Two Results on Linear Embeddings of Complete Binary Trees

Marek Chrobak*
Department of Computer Science
University of California
Riverside

Wojciech Rytter†
Institute of Informatics
Warsaw University

October 4, 1996

Abstract

Given a binary tree T with n vertices, we want to embed T onto a given set A of n points on the line so as to minimize the total embedded edge length. Polynomial time algorithms for the two following special cases of this problem can be found in literature:

(1) When T is arbitrary but $A = \{1 \ldots n\}$;

(2) When T is a complete binary tree and A is arbitrary.

To the best of our knowledge, the complexity of the general problem is open. In this paper we deal with case (2). Bern, Karloff, Raghavan and Schieber presented an algorithm for this case that runs in time $O(n^{5.76})$ and uses $O(n^{3.2})$ space. They also considered the *naive embedding*, which maps the root r of T into the middle point a of A, and then embeds, recursively, the left and right subtrees of r to the left and right of a, respectively. This is equivalent to embedding T from left to right according to the inorder traversal. They prove that this naive algorithm approximates the optimal solution within the factor of 3.

The main result of this paper are: (i) the proof that the approximation ratio of this naive algorithm is exactly $\frac{7}{3}$, and (ii) a more efficient algorithm for computing minimum embeddings of complete binary trees. Our algorithm runs in time $\widetilde{O}(n^{1+\log 3}) = O(n^{2.59})$, and uses $\widetilde{O}(n)$ space, where $\widetilde{O}(f) = O(f \log^c n)$, for some constant $c > 0$.

*Supported by NSF grant CCR9112067.

†Supported by project ALTEC. This work was done while the author was visiting University of California, Riverside.

1 Introduction

Let T be a binary tree, and denote by V_T and E_T, respectively, the sets of vertices and edges of T. Let $n = |V_T|$ be the number of vertices of T. A one-to-one function $f : V_T \to \{1, 2, \ldots, n\}$ will be called a *linear embedding*, or simply *an embedding*. Given such a linear embedding f and a set of real numbers $A = \{a_1, \ldots, a_n\}$, we define the cost of f with respect to A as follows:

$$cost_A(f) = \sum_{(u,v) \in E_T} |a_{f(u)} - a_{f(v)}|.$$

We will write $cost(f)$ instead of $cost_A(f)$, when A is understood from context. A function f that minimizes $cost_A(f)$ will be called a *minimal linear embedding*, or *minimal embedding*, for short. The minimal embedding cost is $opt(A) = cost_A(f)$, for any minimal embedding f.

Given T and A, we would like to construct a minimal embedding of T into A. It is known that when the points in A are equally spaced, then the problem can be solved in polynomial time, see [Ch84, GK76, Sh79]. The fastest known algorithm for this problem, given by Chung [Ch84], runs in time $O(n^{1.6})$. The complexity of the general problem, when the points in A are arbitrary, remains open. (However, the problem is NP-hard if we allow arbitrary graphs on input, even if the points in A are equally spaced, see [ES75].)

Bern, Karloff and Raghavan and Schieber [BKRS89] considered the case when A is arbitrary, but T is assumed to be a complete binary tree. Formally, they studied the following optimization problem:

> MINIMAL EMBEDDING OF COMPLETE BINARY TREES: Given an arbitrary set A of points on the line, with $|A| = n = 2^{k+1} - 1$, find a minimal linear embedding of the depth-k complete binary tree into A.

Quite surprisingly, even in this special case the problem is quite nontrivial. They considered first the *naive embedding* Φ which maps vertices from T into A from left to right, according to the inorder traversal. In other words, it works as follows: Embed the root r of T into the middle point of A: $\Phi(r) = 2^{k-1}$. Then embed the left and right subtrees to the left and right of the root, using recursively the same method. It turns out that this method does not necessarily produce an optimal embedding. For example, the first embedding in Figure 1 is the naive embedding. The second embedding is not the naive one, but its cost is smaller that in the first one. Chung [Ch78] described minimal embeddings for complete binary trees and equally-spaced points. In fact, the minimal embedding for the unequally-spaced points in Figure 1 happens to be Chung's embedding.

Clearly, the naive embedding Φ can be computed in linear time. In fact, the naive algorithm does not depend on the numbers a_i, only on n. Embeddings that have this property will be called *oblivious*. By extending the embedding proposed by Chung in [Ch78] to arbitrary sets A we obtain another oblivious embedding that will be discussed in the last section.

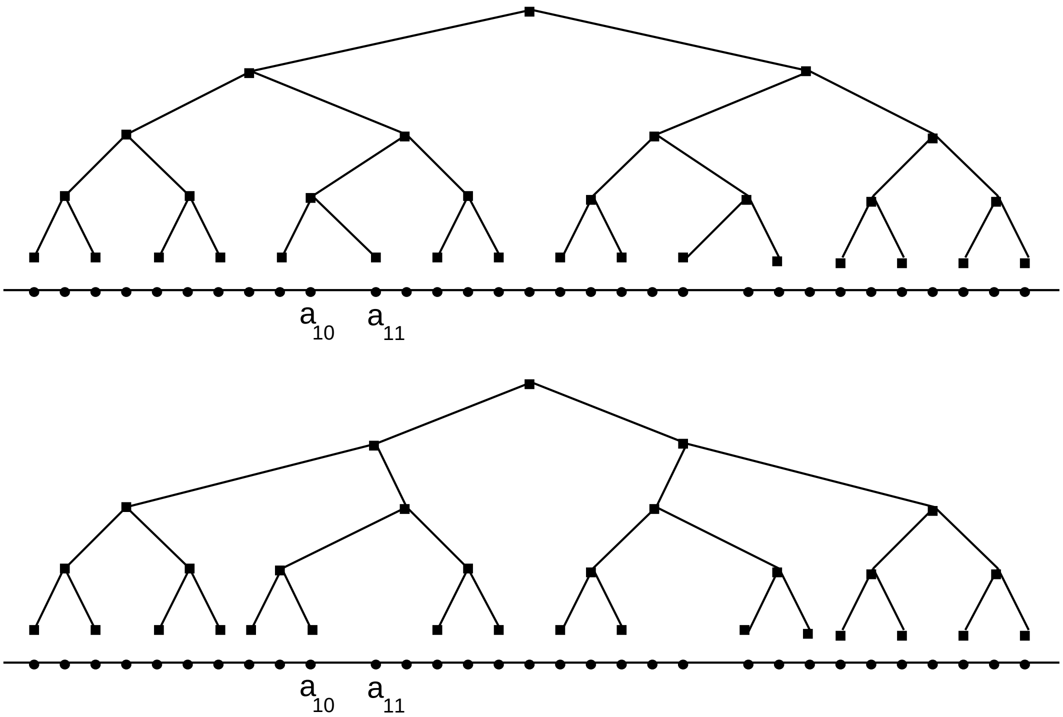

Figure 1: The naive embedding, and a minimal embedding of a complete binary tree. In the naive embedding 4 edges cross the interval $[a_{10}, a_{11}]$, in the optimal embedding only 2 edges cross this interval.

Bern *et al* [BKRS89] proved that the naive embedding has performance ratio at most 3. The above example shows that the ratio is at least 2, just let $a_{11} - a_{10}$ grow to infinity.

Their second result is a dynamic programming algorithm that computes a minimal linear embedding in time $O(n^{5.76})$ and $O(n^{3.2})$ space.

We improve both results from [BKRS89]. First we show that the worst case performance ratio (absolute and asymptotic) of the naive algorithm is 7/3. This means that (i) $cost_A(\Phi) \leq \frac{7}{3} opt(A)$, for each subsets A of the real line with $|A| = 2^{k+1} - 1$ for some k, and that (ii) for each $\epsilon > 0$ there is $n = 2^{k+1} - 1$, and a set A of n points on the line, such that $cost_A(\Phi) \geq (\frac{7}{3} - \epsilon) opt(A)$. In fact, this is true for arbitrarily large n.

The upper bound proof is based on number-theoretical properties of so-called *full representations* of integers (see Theorem 4.1) which, we believe, may be of independent interest.

Our second result is an algorithm which constructs a minimal embedding using less time and space than the one from [BKRS89]. Our algorithm runs in time $\widetilde{O}(n^{1+\log 3}) = O(n^{2.59})$, and uses $\widetilde{O}(n)$ space, where $\widetilde{O}(f) = O(f \log^c n)$, for some constant $c > 0$ [1].

