Lossless Compression of Binary Trees with Correlated Vertex Names

Abram Magner, Krzysztof Turowski and Wojciech Szpankowski, *Fellow, IEEE,*

***Abstract*—Compression schemes for advanced data structures have become a central modern challenge. Information theory has traditionally dealt with conventional data such as text, images, or video. In contrast, most data available today is multitype and context-dependent. To meet this challenge, we have recently initiated a systematic study of advanced data structures such as unlabeled graphs [8]. In this paper, we continue this program by considering trees with statistically correlated vertex names. Trees come in many forms, but here we deal with binary plane trees (where order of subtrees matters) and their non-plane version (where order of subtrees doesn't matter). Furthermore, we assume that each name is generated by a known memoryless source (horizontal independence), but a symbol of a vertex name depends in a Markovian sense on the corresponding symbol of the parent vertex name (vertical Markovian dependency). We first evaluate the entropy for both types of trees. Then we propose two compression schemes: COMPRESSPTREE for plane trees with correlated names, and COMPRESSNPTREE for non-plane trees, each running in linear time in the size of the input tree. We prove that the former scheme achieves the lower bound within two bits, while the latter is within a multiplicative factor of 1% of the optimal compression ratio. For non-plane trees we also design an optimal algorithm within two bits of the entropy, with time complexity $O(n^2)$ (where n is the number of leaves in the underlying non-plane tree).**

I. INTRODUCTION

Over the last decade, repositories of various data have grown enormously. Most of the available data is no longer in conventional form, such as text or images. Instead, biological data (e.g., gene expression data, protein interaction networks, phylogenetic trees), topographical maps (containing various information about temperature, pressure, etc.), medical data (cerebral scans, mammogram, etc.), and social network data archives are in the form of *multimodal* data structures, that is, multitype and context dependent structures (see Figure 1). For efficient compression of such data structures, one must take into account not only several different types of information, but also the statistical dependence between the general data labels and the structures themselves. In Figure 1 we show an example of such a multimodal structure representing an annotated protein interaction network in which graphs, trees, DAGs, and text are involved.

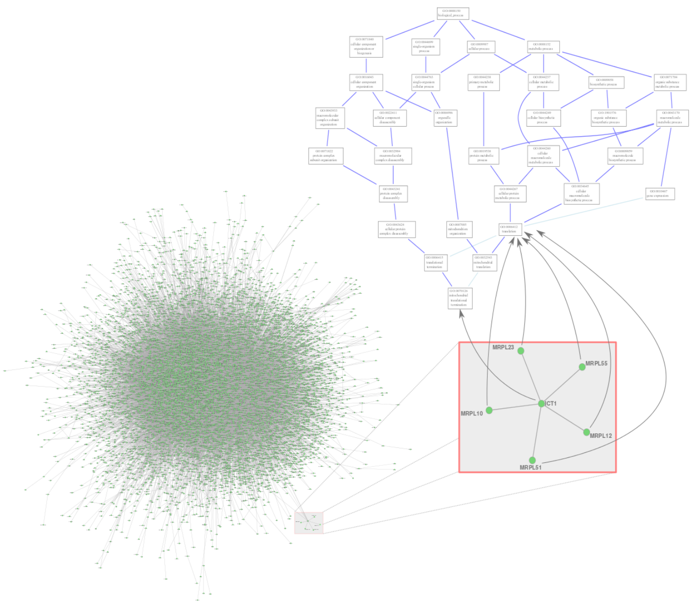


Fig. 1: Annotated protein interaction network: nodes in the network represent proteins in the human interactome, and there is an edge between two nodes if and only if the two proteins interact in some biological process. Associated with each node is a position in a fixed *ontology* (encoded by a DAG whose nodes represent classes of proteins defined by, say, structural or functional characteristics) known as the Gene Ontology [1], [2]. Nodes that are connected in the interaction network are often close in the ontology.

This paper is a first step in the direction of the larger goal of a unified framework for compression of multimodal graph and tree structures. We focus on compression of trees with structure-correlated vertex "names" (strings). There are myriad examples of trees with correlated names. As an example, in Figure 2 we present a Linnaean taxonomy of *hominoid*. As is easy to see, names lower in the tree (children) are (generally) variations of the name at the root of a subtree. Closer to the models that we will consider, there are binary tree models of

Parts of this paper were presented at the 2016 IEEE International Symposium on Information Theory, Barcelona, Spain.

Abram Magner is with the Coordinated Science Lab at Univ. of Illinois at Urbana-Champaign, Urbana, IL, USA, (email: anmagner@illinois.edu).

Krzysztof Turowski is with Dept. of Algorithms and System Modeling, at Gdansk University of Technology, Gdansk, Poland (email: krzturow@pg.gda.pl).

W. Szpankowski is with the Department of Computer Science, Purdue University, IN 47907, USA (e-mail: spa@cs.purdue.edu); also with the Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Poland. This work was supported by NSF Center for Science of Information (CSoI) Grant CCF-0939370, and in addition by NSF Grants CCF-1524312, and NIH Grant 1U01CA198941-01.

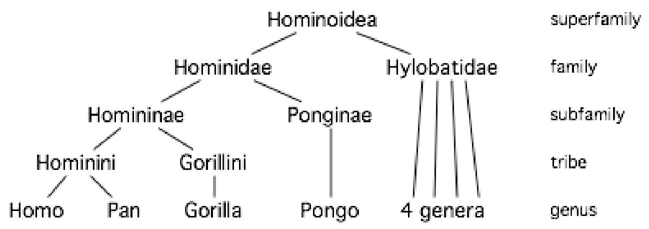


Fig. 2: Linnaean taxonomy of *hominoid*

speciation in phylogenetics, in which each vertex (representing an organism or a species) is associated with a genome (a DNA string); a parent-child relationship in this tree indicates a biological parent-child relationship (either at the level of species or individual organisms, depending on the application context). Naturally, the genome of a child is related to that of its parent; furthermore, the sequence of genomes in a given path down the tree has a Markov property: a child's genome is related to that of its ancestors only through that of its parent.

To capture this intuitive behavior, we first formalize two natural models for random binary trees with correlated names and prove that they are equivalent (see Theorem 1 and Corollary 1).

We focus on two variations: plane-oriented and non-plane-oriented trees [3]. In plane-oriented trees, the order of subtrees matters, while in non-plane-oriented trees (e.g., representing a phylogenetic tree [20]) all orientations of subtrees are equivalent (see Figure 3). For the plane-oriented case, we

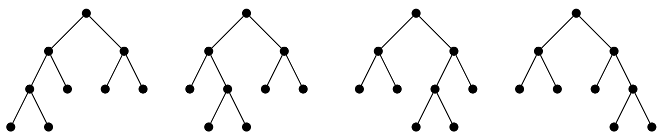


Fig. 3: All plane-ordered representatives of one particular non-plane tree.

give an exact formula for the entropy of the labeled tree (see Theorem 2) as well as an efficient compression algorithm based on arithmetic encoding whose expected code length is within two bits of the optimal length. In the non-plane case, we focus on unlabeled trees and first derive the entropy rate (see Theorem 3). Then we propose two compression algorithms: a simple one of time complexity $O(n)$ (where n is the number of leaves) that achieves compression within a multiplicative factor of 1% of the entropy, and then an optimal algorithm of time complexity $O(n^2)$ that is within two bits from the entropy. We should point out that non-plane trees are more challenging to study due to the typically large number of symmetries. In particular, the entropy for non-plane trees depends on what we call the *tree Rényi entropy* of order 2 discussed in the remark below Theorem 3.

Regarding prior work, literature on tree and graph compression is quite scarce. For unlabeled graphs there are some recent information-theoretic results, including [7], [8] (see also [10]) and [9]. In 1990, Naor [7] proposed an efficiently computable representation for unlabeled graphs (solving Turán's [11] open question) that is optimal up to the first two leading terms of the entropy when all unlabeled graphs are equally likely. Naor's result is asymptotically a special case of recent work of Choi and Szpankowski [8], who extended Naor's result to general Erdős-Rényi graphs. In particular, in [8] the entropy and an optimal compression algorithm (up to two leading terms of the entropy) for Erdős-Rényi graph structures were presented. Furthermore, in [9] an automata approach was used to design an optimal graph compression scheme. There also have been some heuristic methods for real-world graphs compression including grammar-based compression for some data structures. Peshkin [23] proposed an algorithm for a graphical extension of the one-dimensional SEQUITUR compression method. However, SEQUITUR is known not to be asymptotically optimal. For binary plane-oriented trees rigorous information-theoretic results were obtained in [13], complemented by a universal grammar-based lossless coding scheme [14].

However, for structures with vertex names, which is the topic of this paper, there have been almost no attempts at theoretical analyses, with the notable exception of [12] for sparse Erdős-Rényi graphs. The only significant algorithmic results have been based on heuristics, exploiting the well-known properties of special kinds of graphs: for example in the case of Web graphs, their low degree and clustering [24] (see also [25], [27]). Similarly, there are some algorithms with good practical compression rate for phylogenetic trees (see [26]); however, they too lack any theoretical guarantees of their performance.

The tree models (without vertex names) that we consider in this paper are variations of the *Yule* distribution [17], which commonly arises in mathematical phylogenetics. Various aspects of these tree models have been studied in the past (see, e.g., [16]), but the addition of vertex names and the consideration of information-theoretic questions about the resulting models seems to be a novel aspect of the present work.

The rest of the paper is structured as follows: in Section II, we formulate the models and present the main results, including entropy formulas/bounds and performances of the compression algorithms. In Section III, we present the derivations of the entropy results. In Section IV, we present the compression algorithms and their analyses. Finally, in Section V, we conclude with future directions.

II. Main theoretical results

In this section we introduce the concepts of binary plane and non-plane trees together with the notion of locally correlated names associated with their vertices. Then we define two

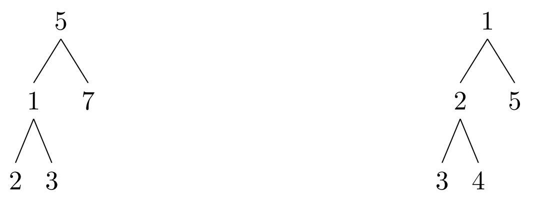


Fig. 4: A rooted plane tree and its standardization. The left tree can be represented by the list of triples $\{(5, 1, 7), (1, 2, 3)\}$. After standardization, this becomes $\{(1, 2, 5), (2, 3, 4)\}$. Note that this is distinct from the tree $\{(1, 2, 3), (3, 4, 5)\}$ or, equivalently, $\{(5, 1, 7), (7, 2, 3)\}$.

models for these types of trees with correlated names and show the equivalence of these two models.

