TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DIPLOMARBEIT

Tree Compression

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Nadja Azzouz
Matrikelnummer 00471382

ausgeführt am Institut für Diskrete Mathematik und Geometrie
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuer: Ao. Univ. Prof. Dr. Bernhard Gittenberger

Wien, 14.05.2025

(Unterschrift Verfasserin) (Unterschrift Betreuer)

Contents

1 Introduction

In computer science, data structures are crucial to store, represent, and process data in ways that preserve critical information, such as relationships within the data. The choice of data structure depends on the specific characteristics of the data and the intended application, with different structures offering distinct advantages based on how the data is perceived and utilized.
Among the most foundational and widely used data structures are trees, which are particularly effective for hierarchically organizing sequential data. For instance, trees are commonly used in file systems, where directories and subdirectories form a hierarchical structure, enabling efficient storage and retrieval of files. The use of tree data structures necessitates their compression to enable efficient processing while they occupy main memory.
Trees come in various forms, including plane and non-plane, labeled and unlabeled, as well as degree-restricted and unrestricted variants. This thesis focuses on the major compactification methods for the class of rooted plane trees. All of those considered share a common structure: a root node at the top from which a collection of subtrees dangle. Each subtree, in turn, consists of its own root and a further collection of subtrees. This recursive characterization of trees plays a fundamental role in their combinatorial analysis, as we will see.
Tree data structures have proven to be more efficient in searching and returning data compared to lists and arrays and offer the possibility of maintaining order ("locational logic") when compared to hash tables. In addition, they are flexible in size and therefore easy to edit and navigate.
In order to further advance in performance and scalability, we need to optimize tree compression such that three main aspects are targeted:

- minimizing memory overhead
- efficient support for tree operations
- preservation of structural order

The development and evaluation of tree compressions will be strongly based on the underlying combinatorial structure of the trees at hand with auxiliary use

of probabilistic and asymptotic methods from information theory, which will be elucidated in this work. The major compression methods in lossless data compression for trees, as listed in [23], are:

- **minimal DAG compression**: The minimal DAG corresponding to a tree is the directed acyclic graph with the fewest vertices among all DAGs created by identifying subtree repeats and merging identical instances. This way of reducing redundancy can be done in linear time, allowing for operations on the compressed version to be conducted in logarithmic time. DAG compression is one of the oldest tree compression methods, which dates back to early studies in computer science. They were already established in the 1950s as a means to reduce memory usage in structures with repeated elements [15]. DAG compression works well for fringe-dominated trees - those with many repeated subtrees. However, the main disadvantage is that it does not make use of repeated patterns within the tree structure. This shortcoming is avoided in:

- **top-tree compression**: Introduced by Bille et al. [1] in 2015 this technique gives an exponential improvement on compression when compared to DAG minimization. Their compression algorithm works by splitting a tree into clusters of which each becomes a representative meta-node. The meta-nodes get then connected to another, symbolized by an edge, according to the parent-child relationship of the respective subtrees in the original tree. This offers a $\log_{\sigma}^{0.19}(n)$ worst-case compression ratio for a tree of size n labeled from an alphabet of size σ. Additionally, navigation and a number of other operations are supported in $\mathcal{O}(\log n)$ time directly on the compressed representation.

- **tree grammar compression**: Another promising alternative to minimal DAG compression which can perform exponentially better is given by generalizing grammar compression from strings to trees. Unfortunately, it is NP-hard to find a tree's smallest grammar, even over a finite alphabet, leading to the requirement of profitable heuristics, [5]. One of such is RePair which is not applied to the tree directly but rather its string representation. The extended version, TreeRePair, which is applied directly to trees, is not yet developed enough to support navigation in sublinear time.

- **succinct compression**: Unlike the previously introduced compression algorithms, succinct compression does not compress the structure of the tree. Instead, this compression method aims to achieve optimal information-theoretic limits in terms of storage space while still allowing efficient queries on the compressed representation. Grammar-based and Top-DAGs compression typically achieve logarithmic query times in the Word-RAM model for supported

navigational queries. In contrast, some succinct compression methods achieve constant query times, [33].

In this thesis we will focus on minimal DAG and entropy-based compression methods. We will also cover one example of a grammar-based compression code. At the end, we also highlight a much more refined version of succinct compression called hypersuccinct compression that was developed by Munro et al. [31] in 2021.
The main advantage this new compression offers, besides achieving different information-theoretic optima on entropy, is that the hypersuccinct encoding is equipped with an automated adaption of the encoding to a wide range of tree sources. Therefore, it extends the information-theoretic idea of universal coding from strings to trees in a revolutionary universal way for binary and plane tree sources.

> We present a new universal source code for distributions of unlabeled binary and ordinal trees that achieves optimal compression to within lower order terms for all tree sources covered by existing universal codes. At the same time, it supports answering many navigational queries on the compressed representation in constant time on the word-RAM; this is not known to be possible for any existing tree compression method. The resulting data structures, "hypersuccinct trees", hence combine the compression achieved by the best known universal codes with the operation support of the best succinct tree data structures.
>
> *Munro et al.* [31]

1.1 Preliminaries on combinatorics

In order to determine the likelihood of a tree occurring — and therefore the amount of information it shall carry — we need to know how many different trees there are for each possible size. The main tool to derive this counting (or probability) sequence $(a_n)_{n\in\mathbb{N}}$, which gives for each n the number of distinct trees in our structure of size n, is the generating function. A generating function is a formal power series whose coefficients correspond to the terms of the counting sequence. Coating the counting sequence of interest in this way turns out to be advantageous: It enables us to translate all kinds of combinations of tree structures such as addition, sequencing, or intertwining trees according to a specific pattern directly into operations on their generating functions. This, in turn, allows us to derive the counting sequence of the newly constructed object more or less straightforwardly. The content of this subsection contains a selection from [18].

Generating functions and models

Definition P1. A **binary tree** is a rooted plane tree in which each node has two or zero successors. We call these internal and external nodes, respectively. The **size** of a binary tree is the number of internal nodes. Note that a binary tree with n internal nodes has $n+1$ external nodes. For $n=0$ there is a single binary tree consisting of a root only considered as an external leaf. For $n=1$ we have exactly one tree as well with the root as an internal node and two leaves attached.

Remark P2. Let $\mathcal{Z}$ be the symbol for an internal node, and $\square$ the symbol for an external node. The class $\mathcal{B}$ of binary trees has a recursive characterization that yields the functional equation of its generating function:

$$\mathcal{B} = \square + \mathcal{Z}\times\mathcal{B}\times\mathcal{B} \implies B(z) = 1 + zB(z)^2 \iff B(z) = \frac{1-\sqrt{1-4z}}{2z}.$$

Note that $B(z)$ is analytic in the open disc $|z| < \frac{1}{4}$, since the function $\sqrt{1-x}$ is analytic for $|x|<1$. The coefficients of the generating sequence are given by the famous Catalan numbers

$$B_n = \frac{1}{n+1}\binom{2n}{n}$$

with the known asymptotics for the coefficients $B_n \approx \frac{4^n}{\sqrt{\pi n^3}}$ derived by Stirling's formula for $n!$. Furthermore we have the recurrence relation

$$B_0 = 1,\ B_n = \frac{2(2n-1)}{n+1}B_{n-1}, \quad n\geq 1 \tag{1.1}$$

derived by

$$B_n = \frac{(2n)!}{(n+1)!n!} \implies \frac{B_n}{B_{n-1}} = \frac{2(2n-1)}{n+1}, \quad n\geq 1.$$

Definition P3. Let ε be the symbol for the empty tree. An **incomplete binary tree** is a rooted plane tree where each node has no, a single left/right or two children. The size is defined as the number of nodes. This uniquely corresponds to a binary tree in which only internal nodes are considered. Let $\hat{\mathcal{B}}$ be the class of incomplete binary trees. Then

$$\hat{\mathcal{B}} = \mathcal{Z} \times (\varepsilon + \varepsilon \times \hat{\mathcal{B}} + \hat{\mathcal{B}} \times \varepsilon + \hat{\mathcal{B}}^2) \implies \hat{B}(z) = \frac{1 - 2z - \sqrt{1 - 4z}}{2z}.$$

Definition P4. A **Catalan tree** is an unlabeled rooted plane tree in which each node has zero or more successors. The size of a Catalan tree is the number of nodes (including the root). The number of distinct Catalan trees of size n corresponds to the $(n-1)$-th Catalan number. This is due to the rotational correspondence, a natural bijection between binary and Catalan trees. For $n = 1$, there is a single Catalan tree consisting of a root only. For $n = 2$, there is a single Catalan tree consisting of a root and one child node.
The recursive characterization and functional equation are given by

$$\mathcal{C} = \mathcal{Z} \times \mathrm{Seq}(\mathcal{C}) \implies C(z) = \frac{z}{1 - C(z)}$$

where $\mathrm{Seq}(\mathcal{C}) = \{\epsilon\} + \mathcal{C} + \mathcal{C}^2 + \mathcal{C}^3 + \ldots$ denotes a sequence of arbitrary length.

Constructions

Definition P5. Let t be a tree with n nodes. We call t a **labeled tree** if each node $v_i,\ i = 1, n$, of t carries a unique label from the set $\{1, \ldots, n\}$. We call t an **m-labeled tree** if the labels are not required to be unique for each node and belong to the set $\{1, \ldots, m\}$.

Let $\mathcal{A}, \mathcal{B}$ be unlabeled combinatorial structures, $|\cdot|_{\mathcal{A}}, |\cdot|_{\mathcal{B}}$ measure their respective object sizes, and $A(z), B(z)$ the associated generating functions with coefficients A_n, B_n. In addition, let $\hat{\mathcal{A}}, \hat{\mathcal{B}}$ denote their labeled versions.

Definition P6. The **cartesian product** $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ forms ordered pairs (a, b) with $a \in \mathcal{A}$ and $b \in \mathcal{B}$ of size $|a|_{\mathcal{A}} + |b|_{\mathcal{B}}$ where the coefficients and the generating function of the product are given by

$$C_n = \sum_{k=0}^{n} A_k B_{n-k} \quad \text{and} \quad C(z) = A(z) \cdot B(z)$$

Definition P7. Pointing: Let $\{\epsilon_1, \ldots, \epsilon_n\}$ be a collection of n distinct objects of size 0 (pointers). Then $\Theta\mathcal{A}$ describes the structure in which in each object one atom of size 1 is pointed at.

$$\Theta\mathcal{A} := \sum_{n \geq 0} A_n \times \{\epsilon_1, \ldots, \epsilon_n\} = z \frac{\partial}{\partial z} A(z) \implies \Theta A_n = n A_n$$

Definition P8. The **boxed product** $\hat{\mathcal{C}} = \hat{\mathcal{A}}^{\square} \star \hat{\mathcal{B}}$ forms ordered pairs $(\hat{a}, \hat{b})$ with $\hat{a} \in \hat{\mathcal{A}}$ and $\hat{b} \in \hat{\mathcal{B}}$ where the smallest label always lies in the first component $\hat{a}$. The exponential generating function (EGF) $\hat{C}(x)$ is then given by

$$\hat{C}(x) = \int_0^x \frac{\mathrm{d}}{\mathrm{d}t} \hat{A}(t)\hat{B}(t)\,\mathrm{d}t$$

whereas the coefficients are given by

$$\hat{C}_n = \sum_{k=0}^{n} \binom{n}{k} (k\hat{A}_k)\hat{B}_k$$

Lemma P9. ***(Combinatorial interpretation of Taylor's formula.)***
In accordance with the pointing definition above, the operator $\frac{1}{k!}\frac{\partial^k}{\partial x^k}$ corresponds to all variations of pointing at k distinct objects (disregarding order) marked by x and replacing them with an object of size zero. Taylor's formula

$$f(x+y) = \sum_{k \geq 0} \left(\frac{1}{k!} \frac{\partial^k}{\partial x^k} f(x) \right) y^k \tag{1.2}$$

can be combinatorially interpreted to give the bicolored version of a given structure $\mathcal{F}$ (enumerated by f) by $f(x+y)$. Each object can be painted in x-color or y-color. Taylor's formula expresses that this is equivalent to choosing k objects from x and repainting them to y.