[1] Unless otherwise specified, the logarithms are to base 2.

2 Preliminaries

For each $k \geq 0$, define the k-th *full number* to be $F_k = 2^{k+1} - 1$. Thus F_k is the size (number of nodes) of a complete binary tree of depth k. Recall that given a binary tree T, V_T and E_T denote, respectively, the sets of vertices and edges of T.

Let T be a complete binary tree of depth k. By r we will denote the root of T. The parent of a vertex $u \neq r$ is denoted by $father(u)$. The *depth* of a vertex u, denoted by $depth(u)$, is the length of the path (= number of edges) from root r to u. If $v = father(u)$, then $depth(u, v) = depth(v)$ is called the *depth* of edge (u, v).

If u is not a leaf, then $left(u)$ and $right(u)$ denote the left nad right children of u, respectively. Given any $i \in \{0, \ldots, k - depth(u)\}$, we define $left^i(u)$ inductively as follows: $left^0(u) = u$, and $left^{i+1}(u) = left(left^i(u))$. In a similar manner we define $right^i(u)$.

For each node u, $T(u)$ denotes the subtree rooted at u, and $Anc(u)$ is the set of the ancestors of u, including u. Thus $v \in Anc(u)$ iff $u \in T(v)$. Two nodes u, v are called *nonancestral* if $u \notin T(v)$ and $v \notin T(u)$.

Let $A = \{a_1, \ldots, a_n\}$, with $n = F_k$, where $a_1 < \ldots < a_n$. Let f be an arbitrary embedding of T into A. For a set $U \subseteq V_T$, $f(U)$ is the image of U under f. Define $min_f(u) = min\{f(T(u))\}$, and, similarly, $max_f(u) = max\{f(T(u))\}$.

Given $i \in \{1, \ldots, F_k - 1\}$ and $v = father(u)$, we say that (u, v) is a *cross edge* at i if either $f(v) \leq i < f(u)$ or $f(u) \leq i < f(v)$. If $f(u) < f(v)$ then we say that (u, v) has *left orientation* in f, otherwise it has *right orientation*. The *cut of f at i*, denoted by $CUT_i(f)$, is the set of all cross edges at i.

Observation 2.1 *The cost of an embedding f can be expressed in terms of the cardinalities of its cuts as follows:*

$$cost(f) = \sum_{i=1}^{n-1} (a_{i+1} - a_i) \, |CUT_i(f)| \, .$$

Let Φ denote the naive embedding function, as defined in the introduction. Figure 1 shows that Φ is not a minimal embedding for $k = 4$. Denoting the second embedding given in Figure 1 by f we have $|CUT_{10}(f)| = 2$ and $|CUT_{10}(\Phi)| = 4$.

Given a $u \in V_T$, we will refer to $\Phi(u)$ as the *home location* of u. The interval $[min_\Phi(u), max_\Phi(u)]$ is called the *home interval* of $T(u)$.

Given two nodes $u, v \in V_T$, we write $u \prec v$ if one of the following conditions holds:

(ord1) $u = left^i(v)$ or $v = right^i(u)$, for some i.

(ord2) there is a node w such that $u \in T(left(w))$ and $v \in T(right(w))$.

The relation $\prec$ is a partial order on V_T. This follows from the fact that the naive embedding function is a linear extension of $\prec$.

Transformation of embeddings. Given $u, v \in V_T$ and two embeddings f, g, we write $g = \mathit{Swap}_{u,v}(f)$, if g is obtained from f by exchanging the locations of u and v. More formally, $g(u) = f(v)$, $g(v) = f(u)$ and $g(x) = f(x)$ for $x \in V_T - \{u, v\}$.

Another transformation exchanges whole subtrees. If $u, v \in V_T$ have the same depth, then we write $g = \mathit{TreeSwap}_{u,v}(f)$, if g is obtained from f by exchanging the location of each $x \in T(u)$ with the node $y \in T(v)$ corresponding to x. Let $u' = \mathit{father}(u)$ and $v' = \mathit{father}(v)$. Then *TreeSwap* affects only the costs of edges (u, u') and (v, v'). More specifically,

$$\mathit{cost}(g) \;=\; \mathit{cost}(f) + |a_{f(u')} - a_{f(v)}| + |a_{f(v')} - a_{f(u)}| - |a_{f(u')} - a_{f(u)}| - |a_{f(v')} - a_{f(v)}|.$$

This directly implies the following observation.

Observation 2.2 *Let $u, v \in V_T$ be two nodes with $\mathit{depth}(u) = \mathit{depth}(v)$ and $u' = \mathit{father}(u) \neq v' = \mathit{father}(v)$. If the edges (u, u'), (v, v') cross in f, that is $f(v'), f(u) < f(u'), f(v)$, then $\mathit{TreeSwap}_{u,v}$ produces a better embedding: $\mathit{cost}(g) < \mathit{cost}(f)$ for $g = \mathit{TreeSwap}_{u,v}(f)$.*

3 Normal Embeddings

In this section we introduce a special type of linear embeddings called *normal embeddings*. Normal embeddings are easier to deal with, and they are sufficiently general in the sense that each embedding can be transformed into a normal one without increasing cost. Several useful facts about normal embeddings were presented in [BKRS89]; we sketch the proofs for completeness.

Fix a full number $n = F_k$, and let T be a complete binary tree of depth k. Let $A = \{a_1, \ldots, a_n\}$ be a set of real numbers, where $a_1 < a_2 < \cdots < a_n$.

We define an embedding f to be *normal*, if f is a linear extension of $\prec$, that is, for all $u, v \in V_T$, if $u \prec v$ then $f(u) < f(v)$. Thus, in particular, Φ is a normal embedding.

Theorem 3.1 *There exists a minimal embedding which is normal.*

Proof: We prove the theorem in two steps.

Claim A: There is a minimal embedding g such that $g(\mathit{left}(u)) < g(u) < g(\mathit{right}(u))$ for each $u \in V_T$ which is not a leaf.

Proof of Claim A: Pick an arbitrary minimal embedding f. Without loss of generality we can assume that $f(\mathit{left}(u)) < f(\mathit{right}(u))$ for all $u \in V_T$, since otherwise we can consider a linear embedding $f' = \mathit{TreeSwap}_{x,y}(f)$ for $x = \mathit{left}(u)$ and $y = \mathit{right}(u)$.

Let us call a non-leaf node u *left-balanced* in f, if $f(\mathit{left}(u)) < f(u)$. The term *right-balanced* is defined similarly. A node is balanced if it is both left and right balanced. We extend this definition to leaves, by assuming that all leaves are balanced.

We show how to reduce the number of unbalanced nodes without increasing the cost.

Let $u \in V_T$ be the leftmost node in f (with smallest value of $f(u)$) which is not right-balanced, that is $f(\mathit{left}(u)) < f(\mathit{right}(u)) < f(u)$. Denote $x = \mathit{left}(u)$, $y = \mathit{right}(u)$, and $f(u) = i$.

Consider first the case when $f(y) = i-1$. By the choice of u, y must be right-balanced. Define $f' = \mathit{Swap}_{u,y} f$. Then $\mathit{cost}(f') \le \mathit{cost}(f)$ and f' has one less unbalanced node.

Suppose now that $f(y) < i-1$. Let z be the node mapped into $i-1$, that is $f(z) = i-1$. By the choice of u, z is right-balanced, that is either z is a leaf or $f(\mathit{right}(z)) > f(z)$. Define $f' = \mathit{Swap}_{u,z}(f)$. Then $\mathit{cost}(f') \le \mathit{cost}(f)$, decreasing the cost of edge (u, y). In f', u is again the leftmost node which is not right-balanced, and $f'(u) < f(u)$. Also, no new unbalanced nodes have been introduced. Thus after applying the same procedure $f(u) - f(y)$ times we will restore the balance at u and decrease the number of unbalanced nodes.

Claim B: Assume that the g is a minimal embedding that satisfies Claim A. Then g is normal.

Proof of Claim B: Embedding g has the property that, for all $u, v \in V_T$, if either $u = \mathit{left}^i(v)$ or $v = \mathit{right}^i(u)$, for some i, then $g(u) < g(v)$. It is enough to prove that g preserves the order $\prec$ of nonancestral nodes.