A. Basic definitions and notation

We call a rooted tree a *plane tree* when we distinguish left-to-right-order of the children of the nodes in the embedding of a tree on a plane (see [3]). To avoid confusion, we call a tree with no fixed ordering of its subtrees a *non-plane* tree (also known in the literature as *Otter trees* [20]). In a non-plane tree any orientation of subtrees is equivalent.

Let $\mathcal{T}$ be the set of all binary rooted plane trees having finitely many vertices and, for each positive integer n, let $\mathcal{T}_n$ be the subset of $\mathcal{T}$ consisting of all trees with exactly n leaves. Similarly, let $\mathcal{S}$ and $\mathcal{S}_n$ be the set of all binary rooted non-plane trees with finitely many vertices and exactly n leaves, respectively.

We can also augment our trees with vertex names – given the alphabet $\mathcal{A}$, names are simply words from $\mathcal{A}^m$ for some integer $m \geq 1$. Let $\mathcal{LT}_n$ and $\mathcal{LS}_n$ be the set of all binary rooted plane and non-plane trees with names, respectively, having exactly n leaves with each vertex assigned a name – a word from $\mathcal{A}^m$. In this paper we consider the case where names are correlated with the structure as discussed below.

Formally, a rooted plane binary tree $t \in \mathcal{T}_n$ can be uniquely described by a set of triples (v_i, v_j, v_k), consisting of a parent vertex and its left and right children. Given such a set of triples defining a valid rooted plane binary tree, we can standardize it to a tree defined on the vertex set $[2n-1] = \{1, ..., 2n-1\}$ by replacing vertex v in each triple by the depth-first search index of v. We then consider two trees to be the same if they have the same standard representation. See the example in Figure 4. Furthermore, a tree with names $lt \in \mathcal{LT}_n$ can be uniquely described as a pair (t, f), where t is a tree, described as above, and f is a function from the vertices of t to the words in $\mathcal{A}^m$. We can also describe an element of $\mathcal{LT}_n$ as a set of pairs (v_i, l_i) for all $i = 1, 2, \ldots, 2n-1$ (representing the vertices) and their names. If we identify the vertices (v_i) with the integers $1, 2, \ldots, 2n-1$, f can be described as a sequence of $2n-1$ words from $\mathcal{A}^m$.

1) Tree notation:

Here we give notation regarding trees that will be used throughout the paper. Let t be a binary plane tree, and consider a vertex v in t.

We denote by $\lambda(t, v)$ and $\rho(t, v)$ the left and right child of v in t, respectively (i.e., these are vertices in t). By $t(v)$, we shall mean the subtree of t rooted at v. We denote by t^L and t^R the left and right subtree of t, respectively. We denote by $\Delta(t)$ the number of leaves in the tree t. Finally, we denote by $root(t)$ the root node of t.

B. The model

We now present a model for generating plane trees with structure-correlated names. This model will be such that individual names are tuples of independent and identically distributed symbols, (a property which we shall call *horizontal* independence), but the letters of names of children depend on the name of the parent (vertical Markovian dependence).

Our main model MT_1 is defined as follows: given the number n of leaves in the tree, the length of the names m, the alphabet $\mathcal{A}$ (of size $|\mathcal{A}|$) and the transition probability matrix P of size $|\mathcal{A}| \times |\mathcal{A}|$ (representing an ergodic Markov chain) with its stationary distribution π (i.e. $\pi P = \pi$), we define a random variable LT_n as a result of the following process: starting from a single node with a randomly generated name of length m by a memoryless source with distribution π, we repeat the following steps until a tree has exactly n leaves:

- pick uniformly at random a leaf v in the tree generated in the previous step,
- append two children v_L and v_R to v,
- generate correlated names for v_L and v_R by taking each letter from v and generating new letters according to P: for every letter of the parent we pick the corresponding row of matrix P and generate randomly the respective letters for v_L and v_R.

Alternatively, LT_n is equivalent to an ordered pair of random variables (T_n, F_n), where T_n is a random variable supported on $\mathcal{T}_n$ and F_n is a random variable supported on sequences of words from $\mathcal{A}^m$ of length $2n-1$.

Our second model, MT_2 (also known as the binary search tree model), is ubiquitous in the computer science literature, arising for example in the context of binary search trees formed by inserting a random permutation of $[n-1]$ into a binary search tree [5]. Under this model we generate a random tree T_n as follows: t is equal to the unique tree in $\mathcal{T}_1$ and we associate a number n with its single vertex. Then, in each recursive step, let $v_1, v_2, \ldots, v_k$ be the leaves of t, and let integers $n_1, n_2, \ldots, n_k$ be the values assigned to these leaves, respectively. For each leaf v_i with value $n_i > 1$, randomly select an integer s_i from the set $\{1, \ldots, n_i - 1\}$ with probability $\frac{1}{n_i - 1}$ (independently of all other such leaves), and then grow two edges from v_i with left edge terminating at a leaf of the extended tree with value s_i and right edge terminating at a leaf of the extended tree with value $n_i - s_i$. The extended tree is the result of the current recursive step. Clearly, the recursion terminates with a binary tree having exactly n leaves, in which each leaf has assigned value 1; this tree is T_n.

The assignment of names to such trees, given length m, alphabet $\mathcal{A}$, transition matrix P and its stationary distribution π, proceeds exactly as in the MT_1 model: we first generate a random word from $\mathcal{A}^m$ by a memoryless source with distribution π and assign it to a root, then generate correlated names for children, according to the names of the parents and to the probabilities in P. Throughout the paper, we will write $P(y|x)$, for symbols $x, y \in \mathcal{A}$, as the probability of transitioning from x to y according to the Markov chain P. We will also abuse notation and write $P(x)$ as the probability assigned to x by π.

Recall that $\Delta(t)$ is the number of leaves of a tree t, and $\Delta(t(v))$ denotes the number of leaves of a tree t rooted at v. It is easy to see [13] that under the model MT_2, $\mathbb{P}(T_1 = t_1) = 1$ (where t_1 is the unique tree in $\mathcal{T}_1$) and

$$\mathbb{P}(T_n = t) = \frac{1}{n-1}\mathbb{P}(T_{\Delta(t^L)} = t^L)\mathbb{P}(T_{\Delta(t^R)} = t^R),$$

which leads us to the formula

$$\mathbb{P}(T_n = t) = \prod_{v \in \overset{\circ}{V}(t)} (\Delta(t(v)) - 1)^{-1} \tag{1}$$

where $\overset{\circ}{V}(t)$ is the set of the internal vertices of t. It turns out that the probability distribution in the MT_1 model is exactly the same, as we prove below.

Theorem 1. *Under the model MT_1 it holds that*

$$\mathbb{P}(T_n = t) = \prod_{v \in \overset{\circ}{V}(t)} (\Delta(t(v)) - 1)^{-1} \tag{2}$$

where $\overset{\circ}{V}(t)$ is the set of the internal vertices of t.

Proof. We follow [17] and use the concept of *labeled histories*: a pair (t, ξ), where $t \in \mathcal{T}_n$ (generated by the model MT_1) and ξ is a permutation of the natural numbers from 1 to $n-1$, assigned to the *internal* $n-1$ vertices of t, such that every path from the root forms an ascending sequence. If we think of the internal nodes of t as being associated with their DFS numbers, then the permutation ξ is a function mapping each internal node of t to the time at which its two children were added to it. Thus, for instance, the root always receives the number 1, meaning that $\xi(1) = 1$. Clearly, for each t, each labeled history (t, ξ) corresponds to exactly one sequence of generation, as it defines uniquely the order of the leaves, which are picked during consecutive stages of the algorithm. Moreover, the probability of each feasible labeled history (t, ξ) is equal to $\frac{1}{(n-1)!}$ since it involves choosing one leaf from k available at the kth stage of the algorithm for $k = 1, \ldots, n-1$. Therefore, denoting $q(t) = |\{\xi\colon (t, \xi) \text{ is a labelled history}\}|$, we find

$$\mathbb{P}(T_n = t) = \frac{q(t)}{(n-1)!}.$$

Note that if $\Delta(t(v)) = k$ for any vertex v of t, then we know that the sequence of node choices corresponding to (t, ξ) must contain exactly $k-1$ internal vertices from the subtree of v, and that v is the first of them. Moreover, for the subsequence in the sequence corresponding to (t, ξ) of a subtree $t(v)$ rooted at vertex v, the sequences of $\Delta(t(v)^L) - 1$ vertices from its left subtree and of $\Delta(t(v)^R) - 1$ vertices from its right subtree are interleaved in any order. Thus we arrive at the following recurrence for $q(t)$:

$$\begin{aligned} q(t) &= \binom{\Delta(t) - 2}{\Delta(t^L) - 1} q(t^L) q(t^R) \\ &= \frac{(\Delta(t) - 2)!}{(\Delta(t^L) - 1)!(\Delta(t^R) - 1)!} q(t^L) q(t^R). \end{aligned}$$

This recurrence can be solved by observing that each internal vertex appears exactly once in the numerator, and that each internal vertex not equal to the root r appears exactly once in the denominator. Hence

$$q(t) = \frac{\prod_{v \in \overset{\circ}{V}(t)} (\Delta(t(v)) - 2)!}{\prod_{v \in \overset{\circ}{V}(t) \setminus \{r\}} (\Delta(t(v)) - 1)!}.$$

Since $\Delta(t(r)) = \Delta(t) = n$, we thus obtain

$$q(t) = \frac{\prod_{v \in \overset{\circ}{V}(t)} (\Delta(t(v)) - 2)!}{\prod_{v \in \overset{\circ}{V}(t) \setminus \{r\}} (\Delta(t(v)) - 1)!} = \frac{(n-2)!}{\prod_{v \in \overset{\circ}{V}(t) \setminus \{r\}} (\Delta(t(v)) - 1)}$$

leading finally to

$$\mathbb{P}(T_n = t) = \frac{q(t)}{(n-1)!} = \frac{1}{\prod_{v \in \overset{\circ}{V}(t)} (\Delta(t(v)) - 1)}$$

which completes the proof. □

Clearly, both models are equivalent, since the underlying binary plane trees have exactly the same probability distributions:

Corollary 1. *The models MT_1 and MT_2 are equivalent in the sense that they lead to the same probability distribution on trees.*

We now can extend the above equivalence to non-plane trees. We define models MS_1 and MS_2 for non-plane trees: just generate the tree according to MT_1 (respectively, MT_2) and then treat the resulting plane tree T_n as a non-plane one S_n. Since MT_1 and MT_2 are equivalent, so are MS_1 and MS_2. In graph terminology, MS_1 and MS_2 are unlabeled version of their MT counterparts [8].