Theorem P10. ***(Inclusion-Exclusion principle.)***
Let $A_1, \ldots, A_n$ be a collection of finite subsets of a set A. Then the number of elements in the union of these sets are given by

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} \left| \bigcap_{j \in I} A_j \right| \tag{1.3}$$

The key idea to get the exact amount of numbers in the union by first overcounting, then correcting the overcount, but deducting too much, correcting this and so on.

Theorem P11. ***(Lagrange Inversion Theorem.)***
Let the formal power series $y(z) = \sum_{n \geq 1} y_n z^n$ be defined implicitly by the functional equation

$$y(z) = z \cdot f(y(z))$$

where $f(u) = \sum_{k \geq 0} f_k u^k$ is a formal power series with $f_0 \neq 0$. Then the coefficients y_n of $y(z)$ are given by

$$y_n = [z^n]y(z) = \frac{1}{n}[u^{n-1}](f(u))^n \quad \text{for } n \geq 1.$$

Here, $[z^n]g(z)$ denotes the coefficient of z^n in the series $g(z)$.

Study of asymptotic behavior

For the asymptotic behavior of the coefficients of a generating function, the location and type of so-called dominant singularities — i.e., the singularities on the circle of convergence — are crucial. Below we present a selection of tools for analyzing such.

Lemma P12. ***(Cauchy's coefficient formula.)***
Let $f(z)$ be analytic in a region Ω containing 0 and let γ be a simple loop around 0 in Ω that is positively oriented. Then, the coefficient $[z^n]f(z)$ admits the integral representation

$$[z^n]f(z) = \frac{1}{2\pi i}\int_\gamma f(z)\frac{\mathrm{d}z}{z^{n+1}}.$$

Proof. The proof is provided in [18] and relies on Cauchy's integral formula. ■

Theorem P13. ***(Hankel's formula for the Gamma function.** , [18, p.745])*
Let $\int_{+\infty}^{(0)}$ denote an integral along a contour starting at $+\infty$ in the upper half-plane, winding counterclockwise around the origin, and proceeding towards $+\infty$ in the lower half-plane. Then, for all $s \in \mathbb{C}$,

$$\frac{1}{\Gamma(s)} = \frac{i}{2\pi}\int_{+\infty}^{(0)}(-t)^{-s}e^{-t}\,\mathrm{d}t.$$

Theorem P14. ***(Transfer Theorem for Algebraic-Logarithmic Functions.**, [18, p.385])*
Let $\alpha \in \mathbb{C} \setminus \mathbb{Z}_{\leq 0}$ and $\beta \in \mathbb{R}$. Then the n-th coefficient of the function

$$f(z) = (1-z)^\alpha\left(\frac{1}{z}\log\frac{1}{1-z}\right)^\beta$$

asymptotically admits the following expansion

$$[z^n]f(z) \equiv \frac{n^{-\alpha-1}}{\Gamma(-\alpha)}\log^\beta(n)\left(1+\sum_{k\geq 1}\frac{C_k}{\log n}\right)$$

where $C_k = \binom{\beta}{k}\Gamma(\alpha)\frac{d^k}{ds^k}\frac{1}{\Gamma(s)}\Big|_{s=a}$.

Proof. The proof of this theorem can be found in [18, p. 385]. It is based on Cauchy's coefficient formula and the use of Hankel contours, applying Theorem P13. ■

Exactly the same technical sequence of the proof will be employed later in a lemma to prove Theorem 2.8.

1.2 Preliminaries on information theory

The crucial role of information theory in deriving optimal tree compression is based upon its qualitative and quantitative characterization on how much information (code) we need on average to describe a certain event (in our case: tree). Shannon entropy, for example, provides a fundamental lower bound for compression that we aim to approximate with as little redundancy as possible. The central idea in information theory states that the more likely an event is to occur, the less information (in terms of storage space) it should require. For a more detailed explanation of the concepts introduced in the first two sections, the reader is advised to refer to [11], from which the following material originates. The subchapter on tree sources tailors information-theoretic aspects to tree encoding, as introduced in [34].

Shannon entropy

Definition P15. Let X be a discrete random variable with probability distribution $P_X = (p_i)_{i\in\mathbb{N}}$. We call

$$H(P) = \sum_{i\in\mathbb{N}} p_i \log(\frac{1}{p_i}) = -\sum_{i\in\mathbb{N}} p_i \log(p_i) \tag{1.4}$$

the **entropy** of the distribution P_X. For convenience we will refer to this as $H(X)$, the entropy of the random variable. Generally, the choice of logarithm base reflects the size of the output alphabet used in the encoding. Throughout this work, all concrete encoding schemes use binary representations. Consequently, all logarithms are taken to base 2.

Remark P16. Note that $H(X) \geq 0$, since the logarithm of a probability is non-positive. Entropy quantifies the lower bound on the expected amount of information that is required to describe an event considering all possible outcomes.

Definition P17. Let X,Y be two random variables with probability distributions P_X, P_Y. We define the entropy of the pair (X, Y) as

$$\begin{aligned}
H(X,Y) =& -\sum_x \sum_y \mathbb{P}(X=x, Y=y) \log(\mathbb{P}(X=x, Y=y)) \\
=& -\sum_x \sum_y \mathbb{P}(X=x)\mathbb{P}(Y=y|X=x) \log(\mathbb{P}(X=x)\mathbb{P}(Y=y|X=x)) \\
=& -\sum_x \mathbb{P}(X=x) \log(\mathbb{P}(X=x)) \qquad (1.5) \\
& -\sum_x P(X=x) \sum_y \mathbb{P}(Y=y|X=x) \log(\mathbb{P}(Y=y|X=x)) \\
=& H(X) + H(Y|X). \qquad (1.6)
\end{aligned}$$

where $H(Y|X = x) = \sum_y \mathbb{P}(Y = y|X = x) \log(\mathbb{P}(Y = y|X = x))$ is called the relative entropy of Y under X. If X and Y are independent we have

$$H(X, Y) = H(X) + H(Y) \tag{1.7}$$

Remark P18. The joint and relative entropy for more than two random variables is defined analogously. The entropy of a tuple of random variables measures how much information is required to describe the joint occurrence of events from a collection of random variables. Relative entropy indicates how much information is needed, given that the states of some random variables are already known.

Encoding schemes

Definition P19. Let A^* be the set of finite-length sequences from a finite set A, called Alphabet. A **source code** C for a random variable X maps each $x \in \mathcal{X}$, the domain of X, to A^*. The **expected length** of a source code is given by

$$\sum_{x \in \mathcal{X}} p(x) l(x)$$

where $l(x)$ is the length of the codeword $c(x)$. We call a code **optimal** if the average number of code symbols per source symbol can get arbitrarily close to the entropy. For our purposes $\mathcal{X}$ is a binary or plane tree source that will generate a tree according to the respective probability distribution and A^* will consist of all finite-length bit sequences. Due to this, we can index the probabilities in a countable manner as $p_i,\ i \in \mathbb{N}$.

Remark P20. A code is called **prefix-free** if no codeword is the prefix of another. A prefix-free code is instantaneous, meaning that at any position in a message it is clear — without looking at the following letters — whether a codeword ends there. It is not hard to see that a code is instantaneous if and only if it is prefix-free. In the case of a binary alphabet, this is further equivalent to the code originating from a binary tree: The codewords c_i are represented by the leaves in the corresponding binary tree. The codeword corresponding to a leaf is obtained by tracing the path from the root, recording left (0) and right (1) moves along the path, to the leaf. The length of the codeword c_i, called leaf length, is denoted by l_i.

Lemma P21. ***(Kraft's inequality)*** *A binary tree with leaf lengths $l_1, \ldots, l_m$ exists if and only if*

$$\sum_{i=1}^{m} 2^{-l_i} \leq 1.$$

Equality holds if and only if the tree is complete.

Another important result from information theory at this point is that for every uniquely decodable code — which merely requires the *unique decomposition* of a message into codewords — there exists a prefix-free code with the same codeword lengths:

Proof. Let $M = c_1 c_2 \dots c_k$ be a message of length n where c_1 is the initial codeword of length l_1 and $N = c_2 \dots c_k$ the final segment of length $n - l_1$. Because we require unique decodability, the message M is determined by having identified the codewords in N. Hence the number of messages of length n is given by

$$M_n = \sum_{l_i} M_{n-l_i} \leq 2^n, \quad \text{with } M_0 = 1$$

Thus in $0 \leq z < \frac{1}{2}$

$$M(z) = \sum_{n\geq 0} M_n z^n = 1 + \sum_{n\geq 1} \sum_{|l_i|\leq n} M_{n-l_i} z^n = 1 + \sum_{l_i} \sum_{n\geq 1} M_{n-l_i} z^n$$
$$= 1 + \sum_{l_i} \sum_{n\geq 1} M_{n-l_i} z^n = 1 + M(z) \sum_{l_i} z^{l_i} \iff \sum_{l_i} z^{l_i} = 1 - \frac{1}{M(z)} \leq 1$$

which holds true for $z \to \frac{1}{2}$ resulting in the Kraft's inequality. ■

This justifies the restriction to prefix-free codes, which we will use in this section when searching for an optimal compression code, rather than considering uniquely decodable codes in general. The most prominent example of such an (asymptotically) optimal code is the Huffman code:

Definition P22. Let X be a random variable that takes values in a finite set M with a probability distribution $P = (p_1, \dots, p_m)$. The **Huffman tree** is constructed by:

1. If $m = 1$ then the tree consists only of the root. End.
2. Arrange the probabilities: $p_1 \geq \dots \geq p_m$.
3. Merge the smallest probabilities: $p^*_{m-1} = p_{m-1} + p_m$.
4. Construct the optimal tree for $P^* = (p_1, \dots, p_{m-2}, p^*_{m-1})$.
5. Replace leaf $m - 1$ with an internal node having leaves $m - 1$ and m.

The asymptotic optimality of the Huffman code relies on encoding entire blocks of letters of the message at once. For blocks of size n that means constructing a Huffman tree with m^n leaves. The cost of this increases exponentially with n. Instead, we could consider the entire message as a single block, assign it to an interval in $[0, 1]$, and convert it into a (bit-)code that, on average, achieves the entropy up two

two bits. The coding schemes that follow this procedure are known as **arithmetic coding**.

Lemma P23. *Let C be any uniquely decodable source code for a source $\mathcal{X}$. Then, the expected codeword length is lower-bounded by the Shannon entropy $H(\mathcal{X})$.*

Tree source encoding

Definition P24. Let $\mathcal{T}$ a set of trees with a specified size and $\mathcal{T}_n \subseteq \mathcal{T}$ the subset of trees of size n. We call $(\mathcal{T}, \mathbf{P})$ a **tree source** if the random variables $T_n,\ n \in \mathbb{N}$, taking on a tree of size n together with the function $\mathbf{P} : \mathcal{T} \to [0, 1]$ satisfy

$$\sum_{t \in \mathcal{T}_n} \mathbf{P}[T_n = t] = 1, \quad n \in \mathbb{N}. \tag{1.8}$$

Thus, $\mathbf{P}$ induces a probability distribution over each set $\mathcal{T}_n$. In the following, we will refer to $\mathcal{T}$ as a tree source itself, rather than the tuple $(\mathcal{T}, \mathbf{P})$ and abbreviate $\mathbf{P}(T_n = t)$ as $\mathbf{P}(t)$. If not declared otherwise $\mathbf{P}$ is set to be the function that gives a uniform distribution.