The proof is by contradiction. Assume that $u, v \in V_T$, are nonancestral and that they violate condition (ord2), that is $u \prec v$ but $g(u) > g(v)$. Without loss of generality we can assume that $\mathit{depth}(u) = \mathit{depth}(v)$. For otherwise, suppose that $\mathit{depth}(v) < \mathit{depth}(u)$. Pick $v_1 = \mathit{left}^i(v)$, such that $\mathit{depth}(v_1) = \mathit{depth}(u)$. Then $u \prec v'$, and $g(u) > g(v) > g(v_1)$ because g satisfies Claim A.

Pick two nodes u, v with minimal $\mathit{depth}(u) = \mathit{depth}(v)$ that violate (ord2). Let $u' = \mathit{father}(u)$ and $v' = \mathit{father}(v)$. Since g satisfies Claim A, $u' \neq v'$. By the choice of u, v, we have $u' \prec v'$ and $g(u') < g(v')$. Thus the edges (u, u') and (v, v') cross each other, as defined in Observation 2.2, and consequently the embedding $g' = \mathit{TreeSwap}_{u,v}(g)$ has smaller cost – a contradiction with the optimality of g. $\Box$

Properties of normal embeddings. Now we will prove some properties of normal embeddings that will be used in our algorithm.

Observation 3.1 *If f is normal then, for each nonancestral pair of nodes $u, v \in V_T$, the sets $f(T(u))$ and $f(T(v))$ do not interleave, that is $f(u) < f(v)$ implies $\mathit{max}_f(u) < \mathit{min}_f(v)$.*

Lemma 3.1 *Suppose that f is a normal embedding. Then, for each $i \in \{1, \ldots, n-1\}$, all edges in $CUT_i(f)$ have different depth.*

Proof: Let $(x, y), (u, v) \in CUT_i(f)$, with $x = \mathit{father}(y)$, $u = \mathit{father}(v)$ and $\mathit{depth}(x) = \mathit{depth}(u)$. Without loss of generality, $x \prec u$. The case $x = u$ is impossible, by the normality of f. If $x \neq u$, then, since u, x have the same depth, they must be nonancestral, which implies that $x, y \prec u, v$. Thus, by the normality of f, edges (x,y), (u,v) cannot be both in $CUT_i(f)$. $\square$

Lemma 3.2 *Let f be a normal embedding and $v \in V_T$. Then*

$$|min_f(v) - min_\Phi(v)|,\ |max_f(v) - max_\Phi(v)| \ \leq\ depth(v).$$

Proof: We define sets

$$\begin{aligned} L &= \{u \in V_T : u \prec v\} - T(v) - Anc(v) \\ R &= \{u \in V_T : v \prec u\} - T(v) - Anc(v) \end{aligned}$$

Then f embeds the nodes in L to the left of the nodes in $T(v)$ and, similarly, the nodes in R to the right of the nodes in $T(v)$. Since the naive embedding has the same property, only the nodes in $Anc(v) - \{v\}$ can contribute to the differences $min_f(v) - min_\Phi(v)$ and $max_f(v) - max_\Phi(v)$, and each contributes at most one. The lemma follows, since $|Anc(v) - \{v\}| = depth(v)$. $\square$

Lemma 3.1 was used in [BKRS89] to design a dynamic programming algorithm for constructing minimal embeddings. Our method also is also based on dynamic programming, although the way we divide an instance into sub-instances is different than the one in [BKRS89]. The main idea for improving the efficiency is the observation that, by Lemma 3.2, it is sufficient to consider only $\widetilde{O}(n)$ "target" subintervals $[min_f(v), max_f(v)]$ that approximate the home interval of v, where f is a normal embedding.

4 Full Representations of Numbers

Recall that by F_k we denote the kth full number, that is $F_k = 2^{k+1} - 1$. Note also that $F_{k+1} = 2F_k + 1$. In this section we will investigate so-called full representations of integers, in which each x is represented by a sum of full numbers or their negations. Full representations are not unique; an integer may have several different representations.

In the next section we will show that full representations correspond to embeddings of complete binary trees. The naive embedding Φ defines a *naive representation*. There is also an optimal representation that corresponds to the optimal embedding. In Theorem 4.1 of this section we will show that the size (as defined below) of the naive embedding is at most $\frac{7}{3}$ times the size of the optimal embedding – a result that we believe is interesting of its own. It also constitutes the main step towards proving the upper bound on the performance of the naive algorithm.

Definition of full representations. Given an integer α let $\bar{\alpha} = -\alpha$. Let $x \geq 0$. If $x = \sum_{i=0}^{p} \sigma_i F_i$, where $\sigma_i \in \{\bar{1}, 0, 1\}$ for $i = 0, \ldots, p$, then the sequence $\overline{\sigma} = \sigma_p \sigma_{p-1} \ldots \sigma_0$ is called a *full representation* of x. The *size* of this representation $\overline{\sigma}$ is defined as $\|\overline{\sigma}\| = \sum_{i=1}^{p} |\sigma_i|$. Let $\omega(x)$ denote the size of the *optimal representation* of x, that is $\omega(x) = min_{\overline{\sigma}} \|\overline{\sigma}\|$, where the minimum is over all full representations $\overline{\sigma}$ of x.

The *naive representation* of x, denoted $\overline{\eta}(x)$, is defined by induction on x, as follows: First, $\overline{\eta}(0) = 0$ and $\overline{\eta}(F_p) = 10^{p-1}$ for each $p \geq 1$. If $x > 1$ is not a full number, then let p be the smallest integer such that $x < F_p$, and let $\overline{\eta}(z) = \eta_q \eta_{q-1} \dots \eta_0$ be the naive representation of $z = F_p - x$ (where $\eta_q = 1$). Then $\overline{\eta}(x) = 10^{p-q-1} \bar{\eta}_q \dots \bar{\eta}_0$. In other words, $\overline{\eta}(x)$ consists of a 1 at position p, a run of $p - q - 1$ 0's, and the naive representation of z with 1's replaced by $\bar{1}$'s, and $\bar{1}$'s replaced by 1's.

Observation 4.1 *Given $x > 0$, the naive representation $\overline{\eta}(x)$ of x can be computed as follows: Let $x = \alpha_k \alpha_{k-1} \dots \alpha_1$ be the binary representation of x, with $\alpha_k = 1$. Going from left to right, replace each 1 which is not the first in a run of ones by 0, and replace each 0 which is the first in a run of zeros by $\bar{1}$.*

For example, the integer written in binary as 1110001011100010_2 has the naive representation $100\bar{1}001\bar{1}100\bar{1}001\bar{1}$. The number $85 = 1010101_2$ has the naive representation $1\bar{1}1\bar{1}1\bar{1}1$.

By $\phi(x)$ we will denote the size of the naive representation of x, that is $\phi(x) = \sum_{i=0}^{p} |\eta_i|$, where $\eta_p \dots \eta_0 = \overline{\eta}(x)$ is the naive representation of x. We also introduce the function $\phi^+(x) = max_{y \leq x} \phi(y)$.

Observation 4.2 *Given $x \geq 0$, the size of the naive representation of x, $\phi(x)$, can be defined recursively as follows: $\phi(0) = 0$, and $\phi(x) = 1 + \phi(F_p - x)$, where F_p is the smallest full number $\geq x$.*

The naive representation of 85, $85 = 1\bar{1}1\bar{1}1\bar{1}1$ has size 7, that is $\phi(85) = 7$. However, $\omega(85) = 3$, since 85 has a full representation 101100 of size 3, and it does not have a full representation of size 2.

The first 16 values of $\phi(x)$ and $\phi^+(x)$, as well as $\omega(x)$, are given in the table below. Note that $\phi(10) \neq \omega(10)$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi(x)$	0	1	2	1	2	3	2	1	2	3	4	3	2	3	2	1
$\phi^+(x)$	0	1	2	2	2	3	3	3	3	3	4	4	4	4	4	4
$\omega(x)$	0	1	2	1	2	3	2	1	2	3	2	3	2	3	2	1

Lemma 4.1 $2x + \phi^+(x-1) - 1 \leq \frac{7}{3}x$, *for all $x \geq 1$.*

Proof: We prove first the following claim:

Claim A: $\phi^+(x) \leq \phi^+(x-3) + 1$, for all $x \geq 4$.