III. Entropy evaluations

In this section we estimate the entropy for the plane-oriented trees with names and that of non-plane trees without names. We shall see that the latter problem is mathematically more challenging.

A. The entropy of the plane trees with names

Recall that $\Delta(LT_n^L)$ is a random variable corresponding to the number of leaves in the left subtree of LT_n. From the previous section, we know that $\mathbb{P}(\Delta(LT_n^L) = i) = \frac{1}{n-1}$. Let also r_n denote the root of LT_n, for each n.

First, we observe that a tree with names LT_n is uniquely described by the quadruple $(\Delta(LT_n^L), F_n(r_n), LT_n^L, LT_n^R)$, where we recall $F_n(r_n)$ is the name associated with r_n, the root of LT_n. Therefore

$$H(LT_n|F_n(r_n)) = H(\Delta(LT_n^L), F_n(r_n), LT_n^L, LT_n^R|F_n(r_n)).$$

Second, using the fact that $\Delta(LT_n^L)$ is independent from $F_n(r_n)$ and the fact that given the complete description of the root (its name and the number of leaves in both subtrees), LT_n^L and LT_n^R are conditionally independent, we find

$$\begin{aligned} H(LT_n|F_n(r_n)) &= H(\Delta(LT_n^L)) \\ &+ \sum_{k=1}^{n-1} \big(H(LT_n^L|F_n(r_n), \Delta(LT_n^L) = k) + \\ &H(LT_n^R|F_n(r_n), \Delta(LT_n^L) = k)\big)\, \mathbb{P}(\Delta(LT_n^L) = k). \end{aligned} \tag{3}$$

From the definition of $\Delta(LT_n^L)$ we know that $H(\Delta(LT_n^L)) = \log_2(n-1)$.

Next, it is sufficient to note that the random variable LT_n^L conditioned on $\Delta(LT_n^L) = k$ is identical to the random variable LT_k (the model has the hereditary property), and the same holds for LT_n^R and LT_{n-k}. Similarly, recalling that $\lambda(\cdot,\cdot)$ and $\rho(\cdot,\cdot)$ denote the left and right child of a given node in a given tree, F_n on the left (or right) subtree of LT, given $F_n(\lambda(LT_n, r))$ (respectively, $F_n(\rho(LT_n, r))$) and $\Delta(LT_n^L) = k$, has identical distribution to the random variable F_k (resp. F_{n-k}) given $F_k(r_k)$ (resp. $F_{n-k}(r_{n-k})$).

Then we observe that

$$H(LT_n^L|F_n(r_n), \Delta(LT_n^L) = k) =$$

$$H(LT_n^L, F_n(\lambda(LT_n, r_n))|F_n(r_n), \Delta(LT_n^L) = k)$$

since $F_n(\lambda(LT_n, r_n))$ is already contained in LT_n^L. To continue we see, by the chain rule, that the last entropy is equal to

$$\begin{aligned} &H(F_n(\lambda(LT_n, r_n))|F_n(r_n), \Delta(LT_n^L) = k) \\ &+ H(LT_n^L|F_n(\lambda(LT_n, r_n)), F_n(r_n), \Delta(LT_n^L) = k) \end{aligned} \tag{4}$$

but since the first term doesn't depend on $\Delta(LT_n^L) = k$ and we can ignore $F_n(r_n)$ in the second term we find that the expression (4) is equal to

$$\begin{aligned} &H(F_n(\lambda(LT_n, r_n))|F_n(r_n)) + \\ &H(LT_n^L|F_n(\lambda(LT_n, r_n)), \Delta(LT_n^L) = k) \end{aligned}$$

which finally becomes

$$mh(P) + H(LT_k^L|F_k(r_k)), \tag{5}$$

where $h(P) = -\sum_{a\in\mathcal{A}} \pi(a) \sum_{b\in\mathcal{A}} P(b|a) \log P(b|a)$ is the entropy of the Markov process with transition matrix P and stationary distribution π. It is also the entropy of a child letter given its parent letter. Note that the dependence of the names on the tree structure is applied in establishing (5), since we use the fact that the name of $\lambda(LT_n, r_n)$ is generated from that of r_n, its parent. Thus, by (3), this leads us to the following recurrence:

$$\begin{aligned} H(LT_n|F_n(r_n)) &= \log_2(n-1) + 2mh(P) \\ &+ \frac{2}{n-1}\sum_{k=1}^{n-1} H(LT_k|F_k(r_k)). \end{aligned} \tag{6}$$

Here in (6) we encounter a recurrence, that will appear also in several other places in the paper. Therefore, we state the following technical lemma, yielding the solution to this recurrence.

Lemma 1. *The recurrence* $x_1 = 0$,

$$x_n = a_n + \frac{2}{n-1}\sum_{k=1}^{n-1} x_k, \quad n \geq 2 \tag{7}$$

has the following solution for $n \geq 2$*:*

$$x_n = a_n + n\sum_{k=2}^{n-1} \frac{2a_k}{k(k+1)}. \tag{8}$$

Proof. By comparing the equations for x_n and x_{n+1} for any $n \geq 2$ we obtain

$$\frac{x_{n+1}}{n+1} = \frac{x_n}{n} + \frac{na_{n+1} - (n-1)a_n}{n(n+1)}$$

Substituting $y_n = \frac{x_n}{n}$ and $b_n = \frac{na_{n+1}-(n-1)a_n}{n(n+1)}$ we get

$$y_{n+1} = y_n + b_n = y_2 + \sum_{k=2}^{n} b_k$$

$$\begin{aligned} y_n = y_2 + \sum_{k=2}^{n-1} b_k &= y_2 + \sum_{k=2}^{n-1}\left(\frac{a_{k+1}}{k+1} - \frac{a_k}{k} + \frac{2a_k}{k(k+1)}\right) \\ &= y_2 + \frac{a_n}{n} - \frac{a_2}{2} + \sum_{k=2}^{n-1}\frac{2a_k}{k(k+1)} = \frac{a_n}{n} + \sum_{k=2}^{n-1}\frac{2a_k}{k(k+1)} \end{aligned}$$

and finally, after multiplying both sides by n, we obtain the desired result. □

This leads to our first main result.

Theorem 2. *The entropy of a binary tree with names of fixed length m, generated according to the model MT_1, is given by*

$$H(LT_n) = \log_2(n-1) + 2n\sum_{k=2}^{n-1}\frac{\log_2(k-1)}{k(k+1)} \qquad (9)$$
$$+2mh(P)(n-1) + mh(\pi)$$

where $h(\pi) = -\sum_{a\in\mathcal{A}}\pi(a)\log\pi(a)$.

Proof. Clearly, $H(LT_n) = H(LT_n|F_n(r_n)) + H(F_n(r_n))$. Moreover if we apply Lemma 1 with $x_n = H(LT_n|F_n(r_n))$ and $a_n = \log_2(n-1) + 2mh(P)$ to the equation (6) we find

$$\begin{aligned} H(LT_n|F_n(r)) &= \log_2(n-1) + 2mh(P) \qquad (10)\\ &+ 2n\sum_{k=2}^{n-1}\frac{\log_2(k-1)}{k(k+1)}\\ &+ 4nmh(P)\sum_{k=2}^{n-1}\frac{1}{k(k+1)}. \end{aligned}$$

But $\sum_{k=2}^{n-1}\frac{1}{k(k+1)} = 1/2 - 1/n$ and $H(F_n(r_n)) = mh(\pi)$, which completes the proof. □

Remark. Note that the first two terms of the expression for $H(LT_n)$ are equal to $H(T_n)$, the tree without names, while the third and fourth terms correspond to $H(F_n|T_n)$, the names given the structure of the tree. Furthermore, note that because F_n and T_n are statistically dependent, $H(LT_n)$ is *strictly less than* $H(F_n) + H(T_n)$, the fundamental lower bound on the expected code length if the names and the tree structure are compressed separately. This is to be expected: if one is given the multiset of names without any additional information about the tree structure, then the compression of the names either duplicates information from the tree (in the case where the tree can be recovered from the names) or is suboptimal in light of the fact that the dependence structure of the names cannot be estimated. In either case, separate compression is suboptimal.

Remark. We know that (see also [13])

$$2\sum_{k=2}^{n}\frac{\log_2(k-1)}{k(k+1)} \approx 1.736$$

for large n (we took $n = 10^6$). The above series converges slowly, with an error term of order $O(\log n/n)$. Therefore one would prefer a more explicit formula, which we offer next for large n. We approximate the sum using the Euler-Maclaurin formula [4] by the integral

$$\sum_{k=1}^{n-2}\frac{\log k}{(k+1)(k+2)} \sim \int_1^{n-2}\frac{\log(x)}{(x+1)(x+2)}dx$$

$$= -\operatorname{Li}_2\left(1-\frac{n}{2}\right) + \operatorname{Li}_2(2-n) + \log\left(2-\frac{2}{n}\right)\cdot\log(n-2)$$
$$+\operatorname{Li}_2\left(-\frac{1}{2}\right) + \frac{\pi^2}{12},$$

where

$$\operatorname{Li}_2(x) = \int_1^x \frac{\log t}{1-t}dt$$

is the *dilogarithmic integral*. For large x the following holds [18], [19]

$$\operatorname{Li}_2(x) = -\frac{1}{2}\log^2 x - \frac{\pi^2}{6} + O(\log x/x).$$

In fact, to get a better approximation of the sum we need two extra terms in the Euler-Maclaurin formula which leads to the following approximation

$$\sum_{k=1}^{n-2}\frac{\log k}{(k+1)(k+2)} \approx$$
$$\operatorname{Li}_2(3/2) + \frac{1}{12}\pi^2 + \frac{1}{2}\log^2(2) - \frac{1}{72} + \frac{23}{12960} = 0.868\ldots$$

which matches the first three digits of the sum.

Remark Note that the total entropy is proportional to n and m. In a typical setting $m = O(1)$ or $\Theta(\log n)$ (which is certainly needed if we want all the labels to be distinct), so $H(LT_n)$ would be $O(n)$ or $O(n\log n)$, respectively.