Definition P25. Without loss of generality assume that the size of $\mathcal{T}$ is defined as the number of nodes. A **binary code** (f, g) **for a tree source** $\mathcal{T}$ consists of tuples (f_n, g_n) for each $\mathcal{T}_n \subseteq \mathcal{T},\ n \in \mathbb{N}$, where:

- $f_n : \mathcal{T}_n \to \{0, 1\}^*$ is the encoding function, mapping the set $\mathcal{T}_n \subseteq \mathcal{T}$ of all possible trees with n nodes (the domain of the random variable X_n) to a set of binary codewords
- $g_n : f_n(\mathcal{T}_n) \to \mathcal{T}_n$ is the decoding function, mapping each codeword back.

We call the code code **lossless** if for each $n \in \mathbb{N}$ the function f_n is injective and g_n is the inverse of f_n.

Definition P26. Given a binary code (f, g) on a tree source $\mathcal{T}$, the **nth order average redundancy** of the code with respect to the source, for each n, is defined as

$$R\left(f, n, \mathbf{P}\right) = \sum_{t \in \mathcal{T}_n,\ P(t) > 0} \frac{l\left(f_n(t)\right) + \log_2 \mathbf{P}(T_n = t)}{n}\ \mathbf{P}(T_n = t)$$

and measures the expected number of bits by which the encoding exceeds the entropy. We call (f, g) **asymptotically optimal** for the tree source $\mathcal{T}$, if

$$\lim_{n \to \infty} R\left(f, n, \mathbf{P}\right) = 0.$$

Definition P27. A code for a family of tree sources $(\mathcal{B}, \mathbf{P})_{\mathbf{P} \in \mathcal{P}}$, where each $\mathbf{P} \in \mathcal{P}$ satisfies (1.8) is called **universal** if it is a lossless and asymptotically optimal code for every source in the family.

1.3 Cauchy-Euler differential equations

In chapter 4.2 we will require background knowledge on solving Cauchy-Euler differential, which is derived from [35, Chapter 4]. Let $y(x)$ be the nth derivative of the unknown function y(x). Then a Cauchy–Euler equation of order n has the form

$$a_n x^n y^{(n)}(x) + a_{n-1} x^{n-1} y^{(n-1)}(x) + \cdots + a_0 y(x) = g(x) \tag{1.9}$$

where $a_1, \ldots, a_n$ are real constants and $x \in \mathbb{R}$. The general solution for the homogeneous part of this ordinary linear differential equation with variable coefficients can be found with the ansatz

$$y = x^\alpha$$

where α is the parameter to be resolved. Substituting this ansatz into (1.9) yields

$$(a_n \alpha^{\underline{n}} + a_{n-1} \alpha^{\underline{n-1}} + \ldots + a_0) x^\alpha = 0$$

Thus, $y = x^\alpha$ is a solution of (1.9) whenever α is a solution of the indicial equation

$$a_n \alpha^{\underline{n}} + a_{n-1} \alpha^{\underline{n-1}} + \ldots + a_0 = 0.$$

In the case of distinct roots $\alpha_1, \ldots, \alpha_n$ the homogeneous solution is given by

$$y_h = \sum_{k=1}^{n} c_k x^{\alpha_k}.$$

The particular solution can be found by variation of constants as

$$y_p(z) = \sum_{i=1}^{n} x^{\alpha_i} \int_0^z c_i'(x)\, \mathrm{d}x$$

where $c_i'(x),\ i = 1, \ldots, n$ depends on the Wronskian determinants of the system and can be solved using Cramer's rule, which is a standard procedure.

2 Minimal DAG compression

Definition 2.1. The **fringe subtree** t_v of a tree t is the subtree rooted at v, containing v and all nodes that can be reached from v by traversing downward edges.

Definition 2.2. The directed acyclic graph (DAG) representation of a tree is a compressed version of the tree in which the nodes of the DAG correspond to *meta-nodes* representing fringe subtrees of the original tree. We refer to the **minimal directed acyclic graph (minimal DAG)** as the DAG representation in which each occurring fringe subtree appearing in the tree is represented by a single meta-node, and multiple occurrences of the same fringe subtree are merged into a single shared instance. The minimality of a DAG is characterized by the smallest possible number of nodes among all DAG representations. The minimal DAG of a tree is unique up to graph isomorphisms, [12]. This compression significantly reduces the space complexity while preserving the hierarchical structure of the original tree.

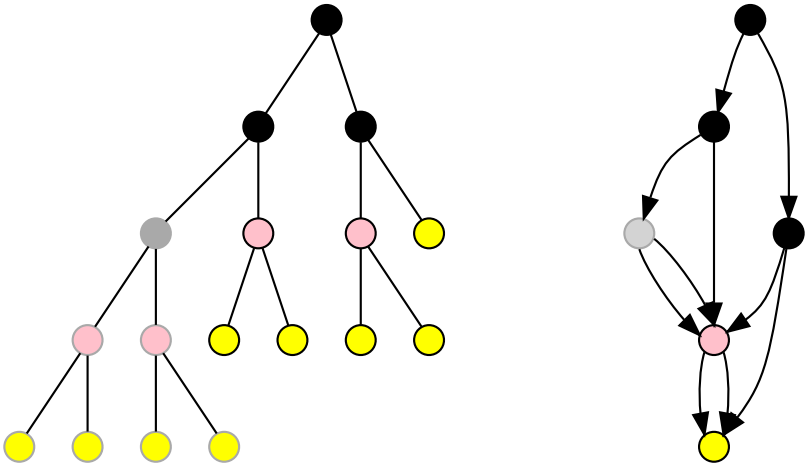


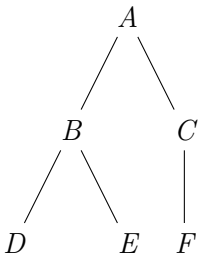
Figure 2.1: A binary tree (left) and its DAG representation (right)

2.1 Computation effort of the minimal DAG

An important consideration in evaluating tree compression algorithms is the computational effort required to construct the compressed representation. We present a linear-time algorithm for constructing the minimal DAG of a labeled rooted plane tree, inspired by the approach in [8]. The algorithm for the (labeled) unrooted plane class was developed one year later by the same authors in [9]. In 2013, Flouri et al. [20] generalized this to the class of non-plane, unrooted (labeled) trees.

Definition 2.3. The **postfix notation** of a labeled rooted plane tree is constructed recursively by traversing each subtree from left to right, denoting occurring labels, and visiting the root last.

Example 2.4. *Consider the following tree:*



Its postfix notation is:

$$\textit{Postfix}\ (T) = D\ E\ B\ F\ C\ A.$$

Note that the postfix notation can be obtained and inversed in $\Theta(n)$ time. This representation forms the basis for the algorithm's preprocessing stage. This algorithm outputs the structure of each unique fringe subtree with the locations where each instance occurs. From this, the minimal DAG of the tree can then be constructed straightforwardly.

Preprocessing: Given is a labeled rooted plane tree t on n nodes in postfix notation. We assume that each node label $x_i \in \Sigma,\ i = 1, \ldots, n$, sets the out-degree (number of children) of that node. Let $\varphi(x_i)$ denote this out-degree. Additionally, the array H, of length n, stores the height of each node given by the maximum number of downward edges until a leaf node is reached.

Overview: The algorithm processes the tree bottom-up where the nodes are indexed according to their position in the postfix notation. We initialize a hash table that maps each subtree sequence to the nodes that they occur on. If a subtree sequence is met twice, it gets added to the subtree repeat list R. The algorithm starts by processing the leaf nodes.

For internal nodes, the algorithm uses a stack-based approach: it pops the children of each node and knows when to stop, given its out-degree (number of children) by φ. This nodes' fringe subtree is stored in postfix notation and generated recursively by concatenating its children subtree sequences and finally appending the current node's own label. The complete sequence is stored in the hash table F. If a sequence appears more than once, it is marked as isomorphic. Two subtrees are isomorphic if and only if their subtrees are structurally identical and carry the same labels at corresponding places.
The dual output of R and F gives both the repeated subtrees and the identification of where every subtree sequence appears. This is useful for applications where both repeat detection and pattern analysis (or tree comparison) are desired.
The algorithm achieves an average time complexity of $\mathcal{O}(n)$ — assuming an efficient hashing scheme F — operations like insertion and lookup take $O(1)$ average time. Computing each fringe subtree sequence from its children's sequences takes time $\mathcal{O}(k)$, where k is the out-degree. The algorithm processes each node once: leaves in $O(1)$ average time and internal nodes with k children in $O(k)$ average time. The total time is $\sum_{l_i \in \text{Leaves}} O(1) + \sum_{n_i \in \text{Internal}} O(\varphi(n_i))$. Since $\sum \varphi(n_i) = n - 1$ for a tree with n nodes, the total complexity is $O(n) + O(n-1) = O(n)$ on average.

Fringe subtree identification algorithm

In line 12 of Algorithm 2 we reverse the order of children popped from the stack to obtain the original ordering in the tree.

Algorithm 1 GetSubtreeSequence

Require: postfix notation $\text{post}(t) = x_1 x_2 \dots x_n$, node index i, children array $Children$, frequency table F

Ensure: sequence seq representing the subtree starting at i

1: $seq \leftarrow$ "" ▷ Initialize empty sequence
2: **for** $j = 1$ to $\varphi(x_i)$ **do** ▷ Process children left to right
3: $c \leftarrow Children[j]$ ▷ j-th child of node i
4: $s \leftarrow F_{\text{rev}}[c]$
5: $seq \leftarrow seq + s$ ▷ Concatenate child sequence
6: **end for**
7: $seq \leftarrow seq + x_i$ ▷ Append root label last
8: **return** seq

Algorithm 2 Subtree Repeat Identifier

Require: tree t with n nodes in postfix notation $\text{post}(t) = x_1 x_2 \dots x_n$ over alphabet $\mathcal{A} = (\Sigma, \varphi)$, array $H[1..n]$

Ensure: Frequency table F, repeated subtrees R

$F \leftarrow [\emptyset],\ R \leftarrow \emptyset$
$stack \leftarrow \emptyset$
for $i = 1$ to n **do** ▷ Build tree and identify subtrees
    $Children \leftarrow []$
    if $H[i] = 0$ **then**
        $F[x_i] \leftarrow F[x_i] \cup \{i\}$
        if $|F[x_i]| \geq 2$ and $x_i \notin R$ **then**
            $R \leftarrow R \cup \{x_i\}$ ▷ Add to repeats if frequency ≥ 2
        end if
        $stack.push(i)$
    else ▷ Internal node
        $Children \leftarrow Reverse([stack.pop() \text{ for } j = 1 \text{ to } \varphi(x_i)])$
        $seq \leftarrow \textsc{GetSubtreeSequence}(\text{post}(t), i, Children)$
        $F[seq] \leftarrow F[seq] \cup \{i\}$
        if $|F[seq]| \geq 2$ and $seq \notin R$ **then**
            $R \leftarrow R \cup \{seq\}$
        end if
        $stack.push(i)$
    end if
end for
return F, R

The function F_{rev} provides the fringe subtrees dangling from each node. Since the leaf nodes have already been processed previously, the algorithm is well-defined.

Navigation and path queries

Buneman et al. [4] demonstrated efficient navigation and path queries on the DAG representation without requiring full initial decompression. By reusing identical subtrees, the XML document's tree structure can be shrunk to less than 10% of the original in practical cases. This allows them to navigate and run queries directly on the compressed XML tree in logarithmic time.

2.2 Compression rate for binary trees

We compute the expected improvement in memory usage for binary trees using the DAG representation as outlined in [19]. For this, our model assumes a uniformly random probability distribution on $\mathcal{B}$. Therefore a tree of size n is selected with probability $\frac{1}{B_n}$.

Definition 2.5. Let t^D denote the DAG representation of a binary tree t and $|t^D|$ the size of it. Let

$$D_n := \sum_{|t|=n} |t^D| \quad \text{such that} \quad \overline{D_n} = \frac{D_n}{B_n}$$

gives the expected size of the compressed tree.

