 1, 2, \ldots, k$), and let $|\underline{w}| = s_k$ be the total weight of the weight sequence. These s_i values are called *separators*. A *cut* of the weight sequence $\underline{w}$ along separator s_i ($1 \leq i \leq k-1$) divides the weight sequence into two child sequences. The left child sequence is $(w_1, w_2, \ldots w_i)$ and the right child sequence is $(w_{i+1}, w_{i+2}, \ldots, w_k)$.

We define our ROBT building method as a sequence of cuts. For each weight sequence, we need to find a separator at which we perform the cut. This will be defined depending on the weight sequence class (these classes will be defined later). The left and right child sequences will be assigned to the two children of the current vertex. For each child, we classify the weight sequence and make a cut accordingly. Continuing this cutting method recursively leads to a well-defined ROBT.

6.1 Cuts and weight sequence classes

Let $\underline{w} = (w_1, w_2, \ldots, w_k)$ be a weight sequence with total weight $W = |\underline{w}|$. We denote by b the index of the biggest weight. (If there are multiple biggest weights, we set b to the least index among them.) Let α be the ratio of the biggest weight to the sum of weights, that is, $w_b = \alpha W$. Furthermore, we denote by ζ the constant $2^{-1/3} \approx 0.794$.

We use one of the following methods to determine the separator for a cut.

Balanced cut: Let s_t be the separator for which $|W/2 - s_t|$ is minimal (the separator closest to the 'middle' of the sequence).

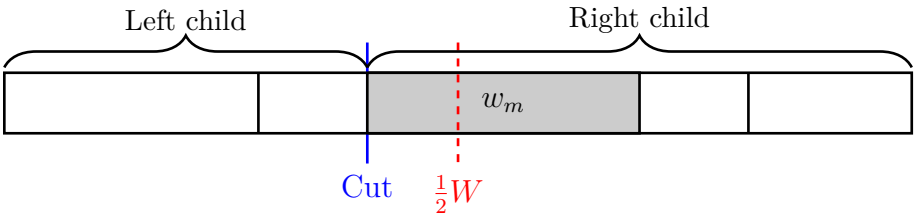

Figure 6: Example of balanced cut

Left-heavy cut: Let s_t be the least separator not smaller than $W/2$ (so $s_t \geq W/2$). If $s_t > \zeta W$ and $t = 1$, then we cut at s_t. If $s_t > \zeta W$ and $t > 1$, then we cut at s_{t-1}.

If $s_t \leq \zeta W$ then let j be the index of the biggest weight in $(w_{t+1}, w_{t+2}, \dots w_k)$. If $s_{j-1} > \zeta W$ then we cut at s_t, otherwise we cut at s_{j-1}. Notice that this method is designed to make the cut in a way that the the second weight sequence has its biggest weight in its first element as long as the left child is not too large.

Right-heavy cut: We perform the 'reflection' of left-heavy cut: we cut at $W - x$ where x is the separator for left-heavy cut on the reverse weight sequence $(w_k, w_{k-1}, \dots, w_1)$.

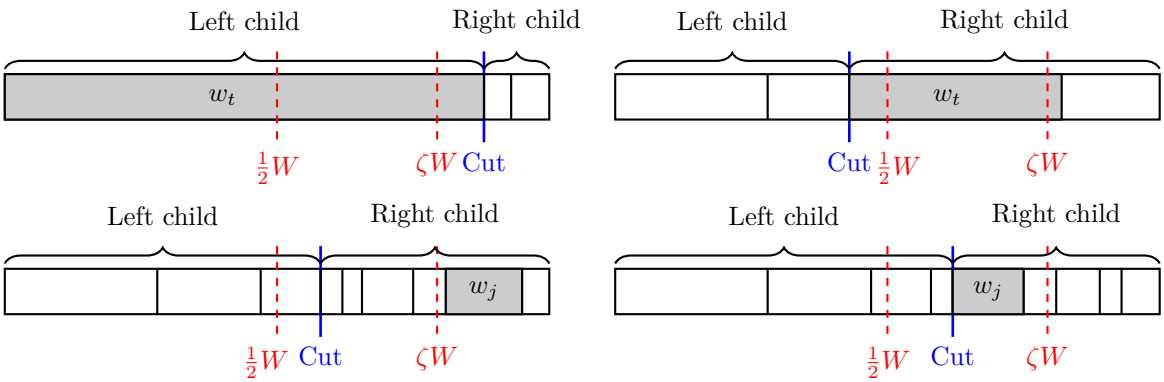

Figure 7: The four possible cases of a left-heavy cut

Next, we describe our weight classes. The classes are organized into class types. Most class types have a real parameter α, which is always the ratio of the biggest weight to the sum of the weights in the weight sequence. The weight sequence classes themselves are just class types with the parameter value fixed.

In case of HPT building, we need to classify only those weight sequences that are assigned to non-leaf vertices of the HPT, since the weights assigned to leafs will be treated in the LCT attached to this leaf, or in case of weight 1, the vertex will be a leaf of the large binary tree. Notice that in case of HPTs every non-leaf vertex has a weight sequence consisting of at least two weights.

We approach the LCTs similarly: we only classify non-leaf vertices. Contrary to HPTs though, it is possible that a non-leaf vertex has a singleton weight sequence assigned to it: if the HPT corresponds to a vertex v_T that is a non-ending point of its heavy path, and v_T has a single light descendant, then the root of the LCT will be assigned a single weight.

Weight sequence classes arising when creating an LCT:

- $\mathbf{D}_\alpha$ $(0 < \alpha \leq 1)$

  Sequence $\underline{w}$ is in $\mathbf{D}_\alpha$ (*'Decreasing'*) if $w_1 = \alpha|\underline{w}|$. (Recall that for LCTs, $w_1 \geq w_2 \geq \cdots \geq w_k$, so the child sequences of a cut are also in a decreasing order.) We use balanced cut to cut class $\mathbf{D}_\alpha$.