Claim A holds for $4 \leq x \leq F_2 = 7$, by inspection. Assume that $x \in (F_a, F_{a+1}]$, for some $a \geq 2$. We have $\phi(y) = 1 + \phi(F_{a+1} - y) \leq 1 + \phi^+(F_a)$ for all $y \in (F_a, F_{a+1}]$. This implies that $\phi^+(x) =\leq 1 + \phi^+(F_a)$. Since $\phi^+(F_a) \geq 3$, $\phi(F_a) = 1$, and $\phi(F_a - 1) = 2$, we obtain $\phi^+(x) \leq 1 + \phi^+(F_a - 2) \leq 1 + \phi^+(x-3)$.

Now we prove the lemma, by induction. For $x = 1, 2, 3, 4$, the left-hand side is 1, 4, 7, 9, while the right-hand side is $\frac{7}{3}$, $\frac{14}{3}$, 7, and $\frac{28}{3}$, respectively. For $x \geq 5$, applying Claim A and the inductive hypothesis, we get

$$2x + \phi^+(x-1) - 1 \;\leq\; 2x + \phi^+(x-4) \;=\; [2(x-3) + \phi^+(x-4) - 1] + 7 \;\leq\; \tfrac{7}{3}(x-3) + 7 \;=\; \tfrac{7}{3}x,$$

completing the proof of the lemma. □

Lemma 4.2 *Let $a \geq 0$. Then*

(i) *If $0 \leq \alpha \leq F_a$, then $\phi(F_a - \alpha) \leq 1 + \phi(\alpha)$.*

(ii) *If $1 \leq \beta \leq F_a + 1$, then $\phi(F_a + \beta) \leq 2 + \phi(\beta - 1)$.*

(iii) *If $1 \leq \gamma$, then $\phi(F_a + \gamma) \leq 2 + \phi^+(\gamma - 1)$.*

(iv) *If $\delta \leq F_a < \delta + \xi$, then $\phi(F_a + \delta + \xi) \leq 2 + \phi^+(\xi - 1)$.*

Proof: (i) For $a = 0$ the inequality is obvious, so let us assume that $a \geq 1$. If $\alpha \geq F_{a-1} + 1$, then $\phi(\alpha) = 1 + \phi(F_a - \alpha)$. If $\alpha \leq F_{a-1}$, then $F_a - \alpha > F_{a-1}$, and thus $\phi(F_a - \alpha) = 1 + \phi(\alpha)$.

(ii) If $\beta = F_a + 1$ then $\phi(F_a + \beta) = \phi(F_{a+1}) = 1$, and (ii) is obvious. If $1 \leq \beta \leq F_a$ then, using (i) with $\alpha = \beta - 1$, we have

$$\phi(F_a + \beta) \;=\; 1 + \phi(F_{a+1} - F_a - \beta) \;=\; 1 + \phi(F_a - \beta + 1) \;\leq\; 2 + \phi(\beta - 1)\,.$$

(iii) Let $b \geq a$ be such that $F_b \leq F_a + \gamma < F_{b+1}$. We can assume that $F_b < F_a + \gamma$, since for $F_b = F_a + \gamma$ the inequality (iii) is obvious. Take $\gamma' = F_a + \gamma - F_b$. Then $1 \leq \gamma' \leq \gamma$ and $\gamma' < F_{b+1} - F_b = F_b + 1$. Thus, using (ii), we get

$$\phi(F_a + \gamma) \;=\; \phi(F_b + \gamma') \;\leq\; 2 + \phi(\gamma' - 1) \;\leq\; 2 + \phi^+(\gamma - 1)\,.$$

(iv) The proof is similar to (iii). Let b be such that $F_b \leq F_a + \delta + \xi < F_{b+1}$. Since $\delta + \xi \geq F_a + 1$, we have $b > a$. Without loss of generality we can assume that $F_b < F_a + \delta + \xi$, since for $F_b = F_a + \delta + \xi$ the inequality (iv) is obvious. Take $\xi' = F_a + \delta + \xi - F_b$. Then $1 \leq \xi' < F_{b+1} - F_b = F_b + 1$ and $\xi' \leq F_a + \delta + \xi - F_{a+1} = \delta + \xi - F_a - 1 < \xi$. Therefore, using (ii), we get

$$\phi(F_a + \delta + \xi) \;=\; \phi(F_b + \xi') \;\leq\; 2 + \phi(\xi' - 1) \;\leq\; 2 + \phi^+(\xi - 1)\,,$$

completing the proof. □

Theorem 4.1 *$\phi(x) \leq \frac{7}{3}\omega(x)$, for each $x \geq 0$.*

Proof: Write $x = \sum_{i=0}^{p-1} \sigma_i F_{t_i}$, where $\sigma_i \in \{-1, 1\}$ for $i = 0, ..., p-1$, and $t_i > t_{i-1}$ for $i = 1, ..., p-1$. It is sufficient to prove the following claim.

Claim A: $\phi(x) \leq \frac{7}{3}p$.

If $p = 1$ then $\phi(F_{t_0}) = 1$. If $p = 2$, then by Lemma 4.2 Part (ii), $\phi(F_{t_1} + F_{t_0}) \leq 2 + \phi(F_{t_0} - 1) \leq 4$. Thus we can assume that $p \geq 3$. For $j = 0, \ldots, p-1$ define $x_j = |\sum_{i=0}^{j} \sigma_i F_{t_i}|$. In particular, $x_{p-1} = x$. Note that the sign of $\sum_{i=0}^{j} \sigma_i F_{t_i}$ depends only on the sign σ_j of the highest term. Also, $x_j = F_{t_j} \pm x_{j-1}$ for $j = 1, \ldots, p-1$.

Given λ, let $s(\lambda) = 0$ if $\lambda \leq 0$, and $s(\lambda) = 1$ if $\lambda > 0$. We introduce the following condition, for $j = p-1, p-2, \ldots, 0$.

COND (j): There exists $\lambda_j \geq -x_j$ such that $-p + 1 + j \leq \lambda_j \leq p - j$ and

$$\phi(x) \;\leq\; 2(p-1-j) - s(\lambda_j) + \phi(x_j + \lambda_j)\,.$$

Condition COND (j) doesn't seem to have any simple, intuitive interpretation; it is merely a technical tool to make the induction work.

COND $(p-1)$ holds trivially with $\lambda_{p-1} = 0$. Claim A follows by induction from Claim B and Claim C below.

Claim B: If COND (0) then $\phi(x) \leq \frac{7}{3}p$.

Claim C: If COND (j), for $j > 0$, then either COND $(j-1)$ or $\phi(x) \leq \frac{7}{3}p$.

Proof of Claim B: We have $\phi(x) \leq 2(p-1) - s(\lambda_0) + \phi(F_{t_0} + \lambda_0)$. If $\lambda_0 \leq 0$, then, by Lemma 4.2 Part (i) with $\alpha = -\lambda_0$, and by Lemma 4.1, we have

$$\phi(x) \;\leq\; 2(p-1) + \phi(F_{t_0} + \lambda_0) \;\leq\; 2p + \phi(-\lambda_0) - 1 \;\leq\; 2p + \phi^+(p-1) - 1 \;\leq\; \tfrac{7}{3}p.$$

If $\lambda_0 > 0$, then, by Lemma 4.2 Part (iii),

$$\phi(x) \;\leq\; 2(p-1) - 1 + \phi(F_{t_0} + \lambda_0) \;\leq\; 2p + \phi^+(\lambda_0 - 1) - 1 \;\leq\; 2p + \phi^+(p-1) - 1 \;\leq\; \tfrac{7}{3}p.$$

Proof of Claim C: We consider several cases.

Case 1: $x_j = F_{t_j} + x_{j-1}$. We have three subcases.