Theorem 2.6. *[19]*

$$\overline{D_n} = \frac{1}{B_n} \sum_{l\geq 1} \sum_{k\geq 1} (-1)^{k-1} \binom{n-kl+1}{k} B_l B_{n-kl}$$

Proof. Since the (meta-)nodes in t^D correspond to distinct fringe subtrees we can calculate D_n by counting all DAGs that have a meta-node representing $u \in \mathcal{B}$. This way, instead of counting nodes per DAG, we sum up one for each DAG that contains a certain fringe subtree as a meta-node. Let $A_{u,n}$ be the number of trees of size n in which the subtree u occurs *at least* once. Then,

$$D_n = \sum_{u\in\mathcal{B}} A_{u,n}.$$

Translating this onto the relationship of the respective generating functions yields

$$D(z) = \sum_{u\in\mathcal{B}} A_u(z). \tag{2.1}$$

In order not to overcount we need to carefully take into account multiple occurrences of a subtree. Let $u \in \mathcal{B}$ be a fixed subtree that appears k-fold, $k \in \mathbb{N}$, in $t \in \mathcal{B}$. Let $v_1, \ldots, v_k,\ k \in \mathbb{N}$, be the nodes that mark the occurrences of $t_{v_i} = u$ for $i = 1, \ldots, k$. The idea is to look at the subtree $t \setminus \{u\}$ where the subtree pattern u is removed and the nodes v_i are treated as leaves. Then we mark those leaves, add u to each of them, resulting in the original tree t. By doing this to all trees that do not contain the pattern u we create all trees containing u at least once and count them exactly once for each respective size $|t \setminus \{u\}| + k|u|$.
We will conduct this approach using Taylor's formula (Lemma P9) and the inclusion-exclusion principle (see Theorem P10).
The bivariate generating function of $\mathcal{B}$ with v marking external nodes is given by

$$B(z, v) = \sum_{n,l\geq 0} B_{n,l} v^l z^n = \sum_{n\geq 0} B_{n,n+1} v^{n+1} z^n = vB(zv) \tag{2.2}$$

since $B_{n,l} = B_n$ for $l = n+1$, 0 otherwise.
Marking k of those leaves by the pointing operation yields the bivariate generating function

$$P(z,v) := \frac{1}{k!}\frac{\partial^k}{\partial v^k}[vB(zv)] = \frac{1}{k!}\sum_{n\geq 0}\frac{(n+1)!}{(n+1-k)!}B_n v^{n+1-k}z^n.$$

The coefficients of the vertical generating function

$$P(z,1) = U_k(z) = \sum_{n\geq 0} U_{n,k}z^n, \quad U_{n,k} = 0 \text{ for } k > n+1$$

now give the number of trees of size n with k distinct leaves marked. For a fixed subtree u of size l we can further derive the generating function of all trees with an (additional) k-fold appearance of u - attached to the marked nodes $v_i,\ i = 1,\ldots,k$ - according to our approach - by

$$z^{kl}P(z,1) = \sum_{n\geq 0} U_{n,k}z^{n+kl}. \tag{2.3}$$

In order to calculate A_u from (2.1) we use an instance of the inclusion-exclusion principle: For $t \in \mathcal{B}$ let $\omega_u[t]$ denote the number of occurrences of u in t. Furthermore, set

$$\alpha_u[t] = \begin{cases} 0, & \omega_u[t] = 0, \\ 1, & \omega_u[t] \geq 1. \end{cases}$$

Then

$$\alpha_u[t] = \sum_{k\geq 1}(-1)^{k-1}\binom{\omega_u[t]}{k}$$

since the sum is either empty in the first case or equal to 1 by the binomial theorem in the second case. Now we see the direct application of the inclusion-exclusion counting (1.3) in place. Since

$$A_u(z) = \sum_{n\geq 0} z^n \sum_{t\in\mathcal{B},\ |t|=n} \alpha_u[t] = \sum_{k\geq 1}(-1)^{k-1}\sum_{t\in\mathcal{B}}\binom{w_u[t]}{k}z^{|t|},$$

together with the fact that the last sum

$$\sum_{t\in\mathcal{B}}\binom{w_u[t]}{k}z^{|t|} = z^{kl}P(z,1)$$

by (2.3) because a tree t with $w_u[t]$ many occurrences of u in it corresponds to

$$\binom{w_u[t]}{k}$$

distinguished ways to produce a k-fold appearance of u on k different marked nodes. Hence,

$$A_u(z) = \sum_{k\geq 1}(-1)^{k-1}z^{kl}P(z,1) = \sum_{k\geq 1}(-1)^{k-1}z^{kl}\frac{1}{k!}\left(\frac{\partial^k}{\partial v^k}B(z,v)\right)\Big|_{v=1}. \tag{2.4}$$

Set $f(x) = xB(zx)$. By Taylor's formula (1.2) with $x = 1$ (leaves) and $y = wz^l$ (u trees, where w quantifies the additional occurrences of u) we get

$$f(1+wz^l) = vB(zv)\Big|_{v=1+wz^l} = \sum_{k\geq 0}\frac{1}{k!}\left(\frac{\partial^k}{\partial v^k}B(z,v)\right)\Big|_{v=1}(wz^l)^k.$$

If we set $v = 1 - z^l$ we obtain

$$-vB(zv)\Big|_{v=1-z^l} = B(z,1) + A_u(z).$$

by comparison with (2.4). This provides

$$A_u(z) = B(z,1) - B(z,1-z^l). \tag{2.5}$$

Hence, by setting $v = 1 - z^l$ in Taylor's formula we remove all occurrences of u since $B(z,1)$ is the generating function for all binary trees. This gives us the generating function of binary trees not containing the subtree u as $B(z,1-z^l)$. Plugging in the explicit formula for $B(z)$ into (2.5), using (2.2), we obtain

$$A_u(z) = \frac{1}{2z}\left(\sqrt{1-4z+4z^{l+1}} - \sqrt{1-4z}\right).$$

Notice that $A_u(z)$ depends only on the size of u. Thus, we substitute it into equation (2.1), but sum up over all possible subtree sizes. To account for this variation, we multiply $A_{u_l}(z)$ - the generating function for an arbitrary subtree u_l of size l - by B_l, the total number of subtrees of size l:

$$D(z) = \sum_{l\geq 0}\frac{1}{2z}B_l\left(\sqrt{1-4z+4z^{l+1}} - \sqrt{1-4z}\right) \tag{2.6}$$

Finally, we derive the coefficients

$$\begin{aligned} D_n &= [z^n]\sum_{l\geq 0}A_{u_l}(z)B_l = [z^n]\sum_{l\geq 0}B_l\sum_{k\geq 1}(-1)^{k-1}\sum_{t\in\mathcal{B}}\binom{w_{u_l}[t]}{k}z^{|t|} \\ &= \sum_{l\geq 0}B_l\sum_{k\geq 1}(-1)^{k-1}\binom{n-kl+1}{k}B_{n-kl}. \end{aligned}$$

where the last equality holds due to the fact that the the tree $t\setminus\{u_l^k\}$ has $n-kl+1$ leaves on which the fringe subtree $u_l \in \mathcal{B}_l$ can be rooted at and because there are B_{n-kl} many possible configurations for the stripped tree $t\setminus\{u_l^k\}$. ■

Theorem 2.7. *[19] The asymptotic expectation on the size of the DAG of a binary tree with n nodes is given by*

$$\overline{D_n} = 2\sqrt{\frac{\log 4}{\pi}}\frac{n}{\sqrt{\log n}}\left(1+\mathcal{O}\left(\frac{1}{\log n}\right)\right).$$

In 2015, Bousquet-Mélou et al. [3] provided a complete proof for the asymptotic behavior initially outlined in [19]. Their result was adapted with respect to the expected number of edges of the minimal DAG for m-labeled binary trees. We will now revisit this proof for the unlabeled case, modifying it to account for binary trees according to our definition.

Theorem 2.8. *[19] The generating function $D(z)$ is analytic in the domain G defined by $|z| < \frac{1}{2}$ and $z \notin \left[\frac{1}{4}, \frac{1}{2}\right]$. For z in the intersection of G and a sufficiently small neighborhood of $\frac{1}{4}$, as $z \to \frac{1}{4}$, one has*

$$D(z) = \frac{2\sqrt{\log 4/\pi}}{\sqrt{(1-4z)\log(1-4z)^{-1}}} + \mathcal{O}\left(\frac{1}{\sqrt{(1-4z)\log^3(1-4z)^{-1}}}\right).$$

To prove Theorem 2.8, we first establish the following auxiliary lemmata:

Lemma 2.9. *[3, p.42] Let $y>0$, and define*

$$a(n,y) := \sum_{l=0}^{n} B_l y^l \tag{2.7}$$

For any $c > \frac{1}{4}$, there exists a neighborhood of c such that

$$a(n,y) = \mathcal{O}((4y)^n n^{-3/2})$$

uniformly in n and y in this neighborhood of c.

Proof. Let $\epsilon, \delta > 0$, $\epsilon < \delta$ and $y \in U_\epsilon(\frac{1}{4}+\delta) \cap \mathbb{R}$. Multiplying (2.7) by x^n and summing up over all $n \in \mathbb{N}$ gives the generating function of the numbers $a(n,y)$ as

$$\sum_{n\geq 0} a(n,y)x^n = \sum_{n\geq 0}\sum_{l=0}^{n} B_l y^l x^n = B(xy)\frac{1}{1-x} = \frac{1-\sqrt{1-4xy}}{2xy(1-x)}$$

implying

$$a(n,y) = [x^n]\frac{1-\sqrt{1-4xy}}{2xy(1-x)} = \frac{1}{2y} - [x^n]\frac{1}{2y}\left(\frac{\sqrt{1-4xy}}{x(1-x)}\right).$$

With Cauchy's coefficient formula (P12) we get the expression

$$[x^n]\frac{\sqrt{1-4xy}}{x(1-x)} = \int_\gamma \frac{\sqrt{1-4xy}}{1-x}\frac{dx}{x^{n+2}},$$

where γ is specified to counterclockwise encircle the origin, small enough so that it does not cross the singularity of the integrand closest to the origin - the *dominant singularity*- which is $x(y) = \frac{1}{4}y^{-1}$.
To confirm that this indeed the dominant singularity, observe that $|4y| > 1$, for this choice of y, and thus the point $x(y) = \frac{1}{4}y^{-1}$ constitutes a branch point of the multi-valued square root function. Substituting $u = 4xy$ we obtain

$$\frac{(4y)^{n+1}}{2\pi i}\int_{\gamma}\frac{\sqrt{1-u}}{4y-u}\frac{du}{u^{n+2}}.$$

We deform the contour integral to a Hankel contour integral and proceed like it is done in [18, p. 382] and illustrated by Figure 2.2. First we extend the contour γ onto

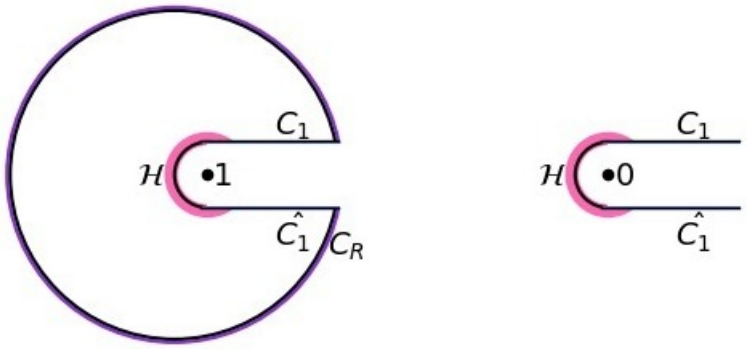


Figure 2.2: keyhole contour(left) that eventually becomes a Hankel contour(right)

a keyhole contour, which is justified by Cauchy's integral theorem. The integrand asymptotically behaves like $(u^n\sqrt{1-u})^{-1}$ and thus vanishes along C_R as the radius R tends to infinity. We are left with the integrals along the arms C_1 and $\hat{C}_1$ and the integral along $\mathcal{H}_n$ that passes at a distance $\frac{1}{n}$ from the real half-line $x \geq 1$. Substituting $u = 1 + \frac{t}{n}$ gives

$$[x^n]\frac{\sqrt{1-4xy}}{x(1-x)} = \frac{(4y)^{n+1}}{n^{\frac{3}{2}}2\pi i}\int_{\mathcal{H}}\frac{\sqrt{-t}}{4y-1-t/n}\left(1+\frac{t}{n}\right)^{-n-2}dt$$

where the Hankel contour $\mathcal{H} \equiv \mathcal{H}(1)$ now winds around 0. Now we distinguish by the size of $\Re(t)$:

1. $\Re(t) \leq \log(n)^2$: For n large enough the value of $4y-(1+t/n)$ in the denominator of the integrand is uniformly lower bounded at some t_0 on the contour $\mathcal{H}$. Furthermore, because of the asymptotic expansion

$$\left(1+\frac{t}{n}\right)^{-n-2} = e^{-t-\frac{2t}{n}+\frac{t^2}{2n}+\frac{t^2}{n^2}-\frac{t^3}{3n^2}+\cdots} = e^{-t}\left(1+\mathcal{O}\left(\frac{t^2}{n}\right)\right) \tag{2.8}$$

and the assumption,

$$t \in \mathcal{O}(\log(n)^2),$$

we can bound the remaining terms of the integrand by

$$\left|\sqrt{-t}\left(1+\frac{t}{n}\right)^{-n-2}\right| \leq c_{\mathcal{O}}\sqrt{\log(n)^2}\left(e^{-t_0}\left(1+\mathcal{O}\left(\frac{1}{n}\right)\right)\right).$$

Thus the integral can be bounded by a constant $K = K(n)$ for each n in this case.