Weight sequence classes arising when creating an HPT:

- **H**

  Sequence $\underline{w}$ is in **H** (*'Heavy'*) if $\underline{w}$ contains the first weight of the weight sequence assigned to the root of the HPT. When cutting weight sequences of class **H**, we are going to use left-heavy cuts.

- $\mathbf{FL}_\alpha$ $(0 < \alpha < 1)$

  Sequence $\underline{w}$ is in FL_α (*'First-Last'*) if it is not in **H**, and its biggest weight (or one of its biggest weights) is the first weight or the last weight, so $w_1 = \alpha|\underline{w}|$ or $w_k = \alpha|\underline{w}|$. If $\alpha > \frac{1}{2}$, then we use a balanced cut. If $\alpha \leq \frac{1}{2}$ and $w_1 = \alpha|\underline{w}|$, then we will use a left-heavy cut, otherwise (if $\alpha \leq \frac{1}{2}$ and $w_k = \alpha|\underline{w}|$) we use right-heavy cut.

- $\mathbf{M}_\alpha$ $(0 < \alpha < 1)$

  Sequence $\underline{w}$ is in M_α (*'Middle'*) if the biggest weight is some weight $w_j = \alpha|\underline{w}|$, such that $1 < j < k$ and $w_1 \neq w_j \neq w_k$. We apply balanced cut for this class.

Now we have fully described our HPT and LCT building method.

7 Bounding the height of the large binary tree

We define a constant for each weight sequence class. These constants are denoted by the non-bold version of the name of the class, e.g., the class constant of $\mathbf{D}_\alpha$ is D_α.

For a function $f(x)$ let $(f(x))^+ = \max(0, f(x))$. We define our class constants the following way:

$$
\begin{aligned}
D_\alpha &= -1 + (2 + 3\log(\alpha))^+ \\
H &= 0 \\
FL_\alpha &= 1 + (1 + 3\log(\alpha))^+ \\
M_\alpha &= 2 + (1 + 3\log(\alpha))^+
\end{aligned}
$$

Note that the functions $(2+3\log(\alpha))^+$ and $(1+3\log(\alpha))^+$ are monotone increasing, and their value is 0 on the interval $[0, \zeta^2]$ and $[0, \zeta]$ respectively. (Some approximate values for these constants: $\zeta \approx 0.794$ and $\zeta^2 \approx 0.630$.)

Usually we are looking for a higher bound on the class constant. We denote by D_1, FL_1 and M_1 the supremum of the constants in each class type. (By our definition, these values are $1, 2$ and 3 respectively.)

Let $\text{height}_B(v)$ be the height of v in the large binary tree B (or equivalently, the height of the subtree of v in B, so $\text{height}_B(r_B)$ is the height of B). Our upper bound for the subtree depth of a vertex v in the binary tree is $3\log W + c$, where c is the class constant of the weight sequence of v, and W is the total weight of v's weight sequence class. We will state this more precisely below. The following two theorems will be the key to prove our main theorem.

Theorem 7.1. *Let α be the parameter of the weight sequence $\underline{w} = (w_1, w_2, \dots w_k)$ assigned to the root r of an LCT. Suppose that* $\text{height}_B(l_i) \leq 3\log w_i$*, where l_i is the leaf to which we assigned the weight w_i in the LCT. It follows that* $\text{height}_B(r) \leq 3\log W + D_\alpha$*. (As before, W is the total weight of $\underline{w}$.)*

Theorem 7.2. *Let r be the root of an HPT with weight sequence $(w_1, w_2, \dots w_k)$ and total weight W. Suppose that* $\text{height}_B(l_1) \leq \max(3\log w_1 - 1,\ 0)$ *and* $\text{height}_B(l_i) \leq 3\log w_i + 1$ *for $2 \leq i \leq k$, where l_i is the leaf to which we assigned the weight w_i in the HPT. It follows that* $\text{height}_B(r) \leq 3\log W$*.*

Finally, we will prove the following upper bound on the height of the large binary tree B:

Theorem 7.3. *The height of B using the optimal tree building method is at most $3\log n$, where n is the number of vertices in the original graph.*

The following lemma will be used multiple times.

Lemma 7.4. *Let $\underline{w}$ be the weight sequence with at least 3 weights, and let w_m be the weight that covers the midpoint. Suppose that either $w_m \leq s_{m-1}$ or $w_m \leq W - s_m$. After a balanced cut in $\underline{w}$, both the resulting weight sequences have total weight at most $\frac{2}{3}W$.*

Proof. We prove this lemma by contradiction. Suppose that the cut happens outside the interval $\left[\frac{1}{3}W, \frac{2}{3}W\right]$. It follows that the middle weight covers the whole interval, so its size must be bigger than $W/3$. Thus $s_{m-1} < W/3$ and $W - s_m < W/3$, so we get that $s_{m-1} < w_m$ and $W - s_m < w_m$. This contradicts the condition in our lemma. □

Our aim is to upper bound the total weight of the child sequences with a constant factor of the total weight of the parent sequence (denoted by W). We will also need to give upper bounds for the constants of the new classes. A $(\mathbf{C}, r, p)$-*child* is a child weight sequence which is in class $\mathbf{C}$, has total weight at most rW, and its class parameter is at most p. (For classes without parameters we omit the third term in the brackets.)

7.1 Proof of Theorem 7.1

A special kind of a child is a leaf of the LCT. This child sequence always contains a single weight. We denote this child by $(\mathbf{L}, r)$ where r is the weight ratio as before.

Lemma 7.5. *The possible children of* $\mathbf{D}_\alpha$ *after applying our cutting method are:*

1. $(\mathbf{L}, \alpha)$
2. $(\mathbf{D}, \beta, \frac{1-\beta}{\beta})$, *where* $\frac{1}{2} \leq \beta \leq \frac{2}{3}$

Proof. If there is a single child class, then $\alpha = 1$, and the child is a leaf of weight ratio 1. Otherwise the smaller child class is either a leaf with weight ratio at most α, or it has class type $\mathbf{D}$, has weight ratio at most $\frac{1}{2}$, and its parameter is at most 1, so it is a $(\mathbf{D}, \frac{1}{2}, 1)$ child. This falls under the category of $(\mathbf{D}, \beta, \frac{1-\beta}{\beta})$, where $\beta = \frac{1}{2}$

The larger child class can also be a leaf, with weight at most α. If it is not a leaf, then it has class type $\mathbf{D}$. Let β be the weight ratio of the larger class. By Lemma 7.4, $\beta \leq \frac{2}{3}$.