Case 1.1: $x_{j-1} + \lambda_j \leq 0$ (and thus $\lambda_j < 0$). Using Lemma 4.2 Part (i), with $\alpha = -x_{j-1} - \lambda_j$, and Lemma 4.1, we obtain

$$\begin{aligned}\phi(x) &\leq 2(p-1-j) + \phi\Big(F_{t_j} + x_{j-1} + \lambda_j\Big) \\ &\leq 2(p-1-j) + 1 + \phi(-x_{j-1} - \lambda_j) \;\leq\; 2p + \phi^+(p-1) - 1 \;\leq\; \tfrac{7}{3}p.\end{aligned}$$

Case 1.2: $1 \leq x_{j-1} + \lambda_j \leq F_{t_j}$. Using Lemma 4.2 Part (ii), with $\beta = x_{j-1} + \lambda_j$, we obtain

$$\begin{aligned}\phi(x) &\leq 2(p-1-j) - s(\lambda_j) + \phi\Big(F_{t_j} + x_{j-1} + \lambda_j\Big) \;\leq\; 2(p-j) - s(\lambda_j) + \phi(x_{j-1} + \lambda_j - 1) \\ &\leq 2[p-1-(j-1)] - s(\lambda_{j-1}) + \phi(x_{j-1} + \lambda_{j-1})\end{aligned}$$

for $\lambda_{j-1} = \lambda_j - 1$, since $s(\lambda - 1) \leq s(\lambda)$ for all λ. Note also that $\lambda_{j-1} + x_{j-1} \geq 0$ and $-p + j \leq \lambda_{j-1} \leq p - j + 1$. Thus COND $(j-1)$ is true.

Case 1.3: $x_{j-1} + \lambda_j \geq F_{t_j} + 1$ (and thus $\lambda_j > 0$). Using Lemma 4.2 Part (iv), with $\delta = x_{j-1}$ and $\xi = \lambda_j$, and Lemma 4.1, we obtain

$$\begin{aligned}\phi(x) &\leq 2(p-1-j) - 1 + \phi\Big(F_{t_j} + x_{j-1} + \lambda_j\Big) \\ &\leq 2(p-j) - 1 + \phi^+(\lambda_j - 1) \ < \ 2p + \phi^+(p-1) - 1 \ \leq \ \tfrac{7}{3}p.\end{aligned}$$

Case 2: $x_j = F_{t_j} - x_{j-1}$. We have two subcases:

Case 2.1: $-x_{j-1} + \lambda_j \leq 0$. Using Lemma 4.2 Part (i), with $\alpha = x_{j-1} - \lambda_j$, we obtain

$$\begin{aligned}\phi(x) &\leq 2(p-1-j) - s(\lambda_j) + \phi\Big(F_{t_j} - x_{j-1} + \lambda_j\Big) \ \leq \ 2(p-j) - s(\lambda_j) - 1 + \phi(x_{j-1} - \lambda_j) \\ &\leq 2[p-1-(j-1)] - s(\lambda_{j-1}) + \phi(x_{j-1} + \lambda_{j-1})\end{aligned}$$

for $\lambda_{j-1} = -\lambda_j$, since $s(\lambda) + 1 \geq s(-\lambda)$ for all λ. Note also that $x_{j-1} + \lambda_{j-1} \geq 0$ and $-p + j \leq \lambda_{j-1} \leq p - j + 1$. Thus COND $(j-1)$ is true.

Case 2.2: $-x_{j-1} + \lambda_j \geq 1$ (and thus $\lambda_j \geq 1$). Using Lemma 4.2 Part (iii) with $\gamma = -x_{j-1} + \lambda_j$, and Lemma 4.1, we obtain

$$\begin{aligned}\phi(x) &\leq 2(p-1-j) - 1 + \phi\Big(F_{t_j} - x_{j-1} + \lambda_j\Big) \ \leq \ 2(p-j) - 1 + \phi^+(-x_{j-1} + \lambda_j - 1) \\ &< 2p + \phi^+(p-1) - 1 \ \leq \ \tfrac{7}{3}p.\end{aligned}$$

This completes the proof of the lemma. □

5 The Analysis of the Naive Algorithm

Let T be the complete binary tree of depth k and $A = \{a_1, \ldots, a_n\}$, where $n = F_k$, be a fixed set of real numbers. We assume that $a_1 < \ldots < a_n$. As we shall see, any normal optimal embedding f of T determines a full representation of all numbers $0, 1, \ldots, F_{k-1}$.

Induced representations. Let f be a normal embedding. Fix $x \in \{0, 1, \ldots, F_{k-1}\}$. For $d = 0, ..., k-1$, let $\sigma_d = \sigma_d(f)$ be equal 1,0 or $\bar{1}$, depending on whether $CUT_x(f)$ has an edge of depth $k-d-1$ which is left-oriented in f, does not have an edge of depth $k-d-1$, or has an edge of depth $k-d-1$ which is right-oriented. The following lemmas were given in [BKRS89]; we sketch the proofs for completeness.

Lemma 5.1 *Let f, x, and $\overline{\sigma} = \sigma_{k-1} \ldots \sigma_0$ be as defined above. Then $\overline{\sigma}$ is a full representation of x. Consequently, we have $|CUT_x(f)| \geq \omega(x)$.*

Proof: (Sketch) Let $v = father(u)$. If $(u, v) \in CUT_x(f)$ is leftward then, if the whole tree $T(u)$ was embedded to the left of x (inclusively) it would contribute F_{k-d-1}, for $d = depth(v)$, points to the set $\{1, ..., x\}$. However, some branches of $T(u)$ may cross back to the right of x, and we need to subtract the sizes of the corresponding subtrees, and so on. □

Based on the above lemma, we can refer to sequence $\overline{\sigma}$ defined above as the *full representation of x induced by f*. Now we show that the naive embedding induces naive representations.

Lemma 5.2 *If $\overline{\sigma} = \sigma_{k-1}\ldots\sigma_0$ is the full representation of x induced by the naive embedding Φ, then $\overline{\sigma} = \overline{\eta}(x)$, the naive representation of x.*

Proof: (Sketch) The proof is by induction on x. If $x = 0, 1$ then the lemma holds. Assume that $x > 1$. Pick smallest p such that $x \leq F_p$, and let $v = left^{k-p}(r)$. Since $x \geq 2$, we have $p \geq 1$. Also, by the choice of p, $x \geq F_{i-p} + 1$.

Suppose first that $x = F_{p-1}+1$. In this case we have $x = \Phi(v)$, and therefore $\overline{\sigma} = 110^{p-1} = \overline{\eta}(x)$, as claimed.

Thus we can assume that $x > F_{p-1} + 1$. In this case we have $x \in \Phi(right(v))$. Let $z = F_p - x$. Since $z < x$, the naive embedding induces the naive representation $\overline{\eta}(z) = \eta_q\eta_{q-1}\ldots\eta_0$ of z. By the symmetry of tree $T(v)$, cut $CUT_x(\Phi)$ can be obtained from $CUT_z(\Phi)$ by reversing the direction of all edges and adding the edge $(v, father(v))$. But, in terms of representations, this is equivalent to negating all η_i, and appending 10^{p-q-1}, which yields the naive representation of x. □

As we show in the lemma below, the analysis of the naive algorithm can be reduced to the analysis of naive representations of numbers.

Lemma 5.3 *Let $C > 0$. The following two statements are equivalent:*

(i) *The naive algorithm has performance ratio C.*

(ii) *$\phi(x) \leq C \cdot \omega(x)$, for each integer $x \geq 0$.*

Proof: (i) $\Rightarrow$ (ii). Given x, pick k large enough so that $x \leq F_{k-1}$, and let $n = F_k$. Choose a multiset A such that $a_1 = \ldots = a_x = 0$, and $1 = a_{x+1} = \ldots = a_n$. For this A, for every embedding f, we have $\|\overline{\sigma}\| = cost_A(f)$, where $\overline{\sigma}$ is the full representation of x induced by f. This implies (ii).

One problem with the above argument is that A is a multiset. But this is easy to fix: pick a small $\epsilon > 0$, put $a_1, \ldots, a_x$ within ϵ of 0, and $a_{x+1}, \ldots, a_n$ within ϵ of 1, and let $\epsilon \to 0$.