2. $\Re(t) \geq \log(n)^2$: Along the contour that passes the real half-line $x \geq 1$ at a distance 1 the minimum $\frac{1}{n}$ of $|4y - 1 - \frac{t}{n}|$ is attained at $t_0 = n(4y-1)+i$. Again by (2.8) we can bound the modulus of the integral by

$$Cne^{-\log(n)^2}\int_{\mathcal{H}\cap\Re(t)\geq\log(n)^2}\left|\sqrt{t}\left(1+\frac{t}{n}\right)^{-2}\right|dt = o\left(\exp\left(-\log(n)^2\right)\right).$$

Putting it all together, this asymptotically gives

$$a(n,y) = \frac{(4y)^{n+2}}{n^{\frac{3}{2}}2\pi i}K. \qquad \blacksquare$$

Lemma 2.10. *Let z be in the domain G specified in Theorem 2.8 and $n \in \mathbb{N}$. The function*

$$F(n,z) = \frac{1}{\sqrt{1-4z}}\left(B(z) - \sum_{l=0}^{n} B_l z^l\right)$$

admits for z in the intersection of G and a sufficiently small neighborhood of $\frac{1}{4}$, as $z \to \frac{1}{4}$, the asymptotic expansion

$$\frac{2(4z)^{n+1}}{\sqrt{\pi}\sqrt{n}\sqrt{1-4z}}\left(1+\mathcal{O}\left(\frac{1}{n}\right)\right).$$

Proof. First we evaluate $F(n,z)$ by its generating function:

$$\begin{aligned}
\sum_{n\geq 0} F(n,z)x^n &= \frac{1}{\sqrt{1-4z}}\frac{B(z)-B(zx)}{1-x} \\
&= \frac{1}{\sqrt{1-4z}}\frac{1-\sqrt{1-4z}}{2z(1-x)}\frac{1-\sqrt{1-4zx}}{2zx(1-x)} \\
&= \frac{1}{2zx\sqrt{1-4z}}\left(-1+\frac{(1-x)\left(1+\sqrt{(1-4z)(1-4zx)}\right)}{(1-x)(\sqrt{1-4z}+\sqrt{1-4zx})}\right) \\
= \frac{1}{2zx\sqrt{1-4z}}&\left(-1+\frac{1}{\sqrt{1-4z}+\sqrt{1-4zx}}+\sqrt{1-4z}-\frac{1-4z}{\sqrt{1-4z}+\sqrt{1-4zx}}\right) \\
&= \frac{1}{2zx\sqrt{1-4z}}\left(-1+\sqrt{1-4z}-\frac{4z}{\sqrt{1-4z}+\sqrt{1-4zx}}\right).
\end{aligned}$$

Therefore

$$F(n,z) = [x^n]\frac{1}{2zx\sqrt{1-4z}}\frac{4z}{\sqrt{1-4z}+\sqrt{1-4zx}}$$
$$= \frac{2}{\sqrt{1-4z}}\,[x^n]\frac{1}{x(\sqrt{1-4z}+\sqrt{1-4zx})}.$$

Analogously to Lemma 2.9, we express this coefficient as a contour integral using Cauchy's coefficient formula and obtain the following Hankel integral:

$$[x^n]\frac{1}{x(\sqrt{1-4z}+\sqrt{1-4zx})} = \frac{(4z)^{n+1}}{\sqrt{n}2\pi i}\int_{\mathcal{H}}\frac{1}{\sqrt{-t}+\sqrt{n(1-4z)}}\left(1+\frac{t}{n}\right)^{-n-2}\,\mathrm{d}t$$

Let z be sufficiently close to $\frac{1}{4}$ so that we can disregard $\sqrt{n(1-4z)}$ in the denominator. In the case where $\Re(t) \le \log^2 n$, we use the asymptotic expansion (2.8).Specifically, we take:

$$\left(1+\frac{t}{n}\right)^{-n-2} = e^{-t}\left(1-\frac{2t}{n}+\mathcal{O}\left(\frac{t^2}{n}\right)\right),$$

where the error term $\mathcal{O}\left(\frac{t^2}{n}\right)$ is uniformly bounded for $|t| \le \log^2 n$.

Since $\frac{t}{n} \le \frac{\log^2 n}{n} \to 0$, this expansion converges uniformly on the contour $\mathcal{H}$. This allows us to interchange integration and summation:

$$\frac{(4z)^{n+1}}{\sqrt{n}2\pi i}\int_{\mathcal{H}}(-t)^{-\frac{1}{2}}\left(1+\frac{t}{n}\right)^{-n-2}\,\mathrm{d}t$$

$$= \frac{(4z)^{n+1}}{\sqrt{n}2\pi i}\left[\int_{\mathcal{H}}(-t)^{-\frac{1}{2}}e^{-t}\,\mathrm{d}t + \int_{\mathcal{H}}(-t)^{-\frac{1}{2}}e^{-t}\left(-\frac{2t}{n}\right)dt + \int_{\mathcal{H}}(-t)^{-\frac{1}{2}}e^{-t}\mathcal{O}\left(\frac{t^2}{n}\right)\,\mathrm{d}t\right]$$

$$= \frac{(4z)^{n+1}}{\sqrt{n}\Gamma\left(\frac{1}{2}\right)}\left(1-\frac{2\Gamma\left(\frac{1}{2}\right)}{n\Gamma\left(\frac{3}{2}\right)}+\mathcal{O}\left(\frac{1}{n}\right)\right)$$

$$= \frac{(4z)^{n+1}}{\sqrt{n}\Gamma\left(\frac{1}{2}\right)}\left(1+\mathcal{O}\left(\frac{1}{n}\right)\right),$$

where we used Theorem P13 to evaluate the integrals. Similarly, as in Lemma 2.9, in the case where $\Re(t) \ge \log(n)^2$, the integral is negligible. Thus, with $\Gamma(\frac{1}{2}) = \sqrt{\pi}$, we have

$$F(n,z) = \frac{2(4z)^{n+1}}{\sqrt{\pi}\sqrt{n}\sqrt{1-4z}}\left(1+\mathcal{O}\left(\frac{1}{n}\right)\right).$$

■

Proof of Theorem 2.8. Factoring out $\sqrt{1-4z}$ from (2.6) we get

$$D(z) = \frac{\sqrt{1-4z}}{2z}\sum_{l\ge 0}B_l\left(\sqrt{1+u_l}+\frac{u_l}{2}-\frac{u_l}{2}-1\right) \tag{2.9}$$

$$= \frac{1}{\sqrt{1-4z}}\sum_{l\ge 0}B_l z^l + \frac{\sqrt{1-4z}}{2z}\sum_{l\ge 0}B_l\left(\sqrt{1+u_l}-\frac{u_l}{2}-1\right) \tag{2.10}$$

where

$$u_l = \frac{4z^{l+1}}{1-4z} \quad \text{and} \quad \sum_{l\geq 0} B_l z^l = B(z).$$

Here, note that in the first summand $\sum B_l z^l = B(z)$, hence the first summand is a product of functions that are analytic in G and thus negligible (exponentially small). We employ the following inequalities, which are satisfied for every fixed z in the domain G by the corresponding u_l :

$$\left|\sqrt{1+u_l} - 1 - \frac{u_l}{2}\right| \leq \frac{|u_l|^2}{2} \tag{2.11}$$

$$\left|\sqrt{1+u_l} - 1\right| \leq \left|\sqrt{|u_l|}\right|. \tag{2.12}$$

A proof is sketched in a subsequent remark. Using (2.11) we get absolute and uniform convergence of the second sum of (2.10), for all z in a disk $U_\epsilon(z_0)$ around $z_0 \in G$ sufficiently small so that it lies in G (thus not containing the singularity $\frac{1}{4}$ of $D(z)$), by

$$\sum_{l\geq 0} B_l \left|\sqrt{1+u_l} - \frac{u_l}{2} - 1\right| \leq \frac{1}{2}\sum_{l\geq 0} B_l |u_l|^2 \leq \frac{c_\epsilon}{2}\sum_{l\geq 0} B_l r_\epsilon^l < \infty$$

since $|u_l|^2$ is uniformly bounded on this disk by its maximum at z_ϵ satisfying

$$|u_l(z_\epsilon)|^2 = \frac{16|z_\epsilon|^2}{|1-4z_\epsilon|^2}|z_\epsilon|^{2l} < c_\epsilon r_\epsilon^l, \quad r_\epsilon < \frac{1}{4},\ c_\epsilon \equiv const.$$

Now that we have established the analyticity of $D(z)$ on G, we proceed by decomposing $D(z)$ into partial sums and remainder terms, analyzing their asymptotic behavior individually.
Let $n \in \mathbb{N}$. Define

$$D_1(n,z) := \frac{\sqrt{1-4z}}{2z}\sum_{l=0}^{n} B_l\left(\sqrt{1+u_l} - 1\right),$$

$$D_2(n,z) := \frac{\sqrt{1-4z}}{2z}\sum_{l>n} B_l\left(\sqrt{1+u_l} - 1 - \frac{u_l}{2}\right),$$

$$D_3(n,z) := \frac{1}{\sqrt{1-4z}}\left(B(z) - \sum_{l=0}^{n} B_l z^l\right).$$

We have

$$D(z) = D_1(n,z) + D_2(n,z) + D_3(n,z).$$

1. *Establishing upper bounds for $D_1(n,z)$ and $D_2(n,z)$:*
   We start out by an upper bound for the modulus of $D^1(n,z)$. With the definition of u_l from (2.9) and its bound (2.12) we get

$$|D_1(n,z)| \leq \frac{\sqrt{|1-4z|}}{2|z|}\sum_{l=0}^{n} B_l\sqrt{|u_l|} = \sum_{l=0}^{n} B_l |z|^{\frac{l-1}{2}} \tag{2.13}$$

and use Lemma 2.9 with $y = \sqrt{|z|}$ sufficiently close to $\frac{1}{4}$. This results in

$$(2.13) = \mathcal{O}(4^n |z|^{n/2} n^{-3/2}).$$

We similarly upper-bound $D_2(n, z)$ with (2.11) by

$$|D_2(n,z)| \leq \frac{4|z|^2}{|1-4z|^{\frac{3}{2}}} \sum_{l>n} B_l |z|^{2l}. \tag{2.14}$$

Now we use the recurrence relation (1.1) for the Catalan numbers to bound B_l from above:

$$B_{n+k} = B_n \prod_{j=1}^{k} \frac{2(2(n+j)-1)}{n+j+1} \leq B_n \prod_{j=1}^{k} \left(4 - \frac{2}{n+j+1}\right) \leq 4^k B_n.$$

We continue in (2.14) by

$$\begin{aligned} &\leq \frac{4|z|^2}{|1-4z|^{\frac{3}{2}}} \sum_{l>n} B_n 4^{l-n} |z|^{2l} = \frac{1}{|1-4z|^{\frac{3}{2}}} B_n \sum_{l>n} 4^{l+1-n} |z|^{2(l+1)} \\ &= \frac{4^{-n}}{|1-4z|^{\frac{3}{2}}} B_n \sum_{l>n} 4^{l+1} |z|^{2(l+1)} \end{aligned}$$

For $z \in G \cap U_\epsilon(\frac{1}{4})$ and $\epsilon > 0$ sufficiently small the sum is uniformly bounded (in n and z) by a small constant. Thus inserting this into (2.14) gives $|D_2(n,z)| = \mathcal{O}\left(4^{-n}|1-4z|^{-\frac{3}{2}} B_n\right)$.