The parameter of the child is the ratio of its first weight to its total weight. Its largest weight has size at most αW. If this larger child class is the right child, then it does not contain the first weight $w_1 = \alpha W$, so $\beta \leq 1 - \alpha$ follows. If this is a left child class, then by the definition of balanced cuts the separator at βW is closer to $\frac{1}{2}$ than the αW separator. It follows that $\alpha \leq \frac{1}{2}$ and $\beta - \frac{1}{2} \leq \frac{1}{2} - \alpha$, or equivalently, $\alpha \leq 1 - \beta$. So in both cases, the parameter of this child is at most $\frac{\alpha}{\beta} \leq \frac{1-\beta}{\beta}$, making it a $(\mathbf{D}, \beta, \frac{1-\beta}{\beta})$ child. □

To prove the theorem, we will prove the stronger proposition that for any non-leaf vertex v of the LCT with weight sequence $\underline{w}$, the inequality $\text{height}_B(v) \leq 3 \log W + D_\alpha$ holds, where W is the total weight of $\underline{w}$, and $\mathbf{D}_\alpha$ is its weight sequence class.

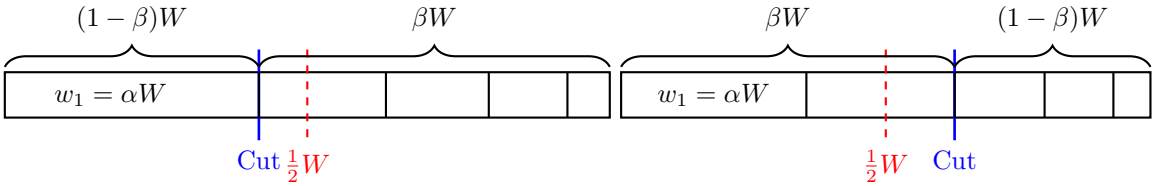

Figure 8: The possible ways of getting a $(\mathbf{D}, \beta, \frac{1-\beta}{\beta})$ child

We prove the statement by induction on W. For $W = 1$, we know that $\alpha = 1$, and the LCT has only two vertices, its root r and the leaf l of weight one, that is a leaf of the big binary tree. Since $\text{height}_B(l) = 0$, we get $\text{height}_B(r) = 1 = 3\log 1 + (-1) + (2 + 3\log 1)^+ = 1$.

Now suppose that $W \geq 2$. In the special case of a single weight class, the child is always a leaf, so by the theorem's condition on leaves ($\text{height}_B(l) \leq 3\log W$ for any leaf l), we get

$$\text{height}_B(r) = 1 + \text{height}_B(l) \leq 1 + 3\log W = 3\log W + D_1.$$

In all other cases, we can bound $\text{height}_B(v)$ the following way:

$$\text{height}_B(v) \leq \max\left(1 + \text{height}_B(v_1), 1 + \text{height}_B(v_2)\right),$$

where v_1 and v_2 are the children of v in the LCT. Since we already have an upper bound for both $\text{height}_B(v_1)$ and $\text{height}_B(v_1)$, we can apply induction because their weights are strictly smaller than W, so it is sufficient to prove that $3\log W + D_\alpha$ is not smaller than this bound. Using Lemma 7.5 we arrive at the following inequalities:

$$3\log W + D_\alpha \geq 1 + 3\log(\alpha W) \tag{1}$$

$$3\log W + D_\alpha \geq 1 + 3\log(\beta W) + D_{\frac{1-\beta}{\beta}} \qquad \text{where } \beta \in \left[\frac{1}{2}, \frac{2}{3}\right]. \tag{2}$$

We subtract $3\log W$, and substitute the class constants. In the second inequality, we decrease the left side by taking the minimum possible value of D_α, which is -1.

$$-1 + (2 + 3\log\alpha)^+ \geq 1 + 3\log\alpha \tag{3}$$

$$-1 \geq 1 + 3\log\beta + -1 + \left(2 + 3\log\frac{1-\beta}{\beta}\right)^+ \qquad \text{where } \beta \in \left[\frac{1}{2}, \frac{2}{3}\right] \tag{4}$$

Inequality (3) is straightforward. For inequality (4) we distinguish two cases. If $\frac{1-\beta}{\beta} \leq \zeta^2$:

$$0 \geq 1 + 3\log\beta \;\Leftrightarrow\; \beta \leq \zeta,$$

which is satisfied since $\beta \leq \frac{2}{3} < \zeta$. If $\frac{1-\beta}{\beta} > \zeta^2$:

$$3\log\beta + 2 + 3\log\frac{1-\beta}{\beta} = 2 + 3\log(1-\beta) \leq -1,$$

where the last inequality holds because $\beta \geq \frac{1}{2}$. □

7.2 Proof of Theorem 7.2

We begin by proving an important property of heavy paths.

Lemma 7.6. *Let $v_1, v_2, \ldots, v_k$ be the vertices of the heavy path in the original spanning tree from bottom to top, i.e., v_i is the parent of v_{i-1}. Let $\underline{w} = (w_1, w_2, \ldots, w_k)$ be the weight sequence assigned to the root of the HPT with separators $s_i = \sum_{j=1}^{i} w_j$, $j = 1, 2, \ldots k$. Then $w_i \leq s_{i-1}$ holds for all $i = 1, 2, \ldots k$.*

Proof. Let $v_i v_{i-1}$ be any edge on this heavy path. By the definition of the heavy path weight sequence,

$$s_{i-1} = \sum_{j=1}^{i-1} w_j = \mathrm{des}(v_1) + \sum_{j=2}^{i-1}(\mathrm{des}(v_j) - \mathrm{des}(v_{j-1})) = \mathrm{des}(v_{i-1}).$$

Since $w_i = \mathrm{des}(v_i) - \mathrm{des}(v_{i-1})$ our inequality can be rewritten as $\mathrm{des}(v_{i-1}) \geq \mathrm{des}(v_i)/2$, which is true by the definition of heavy edges. □

We will need two special classes for the leaves of an HPT. The leftmost leaf will be denoted by **SL** (*starting leaf*), and the rest of the leaves will be denoted by **OL** (*ordinary leaf*).