B. The entropy of the non-plane trees

Now we turn our attention to the non-plane trees and the case when $m = 0$ or, equivalently, S_n instead of LS_n. Let S_n be the random variable with probability distribution given by the MS_2 (or, equivalently, MS_1) model. For any $s \in \mathcal{S}$ and $t \in \mathcal{T}$ let $t \sim s$ mean that the plane tree t is isomorphic to the non-plane tree s. Furthermore, we use the following notation: $[s] = \{t \in \mathcal{T} \colon t \sim s\}$. For any $t_1, t_2 \in \mathcal{T}_n$ such that $t_1 \sim s$ and $t_2 \sim s$ it holds that $\mathbb{P}(T_n = t_1) = \mathbb{P}(T_n = t_2)$, since there is also an isomorphism between them [17]. By definition, s corresponds to $|[s]|$ isomorphic plane trees, so for any $t \sim s$ it holds that

$$\begin{aligned} \mathbb{P}(S_n = s) &= |[s]|\mathbb{P}(T_n = t) \qquad (11)\\ \mathbb{P}(T_n = t|S_n = s) &= \frac{1}{|[s]|}. \qquad (12) \end{aligned}$$

Let us now introduce two functions: $X(t)$ and $Y(t)$, which are equal to the number of internal vertices of $t \in \mathcal{T}$ with unbalanced subtrees (unbalanced meaning that the number of leaves in its left and right subtree are not equal) and the number of internal vertices with balanced, but non isomorphic subtrees, respectively. Similarly, let $X(s)$ and $Y(s)$ denote the number of such vertices for $s \in \mathcal{S}$. Clearly, for any $t \in \mathcal{T}$, $s \in \mathcal{S}$ such that $t \sim s$ it is straightforward that $X(t) = X(s)$ and $Y(t) = Y(s)$. Moreover, observe that any structure $s \in \mathcal{S}$ has exactly

$$|[s]| = 2^{X(s)+Y(s)}$$

distinct plane orderings, since each vertex with non isomorphic subtrees can have either one of two different orderings of the subtrees, whereas when both subtrees are isomorphic we have only one ordering.

Now we conclude that

$$\begin{aligned}
&H(T_n|S_n)\\
&= -\sum_{t\in\mathcal{T}_n, s\in\mathcal{S}_n} \mathbb{P}(T_n=t, S_n=s)\log\mathbb{P}(T_n=t|S_n=s)\\
&= \sum_{s\in\mathcal{S}_n, t\sim s} \mathbb{P}(S_n=s)\log|[s]|\\
&= \sum_{s\in\mathcal{S}_n, t\sim s} \mathbb{P}(S_n=s)(X(s)+Y(s))\\
&= \sum_{t\in\mathcal{T}_n} \mathbb{P}(T_n=t)(X(t)+Y(t)) = \mathbb{E}X_n + \mathbb{E}Y_n
\end{aligned}$$

where we write $X_n := X(T_n)$ and $Y_n := Y(T_n)$. We thus need to evaluate $\mathbb{E}X_n+\mathbb{E}Y_n$. It turns out to be easiest to do this by determining the expected value of a random variable Z_n (giving the number of internal vertices with isomorphic child subtrees in a tree of size n) satisfying $Z_n = n-1-(X_n+Y_n)$, but we first determine $\mathbb{E}[X_n]$, which will be useful later, in the proof of Theorem 6.

The stochastic recurrence for X_n can be expressed as follows: $X_1 = 0$ and

$$X_n = X_{U_{n-1}} + X_{n-U_{n-1}} + I\left(U_{n-1} \neq \frac{n}{2}\right)$$

for $n \geq 2$, where U_n is the variable on $\{1,2,\dots,n\}$ with uniform distribution, and the indicator (denoted by $I(\cdot)$) contributes whenever the left and right subtrees of the root are of unequal size. This leads us to the following formula for $\mathbb{E}X_n$:

$$\mathbb{E}X_n = \frac{1}{n-1}\sum_{k=1}^{n-1}\mathbb{E}(X_n|U_{n-1}=k) =$$
$$\frac{1}{n-1}\sum_{k=1}^{n-1}\left(\mathbb{E}X_k + \mathbb{E}X_{n-k} + \mathbb{E}\left(I\left(U_{n-1}\neq\frac{n}{2}\right)|U_{n-1}=k\right)\right)$$
$$= \mathbb{E}I\left(U_{n-1}\neq\frac{n}{2}\right) + \frac{2}{n-1}\sum_{i=1}^{n-1}\mathbb{E}X_i.$$

Clearly,

$$\mathbb{E}I\left(U_{n-1}\neq\frac{n}{2}\right) = \begin{cases}\frac{n-2}{n-1} & \text{if } n\geq 2,\ n \bmod 2 = 0,\\ 1 & \text{if } n\geq 3,\ n \bmod 2 = 1.\end{cases}$$

Using $x_n = \mathbb{E}X_n$ and $a_n = \mathbb{E}I\left(U_{n-1}\neq\frac{n}{2}\right)$, we may apply Lemma 1 to find

$$\mathbb{E}X_n = a_n + n\left(\sum_{k=2}^{n-1}\frac{2}{k(k+1)} - \sum_{k=1}^{\lfloor(n-1)/2\rfloor}\frac{2}{(2k-1)2k(2k+1)}\right)$$
$$= a_n + n\left(1 - \frac{2}{n} - \sum_{k=1}^{\lfloor(n-1)/2\rfloor}\frac{1}{2k-1} - \frac{2}{2k} + \frac{1}{2k+1}\right).$$

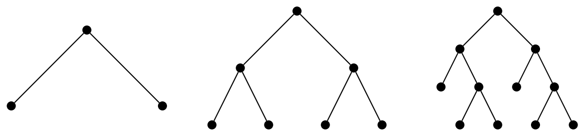


Fig. 5: An example of $\mathfrak{s}_1 * \mathfrak{s}_1$, $\mathfrak{s}_2 * \mathfrak{s}_2$, $\mathfrak{s}_3 * \mathfrak{s}_3$, where $\mathfrak{s}_j$ is, say, the left subtree in the jth pictured tree.

The first term of the summation in the previous expression is equal to $\frac{1}{2k+1}$; furthermore, for any $k\geq 1$, we have $\frac{1}{2k+1} = \frac{1}{2(k+1)-1}$, so that the summation can be written as

$$\left(2\left\lfloor\frac{n+1}{2}\right\rfloor\right)^{-1} + \sum_{k=2}^{\lfloor(n-1)/2\rfloor}\left(\frac{2}{2k-1} - \frac{2}{2k}\right)$$
$$= \left(2\left\lfloor\frac{n+1}{2}\right\rfloor\right)^{-1} + 2\sum_{k=3}^{\lfloor(n-1)/2\rfloor}\frac{(-1)^{k+1}}{k}.$$

This implies

$$\begin{aligned}\mathbb{E}[X_n] &= a_n + n\Bigg(1 - \frac{2}{n} - \left(2\left\lfloor\frac{n+1}{2}\right\rfloor\right)^{-1}\\ &\quad -2\sum_{k=3}^{(n-1)/2}\frac{(-1)^{k+1}}{k}\Bigg)\\ &= 2n(1-\ln 2) + O(1) \approx n\cdot 0.6137.\end{aligned} \tag{13}$$

In order to estimate the entropy we need to introduce another function $Z(t)$: the number of internal vertices of t with isomorphic subtrees. Obviously,

$$X(t) + Y(t) + Z(t) = n-1.$$

Given $\mathfrak{s}\in\mathcal{S}$ we may define $Z(t,\mathfrak{s})$ as the number of internal vertices with both subtrees isomorphic to $\mathfrak{s}$. Clearly, $Z(T_n) = \sum_{\mathfrak{s}\in\mathcal{S}} Z(T_n,\mathfrak{s})$, and $\mathbb{E}Z_n := \mathbb{E}Z(T_n)$.

Let us also use $\mathfrak{s}*\mathfrak{s}$ as a shorthand for a non-plane tree having both subtrees of a root isomorphic to $\mathfrak{s}$. The stochastic recurrence on $Z_n(\mathfrak{s})$ becomes

$$Z_n(\mathfrak{s}) = I\left(T_n\sim\mathfrak{s}*\mathfrak{s}\right) + Z_{U_{n-1}}(\mathfrak{s}) + Z_{n-U_{n-1}}(\mathfrak{s})$$

which leads us to

$$\mathbb{E}Z_n(\mathfrak{s}) = \mathbb{E}I\left(T_n\sim\mathfrak{s}*\mathfrak{s}\right) + \frac{2}{n-1}\sum_{k=1}^{n-1}\mathbb{E}Z_k(\mathfrak{s}).$$

Moreover, since under the condition that $\Delta(T_n^L) = k$ the event that $T_n\sim\mathfrak{s}*\mathfrak{s}$ is equivalent to the intersection of the events $T_n^L\sim\mathfrak{s}$ and $T_n^R\sim\mathfrak{s}$, we have

$$\mathbb{E}I\left(T_n\sim\mathfrak{s}*\mathfrak{s}\right) = \frac{1}{n-1}\sum_{k=1}^{n-1}\mathbb{P}\left(T_n\sim\mathfrak{s}*\mathfrak{s}|U_{n-1}=k\right)$$
$$= I\left(n = 2\Delta(\mathfrak{s})\right)\frac{\mathbb{P}^2(T_{n/2}\sim\mathfrak{s})}{n-1}.$$

Now, we apply Lemma 1 with $x_n = \mathbb{E}Z_n(\mathfrak{s})$ and

$$a_n = I\left(n = 2\Delta(\mathfrak{s})\right)\frac{\mathbb{P}^2\left(T_{n/2}\sim\mathfrak{s}\right)}{n-1}$$

to ultimately find (after some algebra)

$$\mathbb{E}Z_n = n \sum_{k=1}^{\lfloor (n+1)/2 \rfloor} \frac{b_k}{(2k-1)k(2k+1)} + O(1), \qquad (14)$$

where $b_k = \sum_{\mathfrak{s}_k \in \mathcal{T}_k} \mathbb{P}^2(T_k = \mathfrak{s}_k)$ (see also [21] for analytic derivations of (14)). It is easy to compute b_k (see Fig. 5) for a few small values of k, namely

$$b_1 = b_2 = 1, \quad b_3 = \frac{1}{2}, \quad b_4 = \frac{2}{9}, \quad b_5 = \frac{13}{144}, \quad b_6 = \frac{7}{200}. \qquad (15)$$

In fact, in [21] we show that b_n satisfies for $n \geq 2$ the following recurrence

$$b_n = \frac{1}{(n-1)^2} \sum_{j=1}^{n-1} b_j b_{n-j}$$

with $b_1 = 1$.