2. *Employing an estimate on $D_3(n, z)$:*
   From Lemma 2.10 we have

$$D_3(n,z) = \frac{2(4z)^{n+1}}{\sqrt{\pi}\sqrt{n}\sqrt{1-4z}} \left(1 + \mathcal{O}\left(\frac{1}{n}\right)\right)$$

   for z sufficiently close to $\frac{1}{4}$.

3. *Determining the threshhold n(z):*
   Finally, we need to show that there exists a threshold $n \equiv n(z)$ depending on $z \in G$ close to $\frac{1}{4}$, beyond which $D_3(n, z)$ becomes the main contributor to $D(z)$. That is, $D_1(n, z)$ and $D_2(n, z)$ are asymptotically negligible compared to $D_3(n, z)$. This holds for

$$n = \left\lfloor \frac{\log|1-4z|}{\log|z|} \right\rfloor$$

since by this choice we have $n \to \infty$ and $|z|^n = \mathcal{O}(1-4z)$. Together with (2.13), (2.14) and $B_n \sim \frac{4^n}{\sqrt{\pi n^3}}$ we obtain

$$D_1(n,z) = \mathcal{O}\left(\frac{4\log^{\frac{3}{2}}(|z|)|1-4z|^{\frac{1}{2}}}{\log^{\frac{3}{2}}|1-4z|}\right) = \mathcal{O}\left(\log^{-\frac{3}{2}}|1-4z|\right)$$

$$D_2(n,z) = \mathcal{O}\left(\frac{\log^{\frac{3}{2}}(|z|)}{\sqrt{\pi}\log^{\frac{3}{2}}(|1-4z|)|1-4z|^{\frac{1}{2}}}\right)$$

$$= \mathcal{O}\left(|1-4z|^{-\frac{1}{2}}\log^{-\frac{3}{2}}(|1-4z|)\right)$$

which are both in $o\left(\frac{(4z)^n}{\sqrt{1-4z}\sqrt{n}}\right)$ for this choice of n. Plugging in n we finally achieve the claimed asymptotic expansion:

$$D(z) \sim D_3(n,z) = \frac{2\sqrt{\log 4}}{\sqrt{\pi}\sqrt{1-4z}\sqrt{\log(1-4z)^{-1}}}\left(1+\mathcal{O}\left(\frac{1}{\log(1-4z)^{-1}}\right)\right).$$

■

Remark 2.11. The proof of (2.11) and (2.12) is obtained by the observation that the polynomial $1-4z+4z^{l+1}$ does not have a root contained in G, which is shown by using Rouché's theorem with the comparison function $1-4z$, which also has no zeros in G. One can then deduce that u_l does not take any value in $(-\infty,-1]$ since

$$\frac{1-4z+4z^{l+1}}{1-4z} = 1+\frac{4z^{l+1}}{1-4z} = 1+u_l$$

cannot vanish and also not become negative. Therefore, $|\sqrt{1+u_l}+1| \geq 1$ and thus we can individually bound the left-hand sides of (2.11) and (2.12) from above by an appropriate multiple of this.

In order to prove the expected size of the minimal DAG of a binary tree, we must determine the asymptotic behavior of the coefficients of (2.6). We have already seen in the proof of Theorem 2.8 that the asymptotic expansion of the generating function near a singularity (*singular expansion*) is closely linked to estimating the asymptotics of its coefficients.
Fortunately, in our case, both the generating function and the error term of the main contributor in the singular expansion are algebraic-logarithmic functions with isolated singularities. Hence, we can prove Theorem 2.7 using the analytic tools provided in [18], Chapter VI.

Proof of Theorem 2.7. We have shown in Theorem 2.8 that the generating function of $D(z)$ is analytic in the domain G, and as z tends to the dominant singularity $\frac{1}{4}$, given by

$$D(z) = \frac{2\sqrt{\log 4/\pi}}{\sqrt{(1-4z)\log\frac{1}{1-4z}}} + \mathcal{O}\left(\frac{1}{\sqrt{(1-4z)\log^3\frac{1}{1-4z}}}\right).$$

Define

$$K(z) := \frac{1}{\sqrt{1 - 4z \log \frac{1}{1-4z}}}$$

Then $K\left(\frac{z}{4}\right)$ is singular at 1. The nth coefficient of the function $K\left(\frac{z}{4}\right)$ can be derived with Lemma 2.12 below where $\alpha = \beta = -\frac{1}{2}$. Hence, it is given by

$$\frac{1}{\sqrt{n\pi}\sqrt{\log n}} \left(1 + \mathcal{O}\left(\frac{1}{\log n}\right)\right).$$

Similarily as to Theorem P14 one can show that in a sufficiently small neighbourhood to its singularity 1 intersected with the domain $\Delta = \{z \big| |z| < 1 + \epsilon, |\arg(z-1)| > \delta\}$ where $\epsilon > 0, \frac{\pi}{2} > \delta > 0$ the growth of the nth coefficient of a function in

$$\mathcal{O}\left((1-z)^{\alpha} \log^{\beta}\left(\frac{1}{1-z}\right)\right)$$

is of order

$$\mathcal{O}(n^{-\alpha-1} \log^{\beta} n),$$

see [18, p. 390]. This gives the growth order of the error terms to be in

$$\mathcal{O}\left(n^{-\frac{1}{2}} \log^{-\frac{3}{2}} n\right).$$

Therefore, as $z \to \frac{1}{4}$, we have derived the following asymptotics for the coefficients of $D(z)$:

$$\frac{D_n}{4^n} = \frac{2\sqrt{\log 4/\pi}}{\sqrt{n\pi}\sqrt{\log n}} \left(1 + \mathcal{O}\left(\frac{1}{\log n}\right)\right).$$

By applying the known asymptotics $B_n \sim \frac{4^n}{\sqrt{\pi n^3}}$, we find

$$\overline{D_n} = \frac{D_n}{B_n} = \frac{2n\sqrt{\log 4}}{\sqrt{\pi}\sqrt{\log n}} \left(1 + \mathcal{O}\left(\frac{1}{\log n}\right)\right).$$

The justification for deducing an asymptotic estimate of the coefficients of $D(z)$ from the asymptotic analysis of the singular expansion at the dominant singularity $\frac{1}{4}$ requires the so-called Transfer Theorem, found in [18, p. 393]. ■

Lemma 2.12. *Let $\alpha \in \mathbb{R} \setminus \mathbb{Z}_0$ and $\beta \in \mathbb{R}$. Then the nth coefficient of*

$$(1-z)^{\alpha} \log^{\beta}\left(\frac{1}{1-z}\right)$$

is given by

$$\frac{n^{-1-\alpha}}{\Gamma(-\alpha)} \log^{\beta}(n) \left(1 + \mathcal{O}\left(\frac{1}{\log n}\right)\right)$$

Proof. The instance of this Lemma is a special case of Theorem P14 where we replace

$$\left(\frac{1}{z}\log\frac{1}{1-z}\right)^{\beta} \quad \text{by} \quad \left(\log\frac{1}{1-z}\right)^{\beta}.$$

The Taylor expansion of z^{-1} at $z = 1$ yields

$$\frac{1}{z} = \sum_{n\geq 0}(-1)^n(z-1)^n = 1 + \sum_{n\geq 1}\sum_{k=0}^{n}\binom{n}{k}(-1)^k z^k = 1 + \sum_{n\geq 1}(1-z)^n = 1 + \frac{1-z}{z}.$$

Hence

$$\begin{aligned}(1-z)^{\alpha}\log^{\beta}\left(\frac{1}{1-z}\right)\left(\frac{1}{z}\right)^{\beta} &= (1-z)^{\alpha}\log^{\beta}\left(\frac{1}{1-z}\right)\left(1+\frac{z-1}{z}\right)^{\beta}\\ &= (1-z)^{\alpha}\log^{\beta}\left(\frac{1}{1-z}\right) + \mathcal{O}\left((1-z)^{\alpha+1}\log^{\beta}\left(\frac{1}{1-z}\right)\right)\end{aligned}$$

as $z \to 1$ in the domain $\Delta = \{z \big| |z| < 1+\epsilon, |\arg(z-1)| > \delta\}$ where $\epsilon > 0, \frac{\pi}{2} > \delta > 0$. Again, with [18, p. 390], the growth order of the coefficients of the error term is in

$$\mathcal{O}\left(n^{-\alpha-2}\log^{\beta} n\right)$$

Together with Theorem P14 we obtain the claim. ■

Remark 2.13. The statement of Theorem 2.7 is immediately extended to incomplete binary trees and to m-labeled (incomplete) binary trees due to the bijective nature mentioned in Section 1.1. In the last case, one has to adjust the asymptotics for the coefficients B_n to $B_{n,m} = m^n B_n$ and $\log(4m)$ instead of $\log(4)$; see [3]. The case of labeled binary trees is more complex and will be covered in the next section.

Corollary 2.14. *The compression rate of the minimal DAG compression for binary trees on n nodes is asymptotically given by*

$$2\sqrt{\frac{\log 4}{\pi}}\frac{1}{\sqrt{\log n}} \approx 1.3285\frac{1}{\sqrt{\log n}}$$

2.3 Compression rate for general trees

The next step is to analyze the compression performance of the minimal DAG for rooted general trees. We will explicitly derive the compression rate for a family of weighted, rooted trees that are a subclass of the family of simply generated trees, introduced by [30]. In conclusion, the expected size of the minimal DAG for Polya trees, increasing trees and labeled trees will be stated.

2.3.1 Simply generated trees

This class of rooted trees is defined by assigning different weights to trees according to their branching behavior. It consists of labeled and unlabeled, plane or non-plane trees that satisfy a certain functional equation. Instead of different node types like internal and external nodes, each node is assigned a weight depending on its out-degree and order of branches. The size of a tree is set as the number of nodes. For example, the unlabeled binary trees from the previous section can be represented in this model as the incomplete binary tree class consisting of the functional symbols $\epsilon,\ \epsilon \times \hat{\mathcal{B}},\ \hat{\mathcal{B}} \times \epsilon$ and $\hat{\mathcal{B}}^2$ of degrees $0, 1, 1$ and 2 each assigned a weight of 1 given by the functional equation $I = z(1 + I)^2$ and hence uniformly distributed. Since all simply generated trees are rooted, we exclude the root symbol in the functional symbols and add it later. This subsection relies on the work from [19], [13] and [32].