Lemma 7.7. *The possible children of* **H** *after applying the left-heavy cut are:*

1. $(\mathbf{SL}, 1)$
2. $(\mathbf{OL}, \frac{1}{2})$
3. $(\mathbf{H}, \zeta)$
4. $(\mathbf{FL}, \beta, \frac{1-\beta}{\beta})$, *where* $\frac{1}{2} \leq \beta \leq 1 - \frac{\zeta}{2}$
5. $(\mathbf{M}, \beta, \frac{1-\zeta}{\beta})$, *where* $1 - \zeta \leq \beta \leq \frac{1}{2}$
6. $(\mathbf{FL}, 1-\zeta, 1)$

Proof. The only way that the cut could happen at a separator $s \geq \zeta$ is if $w_1 \geq \zeta$ by the definition of left-heavy cuts. So the left child class is either a a $(\mathbf{H}, \zeta)$ child, or a starting leaf with arbitrary weight ratio (at most 1).

If the right child is a leaf, then it is ordinary. Note that its weight w_k cannot exceed $\frac{1}{2}W$, since $w_k \leq s_{k-1}$ by Lemma 7.6.

From now on we can assume that the right child class has at least two weights. Let s be the separator at which the cut happens. If $s \leq \frac{1}{2}W$, then the right child class has a big starting weight w_t: it must cover the interval $\left[\frac{W}{2}, \zeta W\right]$ according to the left-heavy cut procedure (see Figure 9). If this whole right child class has weight ratio β, then we should first observe that $\beta \leq 1 - \frac{\zeta}{2}$: the size of w_t is at least $\zeta W - s$, and at most s by Lemma 7.6. It follows that $\zeta W - s \leq s$, so $\beta W = W - s \leq (1 - \frac{\zeta}{2})W$. Since the size of w_t is much greater than $\frac{\beta}{2}$, this child class is of type FL, and the maximum size of w_t is at most $s_{t-1} = 1 - \beta$, so this is an $(\mathbf{FL}, \beta, \frac{1-\beta}{\beta})$ child.

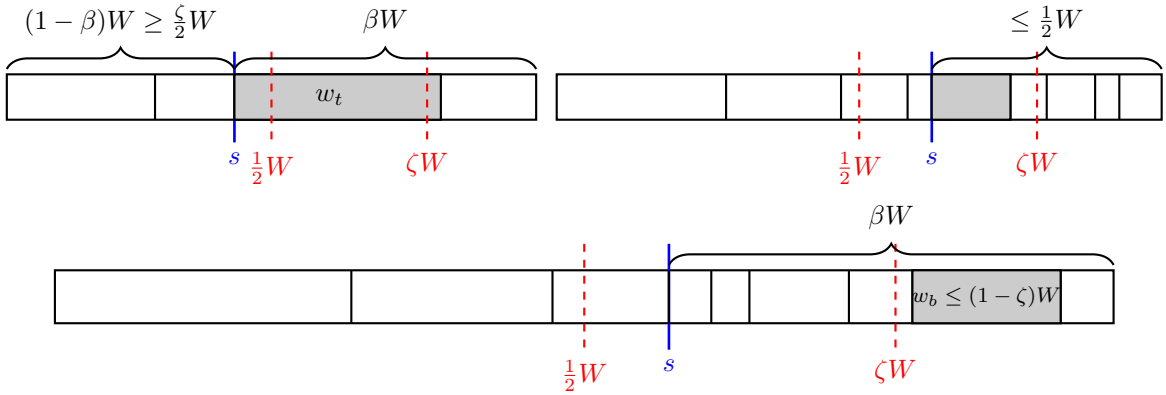

Figure 9: Cutting a weight sequence of type **H**

If $s \in \left[\frac{1}{2}W, \zeta W\right]$, then we distinguish two cases: if we find a suitable large starting weight for the right class, then we get an $(\mathbf{FL}, \frac{1}{2}, 1)$ child (which is included in category 6). Otherwise, the right child class is of type **M**, and the weight ratio is $\beta \in \left[1 - \zeta, \frac{1}{2}\right]$. The biggest weight w_b of this child class must lie inside the interval $[\zeta, 1]$, so $w_b \leq 1 - \zeta$, resulting in an $(\mathbf{M}, \beta, \frac{1-\zeta}{\beta})$ child, which is included in category 5 with $\beta = 1 - \zeta$.

If the cut happens in $[\zeta, 1)$, then the right side is either an $(\mathbf{FL}, 1 - \zeta, 1)$ or $(\mathbf{M}, 1 - \zeta, 1)$ child. □

Lemma 7.8. *The possible children of* $\mathbf{FL}_\alpha$ *are:*

1. $(\mathbf{OL}, \alpha, 1)$
2. $(\mathbf{FL}, \beta, \frac{\alpha}{\beta})$, *where* $\frac{1}{2} \leq \beta \leq \zeta$ *and* $\alpha \leq \frac{1}{2}$

3. $(\mathbf{FL}, \beta, \frac{1-\beta}{\beta})$, *where* $\frac{1}{2} \leq \beta \leq 1 - \frac{\zeta}{2}$ *and* $\alpha \leq \frac{1}{2}$

4. $(\mathbf{M}, \frac{1}{2}, 1)$

Proof. Suppose wlog. that w_1 is the largest weight, so we perform a left-heavy cut. Notice that a child class of type **H** is impossible, since every class of type **FL** is a non-starting subsequence of the root weight sequence. The smaller child can be a leaf (with weight ratio at most α). Otherwise we know that its weight ratio is at most $\frac{1}{2}$, so it is either a $(\mathbf{FL}, \frac{1}{2}, 1)$ (included in category 3 with $\beta = \frac{1}{2}$), or a $(\mathbf{M}, \frac{1}{2}, 1)$ child. From now on we will only need to deal with the larger child.