(ii) $\Rightarrow$ (i). Let $n = F_k$, $|A| = n$, and let f be an arbitrary embedding of a complete binary tree T of depth k. By symmetry, without loss of generality we can assume that $cost(f) \geq 2 \cdot \sum_{i=1}^{F_{k-1}} (a_{i+1} - a_i) CUT_i(f)$. By the left $\leftrightarrow$ right symmetry of T, Lemma 5.1 and Lemma 5.2, we obtain

$$\begin{aligned} cost(\Phi) &= 2 \cdot \sum_{i=1}^{F_{k-1}} (a_{i+1} - a_i)\,|CUT_i(\Phi)| \;=\; 2 \cdot \sum_{i=1}^{F_{k-1}} (a_{i+1} - a_i)\phi(i) \;\leq\; 2C \cdot \sum_{i=1}^{F_{k-1}} (a_{i+1} - a_i)\omega(i) \\ &\leq\; 2C \cdot \sum_{i=1}^{F_{k-1}} (a_{i+1} - a_i)\,|CUT_i(f)| \;\leq\; C \cdot cost(f), \end{aligned}$$

completing the proof. □

Theorem 5.1 *The performance ratio of the naive algorithm is $\frac{7}{3}$. More specifically:*

(a) *for each set A such that $|A|$ is a full number, we have $cost_A(\Phi) \le \frac{7}{3} opt(A)$.*

(b) *For every ϵ there is a full number n and a set A of n points on the line such that $cost_A(\Phi) \ge (\frac{7}{3} - \epsilon) opt(A)$. In fact, there are infinitely many numbers n with this property.*

Proof: Part (a) follows directly from Theorem 4.1 and Lemma 5.3. In order to prove Part (b) we use Lemma 5.3. Since $85 = 127 - 63 + 31 - 15 + 7 - 3 + 1 = 63 + 15 + 7$, we have $\phi(85) = 7$ and $\omega(85) = 3$. Generally, if we take $x = 2^i + 2^j + 5$, for $i - 2 \ge j \ge 4$ then, using Observation 4.1, we obtain $\phi(x) = 7$. Since $x = F_i + F_j + 7$, and by the choice of i, j, we have $\omega(x) = 3$. □

6 An Algorithm for Computing Minimal Embeddings

Let T be the complete binary tree of depth k and size $n = F_k = 2^{k+1}-1$, and fix a set $A = \{a_1, \ldots, a_n\}$ with $a_1 < a_2 < \ldots < a_n$.

Intuitions. In this section we will give a dynamic programming algorithm for computing a minimal embedding. As usual in dynamic programming algorithms, it is convenient to define a partially ordered set Π of instances of a more general embedding problem. Each instance $\pi \in \Pi$ is given by a certain subgraph H of T and an interval I. Some restrictions are imposed on the form of H and position of I, so that the number of instances is small. By an embedding of π we will mean an embedding of H into I. With each such embedding we associate its cost, and by $opt(\pi)$ we will denote the minimal embedding cost of π. Each instance can be either *atomic* or not. Atomic instances are nondecomposable and their optimal cost can be computed in constant time. Each nonatomic instance π can be decomposed into two other instances, π_1 and π_2, that are smaller than π in the partial order on Π. We denote such a composition by $\pi = \pi_1 \otimes \pi_2$. The composition has the property that $opt(\pi) = opt(\pi_1) + opt(\pi_2)$. Typically, π may have many decompositions. This gives rise to a dynamic programming formula to compute the optimal cost of π:

$$opt(\pi) \;=\; min_{\pi=\pi_1\otimes\pi_2} \{opt(\pi_1) + opt(\pi_2)\}\,. \tag{1}$$

The algorithm consists of initializing the optimal cost values for atomic instances, and then computing optimal costs of other instances according to the partial order of Π, using the dynamic programming formula (1). The largest instance in Π corresponds to the problem of embedding T into A, and its minimal cost will be computed last and output by the algorithm.

The key point is that we can assume that the embedding is normalized, due to Theorem 3.1. This means that for each node v and for each neighbor w of v we know in advance whether to embed w to the left or to the right of v. This imposes restrictions on minimal embeddings of instances π, and we do not need to consider embeddings of π that cannot be extended to a normal embedding of the whole tree T.

In order to give the reader some more intuition, let us assume that we start at the root r of T. Let $u = \mathit{left}(r)$, $v = \mathit{right}(r)$ and $m = F_{k-1}$. One of two situations is possible:

- $T(u) \cup \{r\}$ is embedded into $\{1, \ldots, m+1\}$ and $T(v)$ is embedded into $\{m+2, \ldots, n\}$, or
- $T(u)$ is embedded into $\{1, \ldots, m\}$ and $T(v) \cup \{r\}$ is embedded into $\{m+1, \ldots, n\}$.

Hence we have to consider generalized embedding subproblems: embeddings of $T(u) \cup \{r\}$ or $T(v) \cup \{r\}$. The embedding of $T(u) \cup \{r\}$ into $I = \{1, \ldots, m+1\}$ will be described by the instance $\pi = (u, \{r, u\}, 0, I)$. The third parameter, number 0, means here that there are no edges between nodes embedded to the left of I and those embedded to the right of I. For nodes of greater depth, this parameter can take positive values. At the next level we will decompose tree $T(u)$ and, consequently, it will be necessary to consider an instance of the form $\pi_1 = (\mathit{left}(u), \{r, u, \mathit{left}(u)\}, 0, J)$, where J is a subinterval in which the set $T(\mathit{left}(u)) \cup \{r, u\}$ is to be embedded.

Generally, each instance is associated with a node v, and it corresponds to the part of T that could be embedded into an interval $I = [i, j] = \{i, \ldots, j\}$, where i and j are candidate values of $\mathit{min}_f(v)$ and $\mathit{max}_f(v)$, for some normal embedding. Figure 2 shows what needs to be described by an instance: it contains a subtree $T(v)$, interval I, a set $S \subseteq \mathit{Anc}(v)$ that will be embedded into I together with $T(v) - \{v\}$, plus some edges between ancestors of v that cross interval I. However, we will not store those cross edges explicitly, we only need to remember *how many* edges cross I. Such an instance will be denoted by a 4-tuple $\pi = [v, S, \ell, I]$. In order to compute the minimal cost for this instance, we will apply the dynamic programming formula to all partitions of π into a pair of instances of the form $[\mathit{left}(v), S_1, \ell_1, I_1]$, $[\mathit{right}(v), S_2, \ell_2, I_2]$.

Note that we do not require that $v \in S$. In other words, we do not insist that v is embedded into I. If v is an internal node, then we can always assume that $v \in S$; this follows from the definition of normal embeddings. The need to allow the possibility $v \notin S$ arises when we consider the case when v is a leaf. One might try to view this case to be an atomic instance. However, it turns out that computing the optimal embedding for such instances is not easier than when v is an internal node. Thus we will continue to partition each instance $[v, S, \ell, I]$ into instances $[v, S_1, \ell_1, I_1]$, $[v, S_2, \ell_2, I_2]$, and then, clearly, v will belong to only one of S_1, S_2.

Definition of instances. Now we define instances in Π formally. Each instance in Π is a 4-tuple $\pi = [v, S, \ell, I]$, where $v \in V_T$, and

- S is a set of ancestors of v, $S \subseteq \mathit{Anc}(v)$. (Recall that $v \in \mathit{Anc}(v)$.)
- I is an *approximate home interval* for $T(v)$, that is $I = [i, j]$, where

$$\begin{aligned} |i - \mathit{min}_\Phi(v)| &\leq \mathit{depth}(v), \\ |j - \mathit{max}_\Phi(v)| &\leq \mathit{depth}(v), \text{ and} \\ |I| &= j - i + 1 = |S| + |T(v) - \{v\}|. \end{aligned}$$

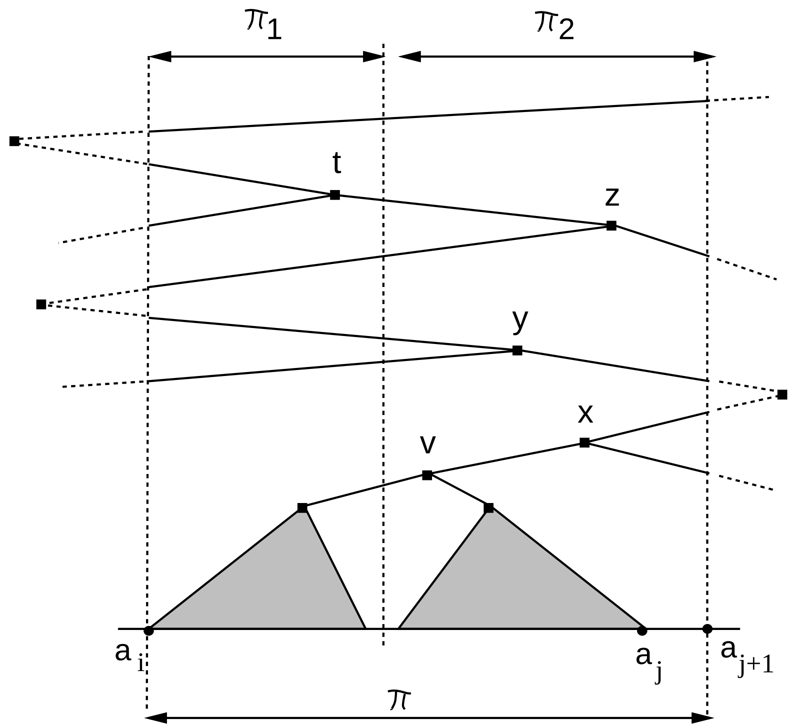

Figure 2: An instance $\pi = [v, I, \ell, S]$ with $I = [i, j]$, $\ell = 1$, $S = \{v, x, y, z, t\}$. The picture shows that $\pi = \pi_1 \otimes \pi_2$, for some instances π_1 and π_2, and it shows a normal embedding of π. The cost consists of the total cost of edges in $T(v)$, plus the solid-line parts of other edges.