Using (15) we find $\mathbb{E}Z_n \approx n(0.3725 \pm 10^{-4})$, and therefore

$$H(T_n|S_n) = n - 1 - \mathbb{E}[Z_n] \approx n \cdot 0.6275\ldots$$

Since $H(T_n) - H(S_n) = H(T_n|S_n)$, on average the compression of the structure (the non-plane tree) requires asymptotically $0.6275n$ fewer bits than the compression of any plane tree isomorphic to it. Furthermore, using Theorem 2, we conclude this section with the following result.

Theorem 3. *The entropy rate $h(s) = \lim_{n\to\infty} H(S_n)/n$ of the non-plane trees is*

$$h(s) = h(t) - h(t|s) \approx 1.109\ldots$$

where

$$h(t|s) = 1 - \sum_{k=1}^{\infty} \frac{b_k}{(2k-1)k(2k+1)},$$
$$h(t) = 2\sum_{k=1}^{\infty} \frac{\log_2(k)}{(k+1)(k+2)}$$

with $b_k = \sum_{\mathfrak{s}_k \in \mathcal{T}_k} \mathbb{P}^2(T_k = \mathfrak{s}_k)$.

Remark. The probability $b_k = \sum_{\mathfrak{s}_k \in \mathcal{T}_k} \mathbb{P}^2(T_k = \mathfrak{s}_k)$ is often called the *coincidence probability*, and in fact is related to the Rényi entropy of order 2 for trees that we introduce next. Recall that for general $\alpha \geq 0$, the Rényi entropy of a random variable X supported on some set $\mathcal{X}$ is given by

$$h_\alpha(X) = \frac{1}{1-\alpha} \log \left(\sum_{x \in \mathcal{X}} \mathbb{P}[X = x]^\alpha \right).$$

We will specialize the above to our distribution on trees and $\alpha = 2$. It is relatively easy to check that the following quantity is $\Theta(1)$ as $n \to \infty$:

$$h_2^t(n) = \frac{-\log \sum_{t \in \mathcal{T}_n} \mathbb{P}^2(T_n = t)}{n},$$

and we write $h_2^t = \lim_{n\to\infty} h_2^t(n)$, if the limit exists. Then

$$b_n = \sum_t \mathbb{P}^2(T_n = t) \sim \exp(-n h_2^t)$$

for large n. Actually, using [22] we prove in [21] that

$$b_n \sim 6n \cdot \rho^n$$

where $\rho = 0.3183843834378459\ldots$. Thus $h_2^t = -\log(\rho)$.

IV. Algorithms

The main idea of the algorithms presented here for plane and non-plane trees is to use a version of the arithmetic coding scheme. We start from the interval $[0, 1)$ and we traverse the tree to be compressed: at each vertex and at each letter of the names (if present) we split the current interval into parts depending on some known probability distributions. Then, we pick as a new interval: one that represents the value associated with the current vertex (the number of leaves in the left subtree) or with the current letter of the name.

After we traverse the whole tree, we obtain an interval $[a, b)$. The crucial idea is that any two intervals corresponding to different trees are disjoint and the length of each interval is equal to the probability of generating its underlying plane tree. For plane trees, we present an optimal algorithm running in time $O(n^2 \log^2 n \log\log n)$ (in the worst case). For non-plane trees we present a suboptimal algorithm that runs in $O(n)$ time (if binary numbers can be multiplied in $O(1)$ steps) and an optimal algorithm that runs in $O(n^2)$ time. In this case, we only consider non-plane trees without names to simplify our presentation.

A. *Algorithm* CompressPTree *for plane trees with names*

Here we explain our algorithm for compression of a plane tree with names.

As was mentioned above, first we set the interval to $[0, 1)$ and then traverse the vertices of the tree and its names (we will call this tree with names lt_n in what follows). Here, we traverse the tree in DFS order, first descending to the left subtree and then to the right.

At each step, if we visit an internal node v, we split the interval according to the probability of the number of leaves in the left subtree. That is, if the subtree rooted at v has k leaves and $lt_n(v)^L$ has l leaves, then we split the interval into $k-1$ equal parts and pick l-th subinterval as the new interval. Then if v is the root of lt_n, for every letter of the name $F_n(v)$ we split the interval according to the distribution π and pick the subinterval representing the respective letter of $F_n(v)$. If v is not the root, for every letter of the name $F_n(v)$, we split the interval according to transition probability distribution in P according to the respective letter in $F_n(u)$, where u is the parent of v, and pick the subinterval representing the letter of $F_n(v)$. Finally, when we obtain an interval $[a, b)$, we output the first $\lceil -\log_2(b-a) \rceil$ bits of $\frac{a+b}{2}$. The pseudocode of this algorithm, called CompressPTree is presented next:

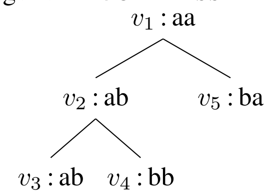
```
function CompressPTree($lt_n$)
    $[a, b) \leftarrow CompressPTreeRec(lt_n)$
```

```
        $p \leftarrow b - a,\ x \leftarrow \frac{a+b}{2}$
        **return** $C_n^{(1)} =$ first $\lceil -\log p \rceil + 1$ bits of $x$
    **function** CompressPTreeRec($t$)
        $v \leftarrow root(t)$
        $l \leftarrow 0, h \leftarrow 1$
        **for** $i = 1, 2, \dots, m$ **do**
            $range \leftarrow h - l$
            $h \leftarrow l + range \sum_{x \in \mathcal{A}:\ x \le F_n(v)_i} P(x|F_n(u)_i)$
            $l \leftarrow l + range \sum_{x \in \mathcal{A}:\ x < F_n(v)_i} P(x|F_n(u)_i)$
        **if** $\Delta(v) \ge 2$ **then**
            $[l_{left}, h_{left}) \leftarrow CompressPTreeRec(t^L)$
            $range \leftarrow h - l$
            $h \leftarrow l + range * h_{left}$
            $l \leftarrow l + range * l_{left}$
            $[l_{right}, h_{right}) \leftarrow CompressPTreeRec(t^R)$
            $range \leftarrow h - l$
            $h \leftarrow l + range * h_{right}$
            $l \leftarrow l + range * l_{right}$
        **return** $[l, h)$
```

Example. Suppose that $\mathcal{A} = \{a, b\}$ with the transition matrix

$$P = \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}. \tag{16}$$

Observe that $\pi = (0.4, 0.6)$. Then for the following tree our algorithm CompressPTree proceeds as follows:



v_1	$\Rightarrow [0.5, 1)$
a	$\Rightarrow [0.5, 0.7)$
a	$\Rightarrow [0.5, 0.58)$
v_2	$\Rightarrow [0.5, 0.58)$
a $\rightarrow$ a	$\Rightarrow [0.5, 0.556)$
$\dots$	

and finally, we obtain $[0.550282624, 0.5507567872)$. We take the middle point of it (0.5505197056) and return its first $\lceil -\log(0.5507567872 - 0.550282624) \rceil + 1 = 13$ bits in the binary representation, that is 1000110011101.

Next we present a proof of correctness and optimality of CompressPTree algorithm for the plane trees.

Theorem 4. *Let lt denote a given plane tree with vertex names. The algorithm* CompressPTree *computes an interval whose length is equal to $\mathbb{P}(LT_n = lt)$.*

Proof. Let $F_n(v)_i$ be the i-the letter of the name associated with vertex v in LT_n. Let also $[a_{lt}, b_{lt})$ be the interval assigned to the tree lt by the algorithm. We want now to prove that $b_{lt} - a_{lt}$ and $\mathbb{P}(LT_n = lt)$ are equal to

$$\prod_{v \in \mathring{V}(lt)} \frac{1}{\Delta(lt(v)) - 1} \prod_{i=1}^{m} P(F_n(r)_i) \prod_{u \to v} \prod_{i=1}^{m} P(F_n(v)_i|F_n(u)_i)$$

where $\mathring{V}(lt)$ is the set of internal vertices of lt, $r = root(lt)$, and the product indexed by $u \to v$ runs over all pairs of vertices (u, v) such that u is a parent of v in lt.

The equality

$$b_{lt} - a_{lt} = \prod_{v \in \mathring{V}(lt)} \frac{1}{\Delta(lt(v)) - 1} \prod_{i=1}^{m} P(F_n(r)_i) \prod_{u \to v} \prod_{i=1}^{m} P(F_n(v)_i|F_n(u)_i)$$

is straightforward from the description of the algorithm: we scale the length of the interval by the $(\Delta(lt(v)) - 1)^{-1}$ for each internal vertex as we split the interval into $\Delta(lt(v)) - 1$ equal parts and pick one of them. Independently, for each letter of the name of the root we scale the length of the interval by the probability of it under a distribution π. Furthermore, for each letter of names of non-root vertices by the conditional probability of it, depending on a transition matrix P and the respective letter for the parent.

The equality

$$\mathbb{P}(LT_n = lt) = \prod_{v \in \mathring{V}(lt)} \frac{1}{\Delta(lt(v)) - 1} \prod_{i=1}^{m} P(F_n(r)_i) \prod_{u \to v \in LT_n} \prod_{i=1}^{m} P(F_n(v)_i|F_n(u)_i)$$

can be proved by induction on n. Clearly, for $n = 1$ the above holds for any m (owing to the horizontal memoryless assumption and the fact that lt consists of a single leaf node): $\mathbb{P}(LT_1 = lt) = \prod_{i=1}^{m} P(F_1(r)_i)$. Assume now, that the equality holds for all $1 \le k < n$ and all possible trees lt of size k with names. Therefore, by the induction assumption, it holds that ($w \in \mathcal{A}^m$ being the name of the root for lt):

$$\mathbb{P}(LT_k = lt | F_k(r) = w) = \prod_{v \in \mathring{V}(lt)} (\Delta(lt(v)) - 1)^{-1} \prod_{u \to v} \prod_{i=1}^{m} P(F_k(v)_i|F_k(u)_i)$$

for any $w \in \mathcal{A}^m$ and any $1 \le k < n$. Then, for a tree lt of size n,

$$\begin{aligned} &\mathbb{P}(LT_n = lt | F_n(r) = w) \\ &= \frac{1}{n-1} \prod_{i=1}^{m} P(F_n(\lambda(LT_n, r))_i | F_n(r)_i) \\ &\prod_{i=1}^{m} P(F_n(\rho(LT_n, r))_i | F_n(r)_i) \\ &\mathbb{P}(LT_n^L = lt^L | F_n(\lambda(lt, r)) = w^L) \\ &\cdot \mathbb{P}(LT_n^R = lt^R | F_n(\rho(lt, r)) = w^R). \end{aligned}$$

Using the induction assumption and the fact that the event $LT_n = lt$ contains also the information that $F_n(r) = w$, we find

$$\begin{aligned} \mathbb{P}(LT_n = lt) &= \mathbb{P}(LT_n = lt, F_n(r) = w) = \\ &= \mathbb{P}(LT_n = lt | F_n(r) = w)\mathbb{P}(F_n(r) = w), \end{aligned}$$

as desired. □

Clearly, if two given plane trees with names $lt, lt' \in \mathcal{LT}_n$ are different, then their respective intervals do not overlap. Indeed, if two trees, lt and lt' are different, then they have different shapes as trees or a different letter in a name for some node. In either case, when the first difference occurs, we proceed into disjoint subintervals for lt and lt'. Further steps can only shrink both intervals – therefore the resulting intervals of both trees are also disjoint.