First, we examine the case $\alpha \leq \frac{1}{2}$. If the cut happens on the left of $\frac{1}{2}$, then the right child class begins with a weight w_t that covers the interval $\left[\frac{1}{2}, \zeta\right]$, so it is of type **FL**. Notice that although this is not a full heavy path, the conclusion of Lemma 7.6 holds for w_t, since $w_t \leq w_1 \leq s_{t-1} = (1-\beta)W$. So by the same argument as in the previous lemma, β satisfies $\frac{1}{2} < \beta \leq 1 - \frac{\zeta}{2}$ and $w_s \leq 1 - \beta$, resulting in an $(\mathbf{FL}, \beta, \frac{1-\beta}{\beta})$ child. If the cut happens on the interval $\left[\frac{1}{2}, \zeta\right]$, then the left child is an $(\mathbf{FL}, \beta, \frac{\alpha}{\beta})$ child.

If $\alpha > \frac{1}{2}$, then the larger child (the left child) is an ordinary leaf with weight ratio α. □

Lemma 7.9. *The possible children of* $\mathbf{M}_\alpha$ *are:*

1. $(\mathbf{OL}, \alpha, 1)$

2. $(\mathbf{M}, \frac{1}{2}, 1)$

3. $(\mathbf{FL}, \frac{1}{2}, 1)$

4. $(\mathbf{FL}, \beta, \frac{\alpha}{\beta})$, *where* $\alpha \leq \beta \leq \frac{1+\alpha}{2}$

5. $(\mathbf{M}, \beta, \frac{\alpha}{\beta})$, *where* $\frac{1}{2} < \beta \leq \frac{2}{3}$ *and* $\alpha \leq \frac{1}{2}$.

Proof. If a child sequence has only one weight, then it is an ordinary leaf of size at most αW. The smaller weight sequence is either a $(\mathbf{FL}, \frac{1}{2}, 1)$ or a $(\mathbf{M}, \frac{1}{2}, 1)$ child. We need to examine any possible larger weight sequence that has at least two weights.

Let the larger child have weight ratio β. Its parameter is at most $\frac{\alpha}{\beta}$. It can be of type **FL**, and in this case $\beta \leq \frac{1+\alpha}{2}$, since the weight covering $W/2$ has size at most α, so the closest separator will be at distance at most $\frac{\alpha}{2}$ from the midpoint. We show that $\beta \geq \alpha$. If $\alpha \leq \frac{1}{2}$, then this is trivial since $\beta \geq \frac{1}{2}$. Otherwise the weight of size

αW is contained in the larger child class, because it is the starting or ending weight of our class, since it covers the midpoint.

Alternatively, the larger child can have type **M**. Since the weight covering the midpoint (denoted by w_m) is either the first or last weight of this child class by the definition of balanced cuts, it is strictly smaller than αW. It follows that the largest weight w_j of size αW and w_m are distinct weights. So either $w_m \leq s_{m-1}$ or $w_m \leq 1 - s_m$. By Lemma 7.4 we get $\beta \leq \frac{2}{3}$. Notice that in this case $\alpha \leq \frac{1}{2}$ because otherwise the largest weight w_j covers the midpoint. □

Now we are ready to start the proof of Theorem 7.2. As before, we will use induction on W, and we will upper bound $\text{height}_B(v)$ by using our bounds on the children of v. Now we present our equations for class H.

Since the total weight in the classes **H**, **FL** and **M** is at least 2, we need to begin with the case $W = 2$. The only weight sequence with $|\underline{w}| = 2$ of that has at least two weights is $(1, 1)$, which can be either of type **H** or type $\mathbf{FL}_{\frac{1}{2}}$. It is straightforward to check the theorem's statement in both cases.

In the general case $W \geq 3$, we will have a lot of inequalities for all possible children of all weight sequence classes. We will sort the inequalities by the parent weight sequence class. The inequalities for class **H**:

$$3\log W + H \geq 1 + \max(3\log W - 1, 0) \tag{5}$$

$$3\log W + H \geq 1 + 3\log\left(\frac{1}{2}W\right) + 1 \tag{6}$$

$$3\log W + H \geq 1 + 3\log(\zeta W) + H \tag{7}$$

$$3\log W + H \geq 1 + 3\log(\beta W) + M_{\frac{1-\zeta}{\beta}} \qquad \text{where } \beta \in \left[1-\zeta, \frac{1}{2}\right) \tag{8}$$

$$3\log W + H \geq 1 + 3\log(\beta W) + FL_{\frac{1-\beta}{\beta}} \qquad \text{where } \beta \in \left[\frac{1}{2}, 1-\frac{\zeta}{2}\right] \tag{9}$$

$$3\log W + H \geq 1 + 3\log((1-\zeta)W) + FL_1 \tag{10}$$

Notice that $3\log W - 1 > 0$ since $W \geq 3$. In each inequality we can subtract $3\log W$, and substitute the class constants: $H = 0$, $FL_1 = 2$. Furthermore, note that $3\log\zeta = 1$ and

$$3\log(1-\zeta) < 3\log 2^{-2} = -6. \tag{11}$$

After these substitutions and simplifications the only equations that need further examination are the following (obtained from (8) and (9)):

$$0 \geq 1 + 3\log\beta + 2 + \left(1 + 3\log\frac{1-\zeta}{\beta}\right)^+ \qquad \text{where } \beta \in \left[1-\zeta, \frac{1}{2}\right) \tag{12}$$

$$0 \geq 1 + 3\log\beta + 1 + \left(1 + 3\log\frac{1-\beta}{\beta}\right)^+ \qquad \text{where } \beta \in \left[\frac{1}{2}, 1-\frac{\zeta}{2}\right] \tag{13}$$

Inequality (12) is easy to prove if the positive part is 0, since $0 \geq 3+3\log\beta$ holds for $\beta < \frac{1}{2}$. If the positive part is greater than 0, then we can subtract $3\log\beta$, and we are left with $0 \geq 4 + 3\log(1-\zeta)$, which follows from our previous bound (11). Similarly, in inequality (13), if the positive part is greater than 0, then we can subtract $3\log\beta$, and we are left to prove $0 \geq 3+3\log(1-\beta)$, which follows from $\beta \geq \frac{1}{2}$. If the positive part is 0, then we need to prove $0 \geq 2+3\log\beta$. This holds if $\beta \leq \zeta^2$, which is implied by $\beta \leq 1 - \frac{\zeta}{2} < \zeta^2$.