Definition 2.15. Let $\mathcal{F}$ be a set of functional symbols where each symbol $f \in \mathcal{F}$ has a predefined degree, $\deg(f) \geq 0$. Secondly, let $(w[f])_{f\in\mathcal{F}}$ be the non-negative weights associated to the functional symbols. Then the **structure power series** associated to $< \mathcal{F}, w >$ is given by

$$\phi(s) = \sum_{f\in\mathcal{F}} w[f]s^{\deg(f)} = \sum_{n\geq 0} \phi_n s^n.$$

Furthermore the generating function $T(z)$ of a weighted class of trees $\mathcal{T}$ is given by

$$T(z) = \sum_{t\in\mathcal{T}} w[t]z^{|t|} = \sum_{n\geq 0} W_n z^n \tag{2.15}$$

where the weights are canonically extended onto the trees as

$$w[t] = \prod_{n\in \text{ nodes}(t)} w[f[n]] \quad \text{and} \quad W_n = \sum_{|t|=n} w[t]. \tag{2.16}$$

Here $f[n]$ provides the functional symbol associated to the node n and $w[\emptyset] = 1$, for all $f \in \mathcal{F}$ with $\deg(f) = 0$. In this model the trees are taken proportional to their contribution to the structure measured by their weight. Hence, the random variable T_n supported on $\mathcal{T}_n \subset \mathcal{T}$ will take on a tree t with n nodes with probability

$$\mathbb{P}(T_n = t) = \frac{w[t]}{W_n}.$$

Remark 2.16. For a weighted tree class $\mathcal{T}$

$$\mathcal{T} = \mathcal{Z} \times \sum_{f\in\mathcal{F}} w(f) \cdot f\left(\mathcal{T}^{\deg(f)}\right)$$

This gives the functional equation of the generating function as

$$T(z) = z\phi(T(z)).$$

This equation is the identifying classification for a tree to belong to the family of "simply generated trees", as defined by Meir and Moon in [30]. Notice the dual influence of the functional symbols, which manifests both in the recursive tree description (outer) and directly on the trees itself (inner structure). Precisely, in case of a labeled class the structure polynomial is given by

$$\phi(s) = \sum_{f \in \mathcal{F}} w[f] \frac{s^{\deg(f)}}{(\deg(f))!}.$$

Example 2.17.

- *Catalan Trees: This class corresponds to unlabeled, plane rooted trees where each node may have any number of children (out-degree $k \geq 0$). The functional symbols are $\{(\cdot)^k\}_{k\geq 0}$, each with degree $\deg((\cdot)^k) = k$ and weight $w[(\cdot)^k] = 1$. The structure polynomial is:*

  $$\phi(s) = \sum_{k\geq 0} w[(\cdot)^k] s^k = \sum_{k\geq 0} s^k = \frac{1}{1-s}.$$

  The generating function satisfies $T(z) = z\phi(T(z)) = z/(1-T(z))$. The probability of a tree t with n nodes is $\mathbb{P}(T_n = t) = w[t]/B_{n-1}$, and $w[t] = 1$, where B_{n-1} is the $(n-1)$th Catalan number, the counting sequence for Catalan trees on n nodes.

- *Full d-ary Plane Trees: These are unlabeled, plane rooted trees where each node has out-degree k with $0 \leq k \leq d$, since we disregard external nodes. The functional symbols are $\{1, (\cdot)^1, \ldots, (\cdot)^d\}$, with $\deg((\cdot)^k) = k$ and $w[(\cdot)^k] = \binom{d}{k}$ to account for possibilities the k internal successors of a node can be positioned at. The structure polynomial is:*

  $$\phi(s) = \sum_{k=0}^{d} w[(\cdot)^k] s^k = \sum_{k=0}^{d} \binom{d}{k} s^k = (1+s)^d.$$

  Thus, $T(z) = z(1+T(z))^d$.

- *Labeled Catalan trees: Assigning a distinct label out of $\{1, \ldots, n\}$ to each node of a tree on n nodes results in $n!$ possible labelings. Since we use an exponential generating function for labeled structures we obtain the structure polynomial:*

  $$\phi(s) = \sum_{n\geq 0} \frac{1}{n!} s^n \implies T(z) = z\phi(T(z)) = \sum_{n\geq 1} \frac{(2n-2)!}{(n-1)!} \frac{z^n}{n!}$$

  where $W_n = \frac{(2n-2)!}{(n-1)!} = n! B_{n-1}$.

- *Cayley Trees: For the class of non-plane rooted labeled trees, called Cayley trees, we obtain the same weights as for labeled Catalan trees. The structure*

polynomial is:

$$\phi(s) = \sum_{k \geq 0} \frac{s^k}{k!} = e^s,$$

leading to $T(z) = ze^{T(z)}$*, the same functional equation as for labeled Catalan trees.*

- *Counter example: Polya trees and increasing trees Polya trees are unlabeled rooted non-plane trees that do not belong to the family of simply generated trees. It was shown in [14] that the associated weights do not fulfill the non-negativity condition. The class of increasing trees, in which the labels along each branch must appear in increasing order, leads to functional equations involving differential equations. As a result, these trees do not fall into the category of simply generated families as well.*

Lemma 2.18. *[19] The generating function* $A_u(z)$ *from (2.1) is given by*

$$A_u(z) = T(z, \phi(0)) - T(z, \phi(0) - w[u]z^{|u|-1}) \tag{2.17}$$

where $T(z, v) = zv + z\phi_{\geq 1}(T(z, v))$ *and* $\phi_{\geq 1}(s) = \phi(s) - \phi(0)$*.*

Proof. Observe that

$$z\phi(T(z)) = z(\phi(0) + \phi_{\geq 1}(T(z)) = T(z, \phi(0)) = T(z, v)\Big|_{v=\phi(0)}$$

and hence $T(z, v)$ is the generalization to the bivariate generating function $B(z, v)$ where we take into account the different (weighted) leaf types. The result (2.5) for the generating function $A_u(z)$, derived in Theorem 2.6 using the inclusion-exclusion principle, can therefore be directly transferred here:

$$A_u(z) = T(z, v)\Big|_{v=\phi(0)} - T(z, v)\Big|_{v=\phi(0)-w[u]z^{|u|-1}}.$$

■

Theorem 2.19. *[19] Let* $\mathcal{T}$ *be a tree class from the family of simply generated trees. Let* $t \in \mathcal{T}$ *of size* n *be drawn at random from the weighted model specified on* $\mathcal{T}$*. The asymptotic expectation on the size of the DAG corresponding to* t *is given by*

$$\overline{D_n} = \frac{D_n}{W_n}.$$

We assume that the maximal out-degree of a node is d *and there are* r *different node types* $n_1, \ldots, n_r$ *with not necessarily distinct out-degrees* $o_1, \ldots, o_r$*. Then the total weight* W_n *of all trees of size* n *from (2.15) is given by*

$$W_n = s_n \frac{1}{n} \sum_{\|\mathbf{m}\|=n} \binom{n}{m_0, \ldots, m_d} \phi_0^{m_0} \cdots \phi_d^{m_d}$$

where $s_n = 1$ *or* $s_n = n!$ *for an unlabeled or labeled class respectively,* $\mathbf{m} = (m_0, \ldots, m_d)$, $\mathbf{l} = (l_1, \ldots, l_r)$ *and*

$$D_n = s_n \sum_{\substack{k,m,l\geq 1 \\ m=n-(l-1)k \\ \|\mathbf{m}\|=m,\ \|\mathbf{l}\|=l}} \frac{(-1)^{k-1}}{ml} \frac{\binom{m_0}{k}}{\phi(0)^k} \binom{m}{m_0, \ldots, m_d} \binom{l}{l_1, \ldots, l_r} \phi_0^{m_0} \cdots \phi_d^{m_d} \left(w[n_1]^{l_1} \cdots w[n_r]^{l_r} \right)^k \tag{2.18}$$

such that $\sum_i i \cdot m_i = m - 1$ *and* $\sum_i o_i \cdot l_i = l - 1$.

Proof. The last restriction guarantees that the number of children is equal to the number of edges on n nodes. This ensures that a rooted tree structure is given by each selection. The weighted counting sequence can be directly derived using the Lagrange Inversion Theorem P11:

$$\begin{aligned} W_n &= s_n [z^n] T(z) = s_n \frac{1}{n} [s^{n-1}] (\phi(s))^n \\ &= s_n \frac{1}{n} [s^{n-1}] \sum_{\substack{n_0+\cdots+n_d=n \\ n_1+2n_2+\cdots+dn_d=n-1}} \binom{n}{n_0, \ldots, n_d} \phi_0^{n_0} \phi_1^{n_1} \cdots \phi_d^{n_d} y^{n_1+2n_2+\cdots+dn_d}. \end{aligned}$$

To determine

$$D_n = [z^n] \sum_{u \in \mathcal{T}} A_u(z)$$

we will use the result of Theorem 2.6 where we derived

$$\begin{aligned} D_n &= s_n [z^n] \sum_{l\geq 0} U_l \sum_{k\geq 1} (-1)^{k-1} \sum_{t \in \mathcal{T}} \binom{w_{u_l}[t]}{k} z^{|t|} \\ &= s_n \sum_{l\geq 0} \sum_{k\geq 1} (-1)^{k-1} U_l^{(k)} \sum_{\substack{t\in\mathcal{T},\ |t|=m \\ m=n-k(l-1)}} \binom{w_{u_l}[t]}{k} W_m. \end{aligned} \tag{2.19}$$

The unweighted number U_l of subtrees of size l and node profile $(l_1, \ldots, l_r)$, such that $l = \sum l_i$, is given by

$$\begin{aligned} U_{l_1,\ldots,l_r} &= \frac{s_l}{w[n_0]^{l_1} \cdots w[n_r]^{l_r}} [v_1^{l_1}, \ldots, v_r^{l_r}] T(z, \mathbf{v}) \\ &= \frac{s_l}{l \cdot w[n_0]^{l_1} \cdots w[n_r]^{l_r}} [v_1^{l_1}, \ldots, v_r^{l_r}] [s^{l-1}] \phi(s, \mathbf{v})^l \\ &= \frac{s_l}{l} \binom{l}{l_1, \ldots, l_r} \end{aligned}$$

where $\mathbf{v} = v_1, \ldots, v_r$ mark the different node types, so $\phi(s, \mathbf{v}) = \sum_{i=1}^r w[n_i] v_i s^{o_i}$. Hence, the weighted number of a k-fold occurrence is

$$U_l^{(k)} = s_l \frac{1}{l} \sum_{\substack{l_1+\ldots+l_r=l \\ o_1 l_1+\ldots+o_r l_r=l-1}} \binom{l}{l_1, \ldots, l_r} (w[n_1]^{l_1} \cdots w[n_r]^{l_r})^k.$$

The choice of $m = n - (l-1)k$ accounts for the removal of k occurrences of the subtree pattern u of size l. Since we are only considering u as a fringe subtree, each removal leaves behind a leaf node. Therefore, we can only insert a k-fold occurrence of u in

$$\binom{m_0}{k}$$

ways, where m_0 denotes the number leaf nodes. Furthermore, we have to adjust the weight by a factor of $1/\phi(0)^k$, as each inserted occurrence of the subtree turns a leaf of weight $\phi(0)$ into the subtree's root node of weight 1. Putting this all together, for fixed k and n we get

$$\sum_{t\in\mathcal{B}} \binom{w_u[t]}{k} z^{|t|} = \sum_{n\geq 0} \sum_{\substack{m=n-(l-1)k \\ m=m_0+\ldots+m_d}} \frac{1}{\phi(0)^k} \binom{m_0}{k} W_m.$$

Substituting into (2.19) gives the result. ■

Remark 2.20. In [19, Theorem 4] it is stated that

$$\overline{D_n} = C_{\mathcal{T}} \frac{n}{\sqrt{\log n}} \left(1 + \mathcal{O}\left(\frac{1}{\log n}\right)\right)$$

where the constant $C_{\mathcal{T}}$ depends on the specified model. As Flajolet et al. pointed out there, allowing for different weights on equal degree types of nodes leads to the presence of rather complex multinomials in the respective generating functions. Thus, the singularity analysis involves approximations via multivariate Gaussian distributions. We will avoid this and instead establish the average compression rate for a subclass of simply generated trees where we do not distinguish the branches ordering. Furthermore we will only consider unlabeled trees, and discuss the deviating result for the labeled version afterwards. In addition, the subclass of simply generated trees studied in [32] set the weights $\phi_i,\ i \in \mathbb{N}$, to be integers where every occurring fringe subtree $t \in \mathcal{T}$ has weight $w[t] = 1$. We refer to this subclass as $\hat{\mathcal{T}}$. This relaxation gives (2.16) as

$$w[t] = \prod_{k\geq 0} \phi_k^{m_k}.$$

Under certain restrictions the generating function $T(z)$ has a dominant singularity of square-root type and its coefficients therefore admit favorable asymptotics:

Theorem 2.21. *[13, p.76] Let $\phi(s) = \sum_{n\geq 0} \phi_n s^n$ is the structure polynomial with non-negative integer weights. Define R as the radius of convergence of $\phi(s)$, and suppose there exists $\tau \in [0, R)$ such that $\tau\phi'(\tau) = \phi(\tau)$. Let $\rho = \tau/\phi(\tau) = 1/\phi'(\tau)$ and let d be the greatest common divisor of all degrees of non-zero weighted functional symbols. Then, for $n \equiv 1 \pmod{d}$, the total weight of trees of size n is asymptotically:*

$$W_n = d\sqrt{\frac{\phi(\tau)}{2\pi\phi''(\tau)}} \cdot \frac{\rho^{-n}}{n^{3/2}} \left(1 + \mathcal{O}\left(\frac{1}{n}\right)\right),$$

and $W_n = 0$ otherwise (if $n \not\equiv 1 \pmod{d}$).