- $\ell \in \{0, \ldots, \mathit{depth}(v) - 1\}$ is the number of edges that cross I.

The above instance π will be called *rooted at* v. Denote $V_\pi = T(v) \cup S$, and let E_π be the set of all edges of T whose at least one endpoint belongs to V_π.

An instance $\pi = [v, S, \ell, I]$ is called *atomic*, if v is a leaf, and $|S| = 1$ (and consequently, $I = [i, i]$, for some i).

Embeddings. By an embedding of π we will understand a 1-1 function $f : V_\pi \to I$. An embedding f is called *normal* if for each edge $(x, y) \in E_\pi$, with $x, y \in V_\pi$, we have $f(x) < f(y)$ iff $x \prec y$. An embedding f of π needs to be normal in order to have an extension to a normal embedding of T. (However, some normal embeddings of π may not have a normal extension, or even may not have any extension at all.) Thus we can restrict ourselves to consider only normal embeddings of all instances π.

The cost of f, $\mathit{cost}_A(f)$, is defined similarly to the cost function for embedding trees, except that now we take into account only edges in E_π, and each such edge contributes only the part that lies within the interval $[a_i, a_{j+1}]$ (if $j = n$, we assume $a_{n+1} = a_n$.) In order to define it precisely, it is convenient to extend f to V_T as follows: Consider $y \notin V_\pi$. If $y \prec x$ for some $x \in V_\pi$ then $f(y) = i$, if $x \prec y$ for some $x \in V_\pi$ then $f(y) = j + 1$. If none of these cases holds, define $f(y)$ arbitrarily. The cost of f, denoted $cost_\pi(f)$, can be now defined by

$$cost_\pi(f) \;=\; \sum_{(u,v)\in E_\pi} |a_{f(u)} - a_{f(v)}|.$$

The optimal embedding cost of π is

$$opt(\pi) \;=\; min_f \{cost_\pi(f)\} + (a_{j+1} - a_i)\ell,$$

where the minimum is over all normal embeddings f of π. The second term is the cost of all edges that cross I.

Composition. Now we define formally the composition relation. Given $\pi, \pi_1, \pi_2 \in \Pi$, with $\pi = [v, S, \ell, I]$, where $I = [i, j]$ and $\pi_s = [u_s, S_s, \ell_s, I_s]$, $s = 1, 2$, we say that π is a *composition* of π_1 and π_2, denoted $\pi = \pi_1 \otimes \pi_2$, if the following conditions hold:

(comp1) If v is internal, then $u_1 = left(v)$ and $u_2 = right(v)$. If v is a leaf then $u_1 = u_2 = v$.

(comp2) $I_1 = [i, p]$ and $I_2 = [p + 1, j]$ for some $i < p < j$,

(comp3) S_1, S_2 is a partition of $S \cup \{left(v), right(v)\}$, that is $S_1 \cup S_2 = S \cup \{left(v), right(v)\}$ and $S_1 \cap S_2 = \emptyset$. (If v is a leaf, then assume $\{left(v), right(v)\} = \emptyset$.)

(comp4) Edges that cross from π_1 to π_2 can me matched with those that cross from π_2 to π_1. Formally, it can be explained as follows: Let m_1 be the number of edges $(x, y) \in E_{\pi_1}$ such that $x \in V_{\pi_1}$, $x \prec y$, but $y \notin V_{\pi_2}$. Define m_2 symmetrically. Then $\ell_1 + m_1 = \ell_2 + m_2$.

General strategy. A straighftorward application of the dynamic programming strategy yields the algorithm $\mathcal{A}$ in Figure 3. We only show how to compute the optimal cost.

Final algorithm. Observe that $\mathcal{A}$ stores a lot of unnecessary information. In order to compute the optimal cost of all instances rooted at a node $u \in V_T$ which is not a leaf, we only need optimal costs of instances rooted at $left(u)$ and $right(u)$. This suggests that, in order to save space, we can traverse the tree in postorder, and after computing optimal costs for instances rooted at a node u we can discard information about all instances rooted at $left(u)$ and $right(u)$. Overall, the algorithm is a combination of divide-and-conquer and dynamic programming. It also resembles the naive embedding, except that now at each node we compute information about optimal costs of small perturbations of the naive embedding.

This new algorithm $\mathcal{B}$ is given in Figure 4. We only show how to compute the optimal cost. In the proof of Theorem 6.1 we will show how to modify Algorithm $\mathcal{B}$ to actually construct the minimal embedding function without increasing the space complexity.

Algorithm $\mathcal{A}$:

for each $\pi \in \Pi$ rooted at a leaf **do**
compute $opt(\pi)$;

for $d := k-1$ **downto** 0 **do**
for each π rooted at depth d **do**
$opt(\pi) \ := \ min_{\pi_1 \otimes \pi_2 = \pi} \{opt(\pi_1) + opt(\pi_2)\}$.

output $opt(\pi_0)$, for $\pi_0 = [r, \{r\}, 0, [1, n]]$.

Figure 3: The algorithm $\mathcal{A}$ for computing minimal embeddings.

Theorem 6.1 *The minimal linear embedding can be computed in time $\widetilde{O}(n^{1+\log 3}) = O(n^{2.59})$, using $\widetilde{O}(n)$ space.*

Proof: We will prove first that Algorithm $\mathcal{B}$ computes the minimal embedding cost in claimed complexity bounds. Later we will show how to modify $\mathcal{B}$ to construct a minimal embedding within the same bounds.

Correctness: We claim that Algorithm $\mathcal{B}$ computes correctly $opt(\pi)$ for each $\pi \in \Pi$. The proof is by induction. The claim is clearly true for atomic instances. In order to prove the inductive step, it is sufficient to show that the equation (1) is true. Clearly, $opt(\pi) \leq min_{\pi = \pi_1 \otimes \pi_2} \{opt(\pi_1) + opt(\pi_2)\}$. The proof of the reverse inequality is based on the observation that Lemma 3.2 holds for normal embeddings of all instances in Π (not only for normal embeddings of the whole tree). Suppose that v is not a leaf (the case when v is a leaf is similar). For a given $\pi = [v, S, \ell, I]$, and a minimal embedding f of π, let $m = max_f(left(v))$. We use the vertex sets $f^{-1}([i, m])$ and $f^{-1}([m+1, j])$ to define two instances π_1 and π_2, rooted at $left(v)$ and $right(v)$ respectively, and then we construct embeddings f_s for π_s, $s = 1, 2$, such that $cost_\pi(f) = cost_{\pi_1}(f_1) + cost_{\pi_2}(f_2)$. The details are left to the reader.

Complexity: The number of atomic instances is $\widetilde{O}(n)$, so the initialization time is $\widetilde{O}(n)$ as well.

We claim that the total number of operations $\otimes$ is $\widetilde{O}(n^{1+\log 3})$. This can be proved as follows: Fix a node v with $depth(v) = d$. If π is rooted at v, say $\pi = [v, S, \ell, I]$, and $|S| = s$, then the number of decompositions of π is bounded from above by $2^s d$, since $\ell \leq d$, and each partition of S determines uniquely the partition of I. The number of subsets $S \subseteq Anc(v)$ of size s is $\binom{d+1}{s}$. Then, since $d \leq k$, the number of compositions of instances rooted at v is bounded by

$$\sum_{s=0}^{d+1} \binom{d+1}{s} 2^s d \ = \ 3^{d+1} d \ \leq \ 3^{k+1} k \ \leq \ (n+1)^{\log 3} \log(n+1).$$

After summing that over all nodes we obtain that the total number of composition is $\widetilde{O}(n^{1+\log 3})$.

```
Algorithm B: call Traverse(r);

  procedure Traverse(v : node);
    begin
      if v is a leaf then
        for each π rooted at v do initialize opt(π)
      else begin
        Traverse(left(v));
        Traverse(right(v));
        for each π rooted at v do begin
          opt(π) := min_{π1⊗π2=π} {opt(π1) + opt(π2)};
          discard information about instances rooted
          at left(v) and right(v)
        end
      end
    end
```

Figure 4: The improved algorithm $\mathcal{B}$.