Our inequalities for the classes $\mathbf{FL}_\alpha$:

$$3\log W + FL_\alpha \geq 1 + 3\log(\alpha W) + 1 \tag{14}$$

$$3\log W + FL_\alpha \geq 1 + 3\log(\beta W) + FL_{\frac{\alpha}{\beta}} \qquad \text{where } \beta \in \left(\frac{1}{2}, \zeta\right], \alpha \in \left(0, \frac{1}{2}\right] \tag{15}$$

$$3\log W + FL_\alpha \geq 1 + 3\log(\beta W) + FL_{\frac{1-\beta}{\beta}} \quad \text{where } \beta \in \left(\frac{1}{2}, 1-\frac{\zeta}{2}\right], \alpha \in \left(0, \frac{1}{2}\right] \tag{16}$$

$$3\log W + FL_\alpha \geq 1 + 3\log\left(\frac{1}{2}W\right) + M_1 \tag{17}$$

Inequalities (14) and (17) are straightforward to check. The rest of the inequalities after substitutions and subtracting $3\log W$:

$$1 \geq 1 + 3\log\beta + 1 + \left(1 + 3\log\frac{\alpha}{\beta}\right)^+ \qquad \text{where } \beta \in \left(\frac{1}{2}, \zeta\right], \alpha \in \left(0, \frac{1}{2}\right] \tag{18}$$

$$1 \geq 1 + 3\log\beta + 1 + \left(1 + 3\log\frac{1-\beta}{\beta}\right)^+ \qquad \text{where } \beta \in \left(\frac{1}{2}, 1-\frac{\zeta}{2}\right] \tag{19}$$

Inequality (19) is strictly weaker than inequality (13), which has been proven previously. For inequality (18), if $\frac{\alpha}{\beta} > \zeta$, we can cancel out $3\log\beta$, and we are left to prove $1 \geq 3 + 3\log\alpha$ which follows from $\alpha \leq \frac{1}{2}$. Otherwise the positive part is 0, so we need $1 \geq 2+3\log\beta$. This is equivalent to $\beta \leq \zeta$, which is exactly our upper bound on β.

Finally, we look at the inequalities for the classes of type $\mathbf{M}_\alpha$.

$$3\log W + M_\alpha \geq 1 + 3\log(\alpha W) + 1 \tag{20}$$

$$3\log W + M_\alpha \geq 1 + 3\log\left(\frac{1}{2}W\right) + M_1 \tag{21}$$

$$3\log W + M_\alpha \geq 1 + 3\log\left(\frac{1}{2}W\right) + FL_1 \tag{22}$$

$$3\log W + M_\alpha \geq 1 + 3\log(\beta W) + FL_{\frac{\alpha}{\beta}} \qquad \text{where } \beta \in \left(\frac{1}{2}, \frac{1+\alpha}{2}\right] \tag{23}$$

$$3\log W + M_\alpha \geq 1 + 3\log(\beta W) + M_{\frac{\alpha}{\beta}} \qquad \text{where } \beta \in \left(\frac{1}{2}, \frac{2}{3}\right],\ \alpha \in \left(0, \frac{1}{2}\right] \tag{24}$$

The first three inequalities are straightforward to check. We do our usual substitutions and simplifications on the other two inequalities.

$$2+(1+3\log\alpha)^+ \geq 2 + 3\log\beta + \left(1 + 3\log\frac{\alpha}{\beta}\right)^+ \qquad \text{where } \beta \in \left(\alpha, \frac{1+\alpha}{2}\right] \tag{25}$$

$$2 \geq 3 + 3\log\beta + \left(1 + 3\log\frac{\alpha}{\beta}\right)^+ \qquad \text{where } \beta \in \left(\frac{1}{2}, \frac{2}{3}\right],\ \alpha \in \left(0, \frac{1}{2}\right] \tag{26}$$

For inequality (25), we consider 3 cases. If $\alpha \geq \zeta$, then $\frac{\alpha}{\beta} > \zeta$, so we can omit both positive parts. We arrive at $3 + 3\log\alpha \geq 3 + 3\log\beta + 3\log\frac{\alpha}{\beta}$, which is in fact an equality. If $\alpha < \zeta$, but $\frac{\alpha}{\beta} \geq \zeta$, then we need to prove $2 \geq 3 + 3\log\beta + 3\log\frac{\alpha}{\beta}$. After crossing out $3\log\beta$ the inequality follows from $\alpha < \zeta$. Finally, if both positive parts are 0, then the inequality reduces to $2 \geq 2 + 3\log\beta$, which is implied by $\beta < 1$.

In inequality (26), if the positive part is 0, then we need $2 \geq 3 + 3\log\beta$, which is equivalent to $\beta \leq \zeta$, but this is satisfied since $\beta \leq \frac{2}{3} < \zeta$. Otherwise we can cancel out $3\log\beta$, and arrive at $2 \geq 4 + 3\log\alpha$, which follows from $\alpha \leq \frac{1}{2}$. □

7.3 Proof of Theorem 7.3 and Theorem 2.1

Proof of Theorem 7.3. First, we observe that the binary tree B_0 that is obtained using the optimal tree building method of Section 4 has equal or less height as the tree B obtained from the weight based cutting method described in section 6. Thus it is sufficient to prove that B has height at most $3\log n$.