Therefore, as in the arithmetic coding (see [6]) if we take the first $\lceil -\log \mathbb{P}(LT_n = lt)\rceil + 1 = \lceil -\log(b-a)\rceil + 1$ bits of the middle of the interval $[a, b)$ we find a codeword $C_n^{(1)}$, which, as a binary number, is guaranteed to be inside this interval.

Theorem 5. *The average length of a codeword $C_n^{(1)}$ of* CompressPTree *is only within* 2 *bits from the entropy.*

Proof. From the analysis above, we know that:

$$\mathbb{E}C_n^{(1)} = \sum_{lt\in\mathcal{LT}_n} \mathbb{P}(LT_n = lt)(\lceil -\log_2 \mathbb{P}(LT_n = lt)\rceil + 1) <$$
$$\sum_{lt\in\mathcal{LT}_n} -\mathbb{P}(LT_n = lt)\log_2 \mathbb{P}(LT_n = lt) + 2 = H(LT_n) + 2$$

which completes the proof. □

Finally, we deal with the complexity of the algorithm. In fact, the complexity depends on the model of computation. If we assume that we can multiply two arbitrary numbers in $O(1)$ time, then clearly the worst-case time complexity of the algorithm is $O(n)$. If not, we assume each number is represented by a pair (num, den). Clearly, if we have n leaves, then $0 \le num \le den \le n!$ so we can write num and den on $O(n\log n)$ bits. Therefore, if we denote by $A(n)$ and $T(n)$, respectively, the average-case and worst-case time complexity, then these satisfy

$$\begin{aligned} A(n) &= \frac{2}{(n-1)}\sum_{k=1}^{n-1} A(k) + O(n\log^2 n\log\log n) \\ T(n) &\le \max_k [T(k) + T(n-k)] + O(n\log^2 n\log\log n), \end{aligned}$$

with a base case of $A(1) = O(1)$ and $T(1) = O(1)$. Here, the $O(n\log^2 n\log\log n)$ is a bound on the number of bit operations needed to multiply two $n\log n$-bit numbers (see [28]. Solving these recurrences, we get that the worst-case complexity is $T(n) = O(n^2\log^2 n\log\log n)$, while on average it is $A(n) = O(n\log^3 n\log\log n)$ by Lemma 1.

B. Algorithms for non-plane trees

For non-plane trees without vertex names, we present two algorithms: a suboptimal (in terms of expected code length) compression algorithm called CompressNPTree that runs in worst-case $O(n)$ time, and an optimal algorithm OptNPTree that runs in worst-case $O(n^2)$ time (provided we can multiply two binary numbers in $O(1)$ time).

B1: *Suboptimal but Time-Efficient Algorithm* CompressNPTree

As before, we first set the interval to $[0, 1)$ and then traverse the vertices of the tree. However, now we visit always the smaller subtree (that is, the one with the smaller number of leaves) first (ties are broken arbitrarily).

At each step, if we visit an internal node v, we split the interval according to the probabilities of the sizes of the smaller subtree – that is, if the subtree rooted at v has k leaves and its smaller subtree has l leaves, then if k is even we split the interval into $\frac{k}{2} - 1$ parts of length $\frac{2}{k-1}$ and one interval of length $\frac{1}{k-1}$. Otherwise, k is odd, so we split the interval into $\frac{k-1}{2}$ parts, all of equal length. For example, if $k = 6$, then the subintervals have lengths $\frac{2}{5}$, $\frac{2}{5}$ and $\frac{1}{5}$ of the original interval. If $k = 7$, then the subintervals have length equal to $\frac{2}{6}$ of the original interval. Finally, in both cases we pick l-th subinterval as the new interval. The pseudocode of algorithm is presented below:

```
function CompressNPTree($s_n$)
    $[a, b) \leftarrow CompressNPTreeRec(s_n)$
    $p \leftarrow b - a,\ x \leftarrow \frac{a+b}{2}$
    return $C_n^{(2)}$ = first $\lceil -\log p\rceil + 1$ bits of $x$
function CompressNPTreeRec($s$)
    $l \leftarrow 0, h \leftarrow 1$
    if $\Delta(s) \ge 2$ then
        if $\Delta(s^L) \le \Delta(s^R)$ then
            $w_1 \leftarrow s^L, w_2 \leftarrow s^R$
        else
            $w_1 \leftarrow s^R, w_2 \leftarrow s^L$
        $[l_{left}, h_{left}) \leftarrow CompressNPTreeRec(w_1)$
        $range \leftarrow h - l$
        $h \leftarrow l + range * h_{left}$
        $l \leftarrow l + range * l_{left}$
        $[l_{right}, h_{right}) \leftarrow CompressNPTreeRec(w_2)$
        $range \leftarrow h - l$
        $h \leftarrow l + range * h_{right}$
        $l \leftarrow l + range * l_{right}$
    return $[l, h)$
```

Here, we abuse notation slightly: we assume that the non-plane tree s_n is given as an arbitrary plane-oriented representative. Then s_n^L and s_n^R are defined with respect to this representative.

Next, we present a proof of correctness of the CompressNPTree algorithm for the non-plane trees, together with the analysis of its performance. The algorithm does not match the entropy rate for non-plane trees, since for every vertex with two non isomorphic subtrees of equal sizes, it can visit either subtree first – resulting in different output intervals, so different codewords too. Clearly, such intervals are shorter than the length of the optimal interval (which would be equal to the sum of the lengths of all intervals that can be obtained using CompressNPTree, given a tree s as an input); therefore, the codeword will be longer. However, given $\mathbb{E}Y_n$, the expected redundancy rate is within 1% of the entropy rate as proved in Theorem 6 below.

Lemma 2. *For a given non-plane tree s with exactly $Y(s)$ vertices with balanced but not isomorphic subtrees,* CompressNPTree *computes an interval whose length is equal to $2^{-Y(s)}\mathbb{P}(S_n = s)$.*

Proof. Let $[a_s, b_s)$ be an interval returned by our algorithm, provided its input was s. Then, in fact we want to prove that $b_s - a_s = 2^{-Y(s)}\mathbb{P}(S_n = s)$.

Fix a plane tree $t \sim s$ corresponding to the order of the visited vertices of s so that for any vertex of t, its left subtree is visited before its right subtree. Let its interval (compressed using the algorithm for plane trees) be $[a_t, b_t)$.

Now, observe that $b_s - a_s = 2^{X(s)}(b_t - a_t)$, where $X(s)$ is a number of vertices with unbalanced subtrees in s, because at each step, when we encounter such vertex, we scale the length of the interval twice as much for t when compared to s. When we encounter a vertex with balanced subtree, we scale both equally.

However, we already noted that all plane tree isomorphic to s have the same probabilities and there are $|[s]| = 2^{X(s)+Y(s)}$ many of them. Therefore (knowing that $X(s) = X(t)$ and $Y(s) = Y(t)$ for any $t \sim s$), we have

$$\begin{aligned}\mathbb{P}(S_n = s) &= |[s]|\mathbb{P}(T_n = t) = 2^{X(s)+Y(s)}\mathbb{P}(T_n = t)\\ &= 2^{X(t)+Y(t)}(b_t - a_t) = 2^{Y(s)}(b_s - a_s),\end{aligned}$$

which completes the proof. □

As before, if we use the arithmetic coding scheme on an interval generated from a tree s, we find a codeword $C_n^{(2)}$, which, as a binary number, is guaranteed to be inside this interval. Moreover, we know the following.

Theorem 6. *The average length of a codeword $C_n^{(2)}$ from algorithm* CompressNPTree *does not exceed $1.013H(S_n)$, where $H(S_n)$ is entropy estimated in Theorem 3.*

Proof. From Lemma 2, we know that

$$\begin{aligned}\mathbb{E}C_n^{(2)} &= \sum_{s\in\mathcal{S}_n}\mathbb{P}(S_n = s)\left(\lceil -\log_2\left(2^{-Y(s)}\mathbb{P}(S_n = s)\right)\rceil + 1\right)\\ &< \sum_{s\in\mathcal{S}_n}\mathbb{P}(S_n = s)\left(-\log_2 \mathbb{P}(S_n = s) + Y(s) + 2\right)\\ &= H(S_n) + 2 + \sum_{s\in\mathcal{S}_n}\mathbb{P}(S_n = s)Y(s)\\ &= H(S_n) + 2 + \sum_{t\in\mathcal{T}_n}\mathbb{P}(T_n = s)Y(t) = H(S_n) + 2 + \mathbb{E}Y_n.\end{aligned}$$

When combined with the asymptotic behaviour of Y_n we find, using (13) and (14),

$$\lim_{n\to\infty}\frac{\mathbb{E}Y_n}{n} = \lim_{n\to\infty}\frac{n - 1 - \mathbb{E}X_n - \mathbb{E}Z_n}{n} \le 0.014$$

that leads to

$$\lim_{n\to\infty}\frac{\mathbb{E}C_n^{(2)}}{n} \le \lim_{n\to\infty}\frac{H(S_n)}{n} + \frac{\mathbb{E}Y_n}{n} \le 1.124$$

and

$$\lim_{n\to\infty}\frac{\mathbb{E}C_n^{(2)}}{H(S_n)} \le 1 + \lim_{n\to\infty}\frac{\mathbb{E}Y_n}{H(S_n)} \le 1 + \frac{0.014}{1.109} \le 1.013$$

as needed. □

In passing, we should point out that the running time (both worst- and average-case) is of the same asymptotic order as that of COMPRESSPTree, since the additional step of comparing the sizes of subtrees can be made to incur only an extra cost of $O(n)$ arithmetic operations (comparisons of $\Theta(\log n)$-bit numbers).