Given π, each decomposition $\pi = \pi_1 \otimes \pi_2$ can be generated in time $\widetilde{O}(1)$, using a simple recursive procedure for generating partitions of S. If we store the instances in a lexicographic order, finding $opt(\pi_1)$ and $opt(\pi_2)$ can be done in time $\widetilde{O}(1)$. Therefore the total time complexity is $\widetilde{O}(n^{1+\log 3}) = O(n^{2.59})$.

Constructing a minimal embedding. A standard approach is to keep track of minimal embeddings of all instances, and then reconstruct the whole embedding from this information. However, this does not seem to be feasible if we want to use only $\widetilde{O}(n)$ memory.

We modify Algorithm $\mathcal{B}$ as follows. First, in step 0, compute the minimal embedding cost, but at the root we record the information about the instances π_{v_1} and π_{v_2}, rooted at $v_1 = left(r)$ and $v_2 = right(r)$, respectively, such that $opt(\pi_1) + opt(\pi_2) = opt(A)$.

In step d, $1 \leq d \leq k-1$, for each node v of depth d, we remember an instance π_v that is a part of the minimal embedding. More specifically, let $\pi_v = [v, S_v, \ell_v, I_v]$, for each v. Then (a) the sets I_v form a partition of $[1, n]$, (b) the sets S_v form a partition of the set of all nodes of depth $\leq d$, (c) for each v the number ℓ_v is equal to the number of edges (x, y), where $x \in S_{v_1}$, $y \in S_{v_2}$ for some nodes v_1, v_2 of depth d such that $v_1 \prec v \prec v_2$, and (d) $\sum_v opt(\pi_v) = opt(A)$, where the summation is over all nodes of depth d.

In step d we execute Algorithm $\mathcal{B}$ as in the first step, but we stop at depth d, and we store instances π_u with $depth(u) = d+1$. By repeating this process k times, we can reconstruct the minimal embedding. The time complexity will increase by a factor of k, which is still $\widetilde{O}(n^{1+\log 3})$,

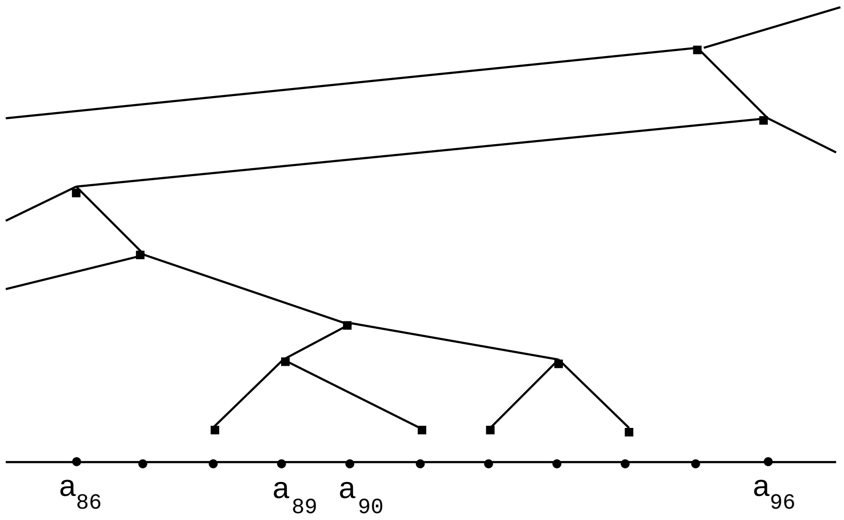

Figure 5: Fragment of the Chung embedding for $k = 7$.

and the space complexity will remain $\widetilde{O}(n)$. $\Box$

7 Final Comments

As we mentioned in the introduction, the naive embedding is *oblivious*, in the sense that it depends only on n, the cardinality of A. In spite of this, as we have shown, it approximates the minimal solution within factor of $\frac{7}{3}$. It raises a natural question: how well can we approximate the minimal embedding with oblivious embeddings? In particular, is it possible that there exists a minimal oblivious embedding for each full number n?

Another oblivious embedding of complete binary trees was given by Chung in [Ch78]. The Chung embedding Λ is defined as follows: We embed the root in the middle, $\Lambda(r) = F_{k-1} + 1$. The left and right subtrees are embedded symmetrically, so we will show only the embedding of the left one. Let $v = \mathit{left}^{k-h}(r)$, for some $1 \le h \le k-1$. If $h = 1$, that is if v is the leftmost leaf, then $\Lambda(v) = 1$. If $h > 1$, then let $\Lambda(v) = F_{h-1} + F_{h-2} + 1$, $\Lambda(\mathit{right}(v)) = F_{h-1} + F_{h-2} + 2$, and embed recursively $T(\mathit{left}(\mathit{right}(v)))$ into the interval $[F_{h-1} + 1, F_{h-1} + F_{h-2}]$ and $T(\mathit{right}(\mathit{right}(v)))$ into the interval $[F_{h-1} + F_{h-2} + 3, F_h]$. (In this definition, nodes of depth k and $k-2$ are embedded differently than in Figure 1, for the sake of uniformity. The cost remains the same.)

The Chung embedding is minimal when the points in A are equally spaced. In the analysis of the general case, it may be helpful to observe that Lemma 5.3 holds for arbitrary oblivious embeddings, not only for the naive one. Thus the Chung embedding has performance ratio C for a given k iff $|CUT_\Lambda(x)| \le C \cdot \omega(x)$, for each $x = 1, \ldots, F_{k-1}$. In this way, the analysis of the Chung embedding can be reduced to analyzing the cut cardinalities, or its corresponding *Chung* representations. Using this approach one can easily check, by inspection, that the Chung embedding is optimal for complete binary trees of depth $k \leq 6$. However, it is not optimal for $k = 7$, since $|CUT_\Lambda(89)| = 5$ while $\omega(89) = 3$. Figure 5 shows a fragment of the Chung embedding of the complete binary tree of depth 7, showing only the interval between a_{86} and a_{96}.

Open Problem 1: What is the performance ratio of the Chung embedding? We conjecture that it has a better performance ratio than the naive embedding.

Open Problem 2: What is the best performance ratio that can be achieved with an oblivious embedding?

Open Problem 3: Our analysis of algorithm $\mathcal{B}$ is not tight. It is not difficult, although a little tedious, to make slight improvements of the time complexity by counting more carefully compositions of instances. However, improving time complexity to $O(n^2)$ seems to require a better algorithm. Does there exist an $O(n^2)$ algorithm for this problem?

Acknowledgements. We would like to thank the anonymous referee for useful suggestions that helped us improve the presentation of the paper.

References

[BKRS89] M. Bern, H. Karloff, P. Raghavan, B. Schieber, *Fast geometric approximation techniques and geometric embedding problems*, Theoretical Computer Science 106 (1992) 265-282.

[Ch78] F.R.K. Chung, *A conjectured minimum valuation tree*, in "Problems and Solutions", SIAM Review 20 (1978) 601-604.

[Ch84] F.R.K. Chung, *On optimal linear arrangements of trees*, Comp. and Math. with Appl. 10 (1984) 43-60.

[ES75] S. Even, Y. Shiloach, *NP-completeness of several arrangement problems*, Report No. 43, Department of Computer Science, Technion, Haifa, 1975.

[GJ79] M.R. Garey, D.S. Johnson, *Computers and Intractability*, W.H. Freeman and Co., 1979.

[GK76] M.K. Goldberg, I.A. Klipker, *Minimal placing of trees on a line*, Technical Report, Institute of Low Temperatures, Academy of Sciences of Ukraina, 1976, (in Russian).

[Sh79] Y. Shiloach, *A minimum linear arrangement algorithm for undirected trees*, SIAM Journal on Computing 8 (1979) 15-32.