Let $v_B \in B$ be a root of a HPT subtree $\mathcal{H}$ that is also a leaf of an LCT (we denote the LCT subtree by $\mathcal{L}$). Notice that the weight assigned to v_B in $\mathcal{L}$ is $\mathrm{des}(u_T)$ for the light child $u_T \in V(T)$ corresponding to that leaf. In $\mathcal{H}$, the total weight of the weight sequence assigned to v_B is $\mathrm{des}(u_0) + (\mathrm{des}(u_1) - \mathrm{des}(u_0)) + (\mathrm{des}(u_2) - \mathrm{des}(u_1)) + \cdots +$

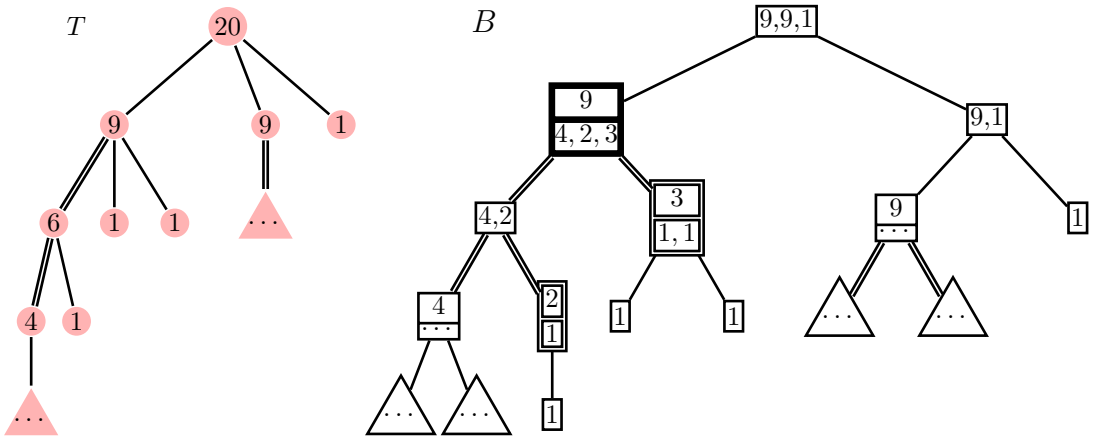

Figure 10: Left: original spanning tree with double lines indicating heavy edges. Unimportant subtrees are indicated with a small triangle. Right: the resulting binary tree, with the assigned weight sequences to each vertex. At HPT-LCT attaching points the top number corresponds to the leaf, the bottom sequence is the weight sequence assigned to the root.

$(\mathrm{des}(u_k) - \mathrm{des}(u_{k-1})) = \mathrm{des}(u_k)$ where $u_0, u_1, \ldots u_k$ are the vertices of the heavy path traversed upward, so the topmost vertex is $u_k = u_T$. Thus the total weight of the root weight sequence in $\mathcal{H}$ is equal to the weight assigned to the leaf in $\mathcal{L}$ (e.g. the double contour vertices in Figure 10).

Now let v_B be a root of an LCT subtree $\mathcal{L}$ which is also a leaf of a HPT denoted by $\mathcal{H}$. (E.g. the double contour vertices in Figure 10.) The weight assigned to v_B in $\mathcal{H}$ is either $\mathrm{des}(v_T)$ if the corresponding vertex $v_T \in T$ is the bottom vertex of a heavy path, or $\mathrm{des}(v_T) - \mathrm{des}(v'_T)$ if its heavy child is v'_T. The total weight assigned to v_B in $\mathcal{L}$ is the sum of the descendants of the light children of v_T, which is exactly $\mathrm{des}(v_T) - 1$ if v_T is the bottom vertex of its heavy path, or $\mathrm{des}(v_T) - \mathrm{des}(v'_T) - 1$ if its heavy child is v'_T.

To prove Theorem 7.3, we can invoke Theorems 7.1 and 7.2 going bottom up in the tree B. The conditions are straightforward to check for the HPT subtrees that have no LCT children in B.

For a HPT subtree $\mathcal{H}$, we need that $\mathrm{height}_B(l_1) \leq \max(3 \log W - 1, 0)$, where W is the weight assigned to this leaf in $\mathcal{H}$. Let v_T be the corresponding vertex in T. The first leaf corresponds to the bottom of the heavy path. If the leaf has no children, then its weight is 1, so the maximum is 0 and the condition is satisfied. Otherwise it has light children only, so the total weight assigned to the root of $\mathcal{L}$ is $W - 1$ by our

previous observation. Also note that the biggest weight has size at most $(W-1)/2$, otherwise there would be a heavy child, so v_T could not be the bottom vertex of its heavy path. By the stronger statement of our theorem, the root of the LCT has class parameter $\leq \frac{1}{2}$, so $\text{height}_B(l_1) \leq 3\log(W-1) - 1 < 3\log W - 1$.

For the rest of the leaves, we need $\text{height}_B(l_i) \leq 3\log W_i + 1$. This is straightforward since the total weight of l_i in L is $W_i - 1$, so $\text{height}_B(l_i) \leq 3\log(W_i - 1) + 1 < 3\log W_i + 1$ by induction.

For a LCT subtree $\mathcal{L}$, we need that $\text{height}_B(l_i) \leq 3\log W$ where W is the weight assigned to l_i in $\mathcal{L}$. The leaf can be a leaf of B, in which case it has weight 1 and the height is 0, or it has a child HPT, which has total weight W by our previous observation. By induction, we have $\text{height}_B(l_i) \leq 3\log W$ from the child HPT.

The topmost small ROBT of B is either a HPT or an LCT with parameter at most $\frac{1}{2}$. In the former case we get the upper bound $3\log n$ on the height of B, and in the latter, we get $3\log n - 1$. □

We can now turn to the proof of Theorem 2.1.