B2: *Optimal algorithm for non-plane trees*

In this section we present an optimal compression algorithm for non-plane trees based on arithmetic encoding. In order to accomplish it we need to define a total order among non-plane trees and compute efficiently the probability distribution $\mathbb{P}(S_n < s)$, where $<$ is the order to be defined. Recall again that $\mathcal{S}$ is the set of all non-plane trees and $\mathcal{S}_n$ is the set of all non-plane rooted trees on n leaves. Furthermore, $\Delta(s)$ is the number of leaves of the non-plane tree s.

We start with the definition of our total ordering. In what follows, we will denote by $\mathrm{subtrees}(s)$ the set of subtrees of the tree s rooted at the children of its root.

Definition 1 (Total ordering on the set of non-plane trees)**.** *The relation $<$ on $\mathcal{S}$ is defined as follows: $s_1 < s_2$ if and only if one of the following holds:*

- $\Delta(s_1) < \Delta(s_2)$,
- *or* $\Delta(s_1) = \Delta(s_2)$ *and* $\min\{\mathrm{subtrees}(s_1)\} < \min\{\mathrm{subtrees}(s_2)\}$,
- *or* $\Delta(s_1) = \Delta(s_2)$, $\min\{\mathrm{subtrees}(s_1)\} = \min\{\mathrm{subtrees}(s_2)\}$ *and* $\max\{\mathrm{subtrees}(s_1)\} < \max\{\mathrm{subtrees}(s_2)\}$.

Here, $\min$ and $\max$ are defined recursively in terms of the order relation.

Theorem 7. *The relation $<$ is a total ordering on $\mathcal{S}$.*

Proof. The reflexivity and anti-symmetry are straightforward since either $s_1 < s_2$ or $s_1 = s_2$ or $s_1 > s_2$.

To prove the transitivity, we assume that $s_1 < s_2$ and $s_2 < s_3$. Now, if $\Delta(s_1) < \Delta(s_2)$ or $\Delta(s_2) < \Delta(s_3)$, then $\Delta(s_1) < \Delta(s_3)$ (so $s_1 < s_3$), as from the definition of $<$ we know that $\Delta(s_1) \le \Delta(s_2) \le \Delta(s_3)$.

The only remaining possibility is that $\Delta(s_1) = \Delta(s_2) = \Delta(s_3)$. Then, we proceed similarly: if $\min\{\mathrm{subtrees}(s_1)\} < \min\{\mathrm{subtrees}(s_2)\}$ or $\min\{\mathrm{subtrees}(s_2)\} < \min\{\mathrm{subtrees}(s_3)\}$, then $\min\{\mathrm{subtrees}(s_1)\} < \min\{\mathrm{subtrees}(s_3)\}$ (so $s_1 < s_3$)

since from the definition of $<$ if $\Delta(s_1) = \Delta(s_2) = \Delta(s_3)$, then $\min\{\text{subtrees}(s_1)\} \leq \min\{\text{subtrees}(s_2)\} \leq \min\{\text{subtrees}(s_3)\}$.

Therefore, the only missing case is when $\Delta(s_1) = \Delta(s_2) = \Delta(s_3)$ and $\min\{\text{subtrees}(s_1)\} = \min\{\text{subtrees}(s_2)\} = \min\{\text{subtrees}(s_3)\}$. Since we know that $s_1 < s_2$ and $s_2 < s_3$, this implies that $\max\{\text{subtrees}(s_1)\} < \max\{\text{subtrees}(s_2)\} < \max\{\text{subtrees}(s_3)\}$ by induction, and this completes the proof. □

In what follows, we denote by $\text{less}(s), \text{gtr}(s)$ the minimum/maximum root subtree, respectively, of s under the ordering just introduced. Moreover, we let T_n denote the (random) plane tree from which S_n is generated, and we recall the notation T_n^L and T_n^R for the left and right subtrees of T_n.

The basic plan is to determine an efficient algorithm that, given a non-plane tree s, outputs $\mathbb{P}(S_n < s)$ and $\mathbb{P}(S_n = s)$. This will allow us to construct an arithmetic coding scheme as follows: we associate to s the half-open interval $[a, b)$, whose left endpoint is given by $\mathbb{P}(S_n < s)$ and whose right endpoint is $\mathbb{P}(S_n \leq s)$. The length of this interval is clearly $\mathbb{P}(S_n = s)$, and, because of our total ordering on non-plane trees, for two trees $s_1 < s_2$, the right endpoint of s_1 is less than or equal to the left endpoint of s_2. That is, the two intervals do not overlap. Having this interval in hand, we take the midpoint and truncate its binary representation as usual. This will give a uniquely decodable code whose expected length is within 2 bits of the entropy $H(S_n)$.

Now we are in a position to derive the probabilities $\mathbb{P}(S_n = s)$ and $\mathbb{P}(S_n < s)$.

First, we derive an expression for $\mathbb{P}(S_n = s)$. In the case where s has two non-equal subtrees, we have

$$\begin{aligned}\mathbb{P}(S_n = s) &= \mathbb{P}(\text{less}(S_n) = \text{less}(s), \text{gtr}(S_n) = \text{gtr}(s))\\ &= \mathbb{P}(T_n^L \sim \text{less}(s), T_n^R \sim \text{gtr}(s))\\ &\quad + \mathbb{P}(T_n^L \sim \text{gtr}(s), T_n^R \sim \text{less}(s))\end{aligned} \tag{17}$$

where we recall $\sim$ denotes isomorphism. To calculate the first term on the right-hand side, we condition on the number of leaves in the left subtree of T_n taking the correct value:

$$\begin{aligned}&\mathbb{P}(T_n^L \sim \text{less}(s), T_n^R \sim \text{gtr}(s))\\ &= \mathbb{P}(\Delta(T_n^L) = \Delta(\text{less}(s)))\\ &\quad \cdot \mathbb{P}(T_n^L \sim \text{less}(s), T_n^R \sim \text{gtr}(s)) | \Delta(T_n^L) = \Delta(\text{less}(s)))\\ &= \frac{1}{(n-1)} \cdot \mathbb{P}(S_{\Delta(\text{less}(s))} = \text{less}(s)) \cdot \mathbb{P}(S_{\Delta(\text{gtr}(s))} = \text{gtr}(s)).\end{aligned}$$

Here, we have applied the conditional independence of the left and right subtrees of T_n given the number of leaves in each. It turns out that the second term of (17) is equal to the first, so we get in this case

$$\begin{aligned}&\mathbb{P}(S_n = s)\\ &= \frac{2}{(n-1)} \cdot \mathbb{P}(S_{\Delta(\text{less}(s))} = \text{less}(s)) \cdot \mathbb{P}(S_{\Delta(\text{gtr}(s))} = \text{gtr}(s)).\end{aligned}$$

In the case where the two subtrees of s are identical, only a single term in (17) is present, and it evaluates to

$$\begin{aligned}&\mathbb{P}(S_n = s)\\ &= \frac{1}{n-1} \cdot \mathbb{P}(S_{\Delta(\text{less}(s))} = \text{less}(s)) \cdot \mathbb{P}(S_{\Delta(\text{gtr}(s))} = \text{gtr}(s))\\ &= \frac{1}{n-1} \cdot \mathbb{P}^2(S_{\Delta(\text{less}(s))} = \text{less}(s)).\end{aligned}$$

Thus, in each case, we have derived a formula for $\mathbb{P}(S_n = s)$ that may be recursively computed with $O(n)$ arithmetic operations in the worst case.

It remains to derive an expression for $\mathbb{P}(S_n < s)$. We again first consider the case where s has two non-identical subtrees. Following the definition of $<$, this event is equivalent to

$$\begin{aligned}&[\Delta(\text{less}(S_n)) < \Delta(\text{less}(s))]\\ &\cup \Big[[\Delta(\text{less}(S_n)) = \Delta(\text{less}(s))] \quad \cap [\text{less}(S_n) < \text{less}(s)]\Big]\\ &\cup \Big[[\Delta(\text{less}(S_n)) = \Delta(\text{less}(s))] \quad \cap [\text{less}(S_n) = \text{less}(s)]\\ &\qquad\qquad\qquad\qquad \cap [\text{gtr}(S_n) < \text{gtr}(s)]\Big].\end{aligned} \tag{18}$$

The terms of this union are disjoint, so the total probability is the sum of the probabilities of the individual intersection events.

The first event is equivalent to the union of the disjoint events that the left subtree of T_n has $< \Delta(\text{less}(s))$ leaves or more than $n - \Delta(\text{less}(s))$ leaves. The probability of this event is thus

$$\mathbb{P}(\Delta(\text{less}(S_n)) < \Delta(\text{less}(s))) = 2 \cdot \frac{\Delta(\text{less}(s)) - 1}{n-1}.$$

The second and third events can be similarly written in terms of disjoint unions of events involving the left and right subtrees of T_n. This gives recursive formulas for their probabilities. Since the formulas are conceptually simple to derive but tedious to write out explicitly, we do not list them, but we mention that at most 4 recursive tree comparison calls are necessary to evaluate $\mathbb{P}(S_n < s)$: we need to know the probability that a tree of the appropriate size is $< \text{less}(s)$, $= \text{less}(s)$, $< \text{gtr}(s)$, and $= \text{gtr}(s)$.

Finally, we compute $\mathbb{P}(S_n < s)$ in the case where the two subtrees of s are equal. This happens if

$$\Delta(\text{less}(S_n)) < n/2$$

or $\Delta(\text{less}(S_n)) = n/2$ and either subtree of S_n is less than the tree s' comprising the two subtrees of s.

The probability of the first event is

$$\begin{aligned}&\mathbb{P}(\Delta(\text{less}(S_n)) < n/2)\\ &= \mathbb{P}(\Delta(T_n^L) < n/2) + \mathbb{P}(\Delta(T_n^L) > n/2)\\ &= 1 - \frac{1}{n-1}.\end{aligned}$$

The probability of the second event may be computed by conditioning: the probability that $\Delta(\mathrm{less}(S_n)) = \Delta(T_n^L) = n/2$ is $\frac{1}{n-1}$, and the probability, conditioned on this event, that either subtree of S_n is less than s' is

$$\begin{aligned}&\mathbb{P}(T_n^L < s' \cup T_n^R < s' | \Delta(T_n^L) = n/2)\\ &= 1 - \mathbb{P}(T_n^L \geq s' | \Delta(T_n^L) = n/2) \cdot \mathbb{P}(T_n^R \geq s' | \Delta(T_n^L) = n/2)\\ &= 1 - (1 - \mathbb{P}(S_{n/2} < s'))^2.\end{aligned}$$

where we have used the conditional independence of the two subtrees of T_n given their sizes.