Proof of Theorem 2.1. Choose any spanning tree T of G, perform our embedding. Use $\text{code}(c_1(v))$ and $\text{code}(c_2(v))$ in Lemma 3.1 to create a single bit sequence $f(v)$. Now f with the distance function d is a guaranteed greedy embedding by Theorem 5.1. Since the length of $\text{code}(c_2(v))$ is at most the depth of B, it is at most $\lfloor 3\log n \rfloor$ by Theorem 7.3. So by Lemma 3.1 the length of $f(v)$ is at most

$$\lfloor 3\log n \rfloor + \lceil \log \lfloor 3\log n \rfloor \rceil + 1 \leq 3\log n + \lceil \log 3 + \log\log n \rceil + 1 < 3\log n + \log\log n + 3.59.$$

□

7.4 Sharpness and code length

We will show that Theorem 7.3 is sharp, i.e., there is a tree sequence for which the optimal binary tree described in Section 4 has depth $3\log n - O(1)$. This shows that even if our cutting lemmas do not guarantee the best possible binary tree depth, our worst-case theoretical guarantee is the same for the weight based version and the optimal version.

First, we define a sequence of rooted trees. Let T_0 be the rooted tree with a single vertex and no edges, and let T_n $(n = 1, 2, \dots)$ be the tree depicted on Figure 11: the root has three children, two of those children have one child each, whose subtrees are isomorphic to T_{n-1}.

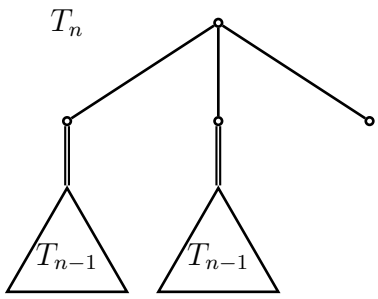

Figure 11: Construction of the tree T_i

As an example, we depicted the tree T_2 and also an optimal depth binary tree corresponding to this tree in Figure 12. Notice that for the tree T_i, at least one of the two non-leaf children of the root will be placed two levels below the root of the corresponding binary tree. The heavy edge will add another level, resulting in an optimal binary tree that has depth at least $3i$ for the tree T_i. (The subtrees of the binary tree that correspond to heavy paths are marked with double lines.)

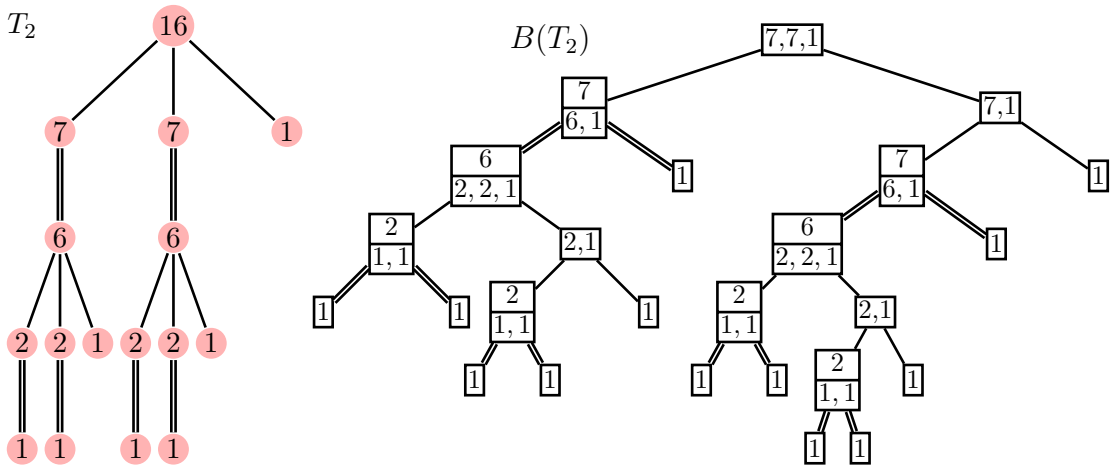

Figure 12: The tree T_2 and the corresponding binary tree $B(T_2)$.

Let $w(n)$ be the total number of vertices in the tree T_n. We know that $w(0) = 1$ and $w(n+1) = 2w(n) + 4$. Solving this recursion we get

$$w(n) = 2^{n+2} + 2^n - 4.$$

For the sharpness result we need to prove that the depth $3n$ is $3\log(w(n)) - O(1)$. But this is straightforward:

$$3\log(w(n)) = 3\log(5 \cdot 2^n - 4) = 3n + 3\log 5 + o(1) = 3n + O(1).$$

8 Conclusion and future work

In this paper, we have presented a graph embedding and a method to assign an address to each vertex. We defined an asymmetric 'distance' function on the addresses that can be used for greedy message delivery in this graph. The address length of each vertex is at most $3 \log n + \log \log n + 3.59$ bits. To prove this result, we have introduced an alternative weight based embedding, and used a two-stage induction, and finally demonstrated that the first method has the same worst-case performance as the weight-based one.

There are several questions worth considering for future work, we only mention two of them. The empirically demonstrated claim that the code length of the algorithm is far below its worst-case code length could be supported by a theorem, i.e., one might be able to show that the expected code length for a certain random graph class is much less than $3 \log n$. Alternatively, it might be possible to identify a class of graphs in which every member has a low-length embedding.

Finally, the most natural question to ask is whether it is possible to make a greedy embedding with shorter code length. What is the minimum constant c for which there is an embedding with code length $c \log n + o(\log n)$ for every n vertex graph?

References

[1] S. Alstrup, E. B. Halvorsen, and K. G. Larsen. Near-optimal labeling schemes for nearest common ancestors. *arXiv preprint arXiv:1312.4413*, 2013.

[2] D. Eppstein and M. Goodrich. Succinct greedy geometric routing using hyperbolic geometry. *Computers, IEEE Transactions on*, 60(11):1571–1580, 2011.

[3] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[4] S. Kisfaludi-Bak, T. Sebők, Z. Király, and I. Csabai. Guaranteed Milgram routing using near-optimal address lengths. *Submitted manuscript.*

[5] R. Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1902–1909, 2007.

[6] D. E. Knuth. Optimum binary search trees. *Acta informatica*, 1(1):14–25, 1971.

[7] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108. ACM, 2003.

[8] H. Zhang and S. Govindaiah. Greedy routing via embedding graphs onto semi-metric spaces. *Theoretical Computer Science*, 508(0):26 – 34, 2013. Frontiers of Algorithmics.