Thus, the quantities $\mathbb{P}(S_n < s)$ and $\mathbb{P}(S_n = s)$ may be recursively computed. Importantly, all of the involved probabilities are rational numbers, and arithmetic operations are only performed on integers with value at most $O(2^{n^2})$, so that the interval corresponding to s in our arithmetic coding scheme is exactly computable in polynomial time.

With these results in hand, the aforementioned arithmetic coding scheme, in which the interval corresponding to a tree $s \in \mathcal{S}_n$ is $[\mathbb{P}(S_n < s), \mathbb{P}(S_n \leq s))$, which we call OPT-NPTREE, yields the following optimal compression result:

Theorem 8. *The expected code length of the algorithm* OPT-NPTREE *is at most $H(S_n) + 2$ bits.*

Finally, we analyze the running time of the natural recursive algorithm for computing the left and right endpoints of the interval for s. We note that, to use the recursive formulas for the probabilities, we need to know $\mathrm{less}(s)$. To facilitate this, we can first construct a *canonical representative* plane tree t isomorphic to s, in which for each internal node of t, the left subtree is less than or equal to the right one. Note that the left and right subtrees of t are then canonical representations of $\mathrm{less}(s)$ and $\mathrm{gtr}(s)$, respectively. To construct such a tree from an arbitrary plane representative of $s \in \mathcal{S}_n$ can be done recursively, as the following algorithm shows:

```
function CANONIZE ($t_n$)
▷ Base case
    if $n \leq 2$ then return $t_n$
▷ Recursive case
    CANONIZE ($t_n^L$)
    CANONIZE ($t_n^R$)
    if $t_n^L > t_n^R$ then
        swap $t_n^L$, $t_n^R$
    return $t_n$
```

When testing whether or not $t_n^L > t_n^R$, we can take advantage of the fact that both subtrees have already been canonized. This saves some time in determining which subtree of each subtree is the lesser one. Nonetheless, the number of comparisons needed to compare two trees is $O(n)$ in the worst case.

To analyze the CANONIZE algorithm, denote by $C(n)$ the number of integer comparisons used by the algorithm on a given tree t. It satisfies the recurrence

$$C(n) = C(\Delta(t^L)) + C(\Delta(t^R)) + O(n),$$

with a base case of $C(1) = O(1)$. The solution of this recurrence is $O(n^2)$ in the worst case, but on average is $O(n \log n)$.

Having a canonical representative of s in hand, calculating the left and right endpoints of the corresponding interval takes $\Theta(1)$ arithmetic operations at each recursive step (and, thanks to the canonical representation, deciding which subtree is the lesser one takes $\Theta(1)$ time), plus $O(n)$ operations to evaluate the probability $\mathbb{P}(S_{\Delta(\mathrm{less}(s))} = \mathrm{less}(s))$. Thus, if we denote by $T(n)$ the number of arithmetic and tree comparison operations to determine the left and right endpoints of the interval for a canonical representation t with n leaves, we have

$$T(n) \leq T(\Delta(t^L)) + T(\Delta(t^R)) + O(n),$$

with a base case of $T(1) = O(1)$. Solving this, we find that, in the worst case, $T(n) = O(n^2)$. Thus, in total, the number of arithmetic and tree comparison operations, including the construction of the canonical representation, is at most $\Theta(n^2)$ in the worst case.

Now we refine the analysis by taking into account the time taken by the arithmetic operations. In the calculation of each interval endpoint, we must keep track of an integer numerator and denominator. In each step, the numerator and denominator are both at most $\Theta(2^{n^2})$, so each may be represented exactly with $\Theta(n^2)$ bits. Each arithmetic operation between two such numbers takes at most $O(n^2 \log n \log\log n)$ bit operations (since multiplying two N-digit numbers takes at most $O(N \log N \log\log N)$ operations). Furthermore, taking into account the lengths of the involved integers, the algorithm for computing $\mathbb{P}(S_{\Delta(s')} = s')$ takes time $O(n^2 \log^2 n \log\log n)$ (since $\Delta(s') < n$, and an easy inductive proof shows that the lengths of the integers required for the calculation never exceed $O(n!)$). Then the recurrence for the running time $T(n)$ becomes

$$T(n) = T(\Delta(t^L)) + T(\Delta(t^R)) + O(n^2 \log^2 n \log\log n),$$

again with a base case of $T(1) = O(1)$. Solving this, we get that

$$T(n) = O(n^3 \log^2 n \log\log n).$$

Since this is asymptotically larger than the time taken to construct the canonical representation, the worst-case total running time is $O(n^3 \log^2 n \log\log n)$. The average-case running time can similarly be shown to be $O(n^2 \log^2 n \log\log n)$.

V. CONCLUSION

In this paper we first studied binary plane trees with correlated names and its entropy – which gives the fundamental lower bound on the compression rate – and finally designed an algorithm achieving this bound within an additive constant number of bits in expectation. We also derived the more challenging entropy for non-plane trees and designed an optimal (in terms of expected code length) $O(n^2)$- time algorithm, as well as a suboptimal $O(n)$-time algorithm (in a model in which arithmetic operations can be done in $O(1)$ time).

The future directions may focus on construction of universal compression schemes for the presented models (e.g., in the setting where the transition matrix for the names is not known). One may also introduce some horizontal dependency between the letters of the names, using for example mixing sources (noting that a Markov horizontal dependency would be a trivial extension of the results of the present paper). Very recent work [15] concentrates on another nontrivial extension, namely to d-ary recursive trees (described in [3]), in which each internal vertex has exactly d children.

Finally, as argued in the introduction, our broader goal is to propose adequate models, bounds and algorithms for a wider class of structures, for example random graphs with vertex names.

References

[1] Ashburner et al., Gene ontology: tool for the unification of biology, *Nat. Genet.*, 2000, 25(1):25–9.

[2] The Gene Ontology Consortium. Gene ontology consortium: going forward, *Nucl. Acids Res.*, 2015, 43, Database issue D1049–D1056.

[3] M. Drmota, *Random Trees: An Interplay between Combinatorics and Probability*. Springer Publishing Company, Inc., 2009.

[4] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*. John Wiley & Sons, Inc., New York, NY, 2001.

[5] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*. Addison-Wesley Publishing Co., Inc., Reading, MA, 1996.

[6] T. Cover, J. Thomas, *Elements of Information Theory*, Second Edition, John Wiley & Sons, Inc., Hoboken, NJ, 2005.

[7] M. Naor, Succinct representation of general unlabeled graphs, *Discrete Applied Mathematics*, 28(3), 303–307, 1990.

[8] Yongwook Choi, Wojciech Szpankowski: Compression of Graphical Structures: Fundamental Limits, Algorithms, and Experiments. *IEEE Transactions on Information Theory*, 2012, 58(2):620-638.

[9] M. Mohri, M. Riley, A. T. Suresh, Automata and graph compression. *ISIT 2015*, pp. 2989-2993.

[10] B. Guler, A. Yener, P. Basu, C. Andersen, A. Swami, A study on compressing graphical structures. *GlobalSIP 2014*, pp. 823-827

[11] Gy. Turan, On the succinct representation of graphs, *Discrete Applied Mathematics*, 8(3), 289–294, 1984.

[12] D. Aldous and N. Ross, Entropy of Some Models of Sparse Random Graphs With Vertex-Names. *Probability in the Engineering and Informational Sciences*, 2014, 28:145-168.

[13] J. C. Kieffer, E.-H. Yang, W. Szpankowski, Structural complexity of random binary trees. *ISIT 2009* , pp. 635-639.

[14] J. Zhang, E.-H. Yang, J. C. Kieffer, A Universal Grammar-Based Code for Lossless Compression of Binary Trees. *IEEE Transactions on Information Theory*, 2014, 60(3):1373-1386.

[15] Z. Gołebiewski, A. Magner, W. Szpankowski, Entropy of some general plane trees. To appear in *Proceedings of the IEEE International Symposium on Information Theory*, 2017.

[16] M. Steel, A. McKenzie, Distributions of cherries for two models of trees. *Mathematical Biosciences*, 2000, 164:81-92.

[17] M. Steel, A. McKenzie, Properties of phylogenetic trees generated by Yule-type speciation models. *Mathematical Biosciences*, 2001, 170(1):91-112.

[18] M. Abramowitz, and I. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1964.

[19] M. Hassani, Approximation of the dilogarithm function, *J. Inequalities in Pure and Applied Mathematics*, 8, 1-7, 2007.

[20] M. Bona, P. Flajolet, Isomorphism and symmetries in random phylogenetic trees, *Journal of Applied Probability* 46(4):1005-1019.

[21] J. Cichon, A. Magner, W. Szpankowski, K. Turowski, On symmetries of non-plane trees in a non-uniform model. *Proceedings of the Fourteenth Workshop on Analytic Algorithmics and Combinatorics*, 2017, 156–163.

[22] H-H Chern, M. Fernández-Camacho, H-H. Hwang, and C, Martinez, Psi-series method for equality of random trees and quadratic convolution recurre nces, *Random Structures & Algorithms*, 44, 67-108, 2012.

[23] L. Peshkin, Structure induction by lossless graph compression, *In Proc. of the IEEE Data Compression Conference*, 53–62, 2007.

[24] M. Adler, M. Mitzenmacher, Towards Compressing Web Graphs, *Data Compression Conference* 2001, pp. 203-212.

[25] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On Compressing Social Networks, *Proc. ACM KDD*, 2009.

[26] S. J. Matthews, S.-J. Sul, T. L. Williams, TreeZip: A New Algorithm for Compressing Large Collections of Evolutionary Trees, *Data Compression Conference* 2010, pp. 544-554.

[27] J. Sun, E.M. Bollt, and D. Ben-Avraham, Graph compression–save information by exploiting redundancy, *Journal of Statistical Mechanics: Theory and Experiment*, P06001, 2008.

[28] A. Schönhage and V. Strassen, Schnelle multiplikation großer zahlen, *Computing*, 7:281, 1971.