Theorem 2.22. *[32] Assume the conditions of the last theorem hold for the tree class $\hat{\mathcal{T}}$. Let $b = 1/\rho$ and $c = \sqrt{\phi(\tau)/(2\pi\phi''(\tau))}$, so that $W_n \sim cdn^{-3/2}b^n$ for $n \equiv 1 \pmod{d}$. For a tree t chosen randomly from $\hat{\mathcal{T}}_n$ of size $n \equiv 1 \pmod{d}$, the expected size of its minimal DAG is asymptotically:*

$$\overline{D_n} = \frac{2c}{\tau} \cdot \frac{n}{\sqrt{\log_b n}} + \mathcal{O}\left(\frac{n}{(\log n)^{3/2}}\right).$$

Proof. For simplicity, assume $d = 1$. We estimate D_n by considering the contribution of fringe subtrees based on their size l. The total contribution of distinct fringe subtrees of size at most $\log_b n$ is given by:

$$\sum_{l \le \log_b n} W_l \sim \sum_{l \le \log_b n} cl^{-3/2}b^l = \mathcal{O}\left(\frac{n}{(\log n)^{3/2}}\right).$$

This is negligible compared to the main term. Proceeding further, we cannot use the inclusion-exclusion technique from before as the number of leaves is non-trivial to compute. Instead, for larger fringe subtrees, where $\log_b n < l \le n$, we derive a naive estimate that overcounts by including every occurrence of each subtree. Since we want the asymptotic number of distinct subtrees we justify the correctness at the end. From [13, Theorem 3.13] we know that the asymptotic expected number of leaves L_n in a simply generated tree of size n that satisfies the requirements of this theorem is found to be $L_n = Ln$, where $L = \rho/\tau = 1/(b\tau)$. Thus, with $m = n - (l-1)$ the total number of occurrences is asymptotically

$$\sum_{\log_b n < l \le n} W_m L_m W_l \sim c^2 L \sum_{\log_b n < l \le n} m^{-3/2}b^m \cdot m \cdot l^{-3/2}b^l \tag{2.20}$$

$$= \mathcal{O}\left(\sum_{\log_b n < l \le n} l^{-3/2}b^{n+1}m^{-1/2}\right). \tag{2.21}$$

We will estimate the sum $\sum_{\log_b n < l \le n} W_m L_m W_l$ further by splitting the summation: Let

$$A = \sum_{l \ge n/2} l^{-3/2}b^{n+1}m^{-1/2} \quad \text{and} \quad B = c^2 L \sum_{l < n/2} m^{-3/2}b^m \cdot m \cdot l^{-3/2}b^l.$$

Then,

$$A = b^{n+1} \sum_{m=1}^{n/2} m^{-1/2}(n-m+1)^{-3/2} \sim b^{n+1}n^{-3/2}2\sqrt{n/2}.$$

Thus, since $b^{n+1} \sim bn^{3/2}W_n/c$, the total contribution is $\mathcal{O}(b^{n+1}n^{-1}) = \mathcal{O}(bW_n\sqrt{n})$, yielding an expected contribution of $\mathcal{O}(\sqrt{n})$.

On the other hand,

$$B = b^{n+1}n^{-1/2} \sum_{l=\lfloor \log_b n \rfloor + 1}^{n/2} l^{-3/2} \sim b^{n+1}n^{-1/2} \left(2(\log_b n)^{-1/2} + \mathcal{O}(n^{-1/2})\right).$$

Hence, our upper bound for D_n is asymptotically equal to

$$cLbW_n \cdot n \left(2(\log_b n)^{-1/2} + \mathcal{O}(n^{-1/2})\right).$$

This gives the expected number of distinct fringe subtrees as

$$\frac{2c}{\tau} \frac{n}{\sqrt{\log_b n}} \left(1 + \mathcal{O}\left(\frac{\sqrt{\log_b n}}{\sqrt{n}}\right)\right).$$

We now bound the correction terms in the Inclusion-Exclusion Theorem P10. Consider pairs of identical fringe subtrees of size l. Such a pair is obtained from a tree of size $n - 2l + 2$ with two leaves replaced by identical subtrees. The number of such pairs is at most

$$\sum_{\log_b n < l < n/2} (n - 2l + 2)^2 W_{n-2l+2} W_l.$$

Again, using $W_r \sim cr^{-3/2}b^r$, this is bounded by

$$\sum_{\log_b n < l < n/2} (n - 2l + 2)^{1/2} b^{n-2l+2} \cdot cl^{-3/2}b^l \sim cb^n n^{1/2} \sum_{l > \log_b n} l^{-3/2}b^{-l}.$$

Since $\sum_{l>\log_b n} l^{-3/2}b^{-l} \sim (\log_b n)^{-3/2}/(b-1)$, this is

$$\mathcal{O}\left(\frac{cb^n n^{1/2}}{(\log_b n)^{3/2}}\right) = \mathcal{O}\left(\frac{W_n n^2}{(\log n)^{3/2}}\right) = \mathcal{O}\left(\frac{n}{(\log n)^{3/2}}\right),$$

which confirms that the overcounting is within the error term. Thus, the expected number $\overline{D_n}$ of distinct fringe subtrees is

$$\frac{2c}{\tau} \cdot \frac{n}{\sqrt{\log_b n}} + \mathcal{O}\left(\frac{n}{(\log n)^{3/2}}\right)$$

as claimed. ■

2.3.2 Minimal DAG compression for Pólya and labeled trees

Section 6 of [32] generalizes the results for Pólya trees which are rooted non-plane trees and labeled trees. For *Pólya trees*, which are not simply generated, the generating function satisfies:

$$R(x) = x \exp\left(\sum_{m=1}^{\infty} \frac{1}{m} R(x^m)\right).$$

The number of trees of size n is given by

$$[x^n]R(x) = c \cdot \frac{b^n}{n^{3/2}} \left(1 + \mathcal{O}(n^{-1})\right), \quad c \approx 0.439924,\ b \approx 2.955765.$$

Theorem 2.22 holds, but a proof requires the use of bivariate generating functions due to possible isomorphisms in the subtree attachment process.
For *labeled trees*, with weights $w[f] = 1/[(\deg(f))!]$, the average number of distinct fringe subtrees is shown to be asymptotically equal to

$$\sqrt{\frac{2}{\pi}} \cdot \frac{n\sqrt{\log\log n}}{\sqrt{\log n}}.$$

2.3.3 Minimal DAG compression for binary increasing trees

In 2021, Bodini et al. [2] investigated the compression size for increasing trees which are labeled trees where the labels of nodes along each path from the root increase. The recursive characterization of rooted plane increasing trees is given via the boxed product by

$$\mathcal{T}^i = \mathcal{Z}^{\square} \star \mathrm{Set}(\mathcal{T}^i).$$

Specifically, for plane rooted binary increasing trees, which model binary search trees, this gives

$$\mathcal{B}^i = \mathcal{Z}^{\square} \star (1 + \mathcal{B}^i)^2 \implies B^i(z) = \int_0^z (1 + B^i(t))^2 dt$$

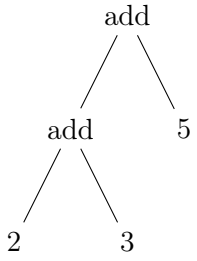
These trees do not belong to the class of simply generated trees because their generating functions are defined by a non-linear autonomous differential equation. It is also important to note that increasing trees are of logarithmic depth, whereas simply generated trees typically have square-root depth. Bodini et al. [2] showed that the average size of the minimal DAG for a random increasing and binary increasing tree with n nodes is $\mathcal{O}(n/\ln n)$ with a lower bound of $\Omega(\sqrt{n})$, and they conjecture it is actually $\Theta(n/\ln n)$. For plane binary increasing trees, they noted that prior work had already established a tighter result: the average size is $\Theta(n/\ln n)$.

3 Grammar-based tree compression

The minimal DAG compression for trees is based on the exploitation of repeated fringe subtree patterns, where each fringe subtree consists of a node and all its descendants. For trees with many repeated fringe subtrees, this method clearly achieves high compression rates. If we relax the rigid structure of fringe subtrees by allowing gaps within subtrees - such that small differences result in minor corrections in the encodings - we can enhance the compression potential. This generalization is realized by **Tree Straight-Line Programs (TSLPs)**, which extend the concept of Straight-Line Programs (SLPs) - originally developed for strings - to trees. Building on this foundation, we will explore the implementation and implications of an asymptotically optimal lossless compression code for a broad family of probabilistic binary tree sources.

Tree grammars

A context-free tree grammar, as defined in [10], is a formal system used to generate and describe trees. A tree grammar consists of an alphabet Σ (symbols called "terminals" with associated degrees), a set of variables (non-terminals), and production rules that define how trees are constructed. To illustrate this, consider the following tree:



This tree represents the grammar expression $\text{add}(\text{add}(2, 3), 5)$, where the alphabet consists of add, which is a binary symbol, and 2, 3, and 5 which are constants. In our example, the alphabet consists of all node labels. The non-terminals are placeholders during the generation process, which can be replaced by other trees according to

the production rules. For this example, let us use non-terminals S, E (for "start" and "expression"). The production rules specify how non-terminals can be rewritten into trees composed of terminals (and other non-terminals). For our example, the production rules are

$$S \to \text{add}(E, 5),$$
$$E \to \text{add}(2, 3).$$

Here, the first rule allows S to be replaced by a tree with add as the root and two subtrees (E and 5), while the other rule replaces E with the node add and the constants 2 and 3 as its children. To derive the tree shown above, we start with S and apply the rules as follows:

$$E \Rightarrow \text{add}(E, E) \Rightarrow \text{add}(\text{add}(E, E), E) \Rightarrow \text{add}(\text{add}(2, 3), 5).$$

Each step replaces a non-terminal E until only terminals remain.

3.1 TSLPs

This subsection is based on [27] and introduces linear context-free tree grammars for tree compression. If not otherwise specified the size of a tree is set as the number of its nodes.

Definition 3.1. Let $\mathcal{T}$ be an Σ-labeled rooted plane tree class where Σ is a finite alphabet. We call Σ a **ranked terminal alphabet**, if each element (terminal) $\sigma \in \Sigma$ sets the out-degree of a node labeled by this element, which we refer to as its **rank**, denoted by $\text{rank}(\sigma)$.

Definition 3.2. A **Tree Straight-Line Program** $\mathcal{G}$ over a ranked terminal alphabet Σ is a quadruple (N, Σ, S, P), where

- N is a finite set of ranked non-terminals, disjoint from Σ,
- $S \in N$ is the start non-terminal of rank 0,
- P is a finite set of production rules. Each rule is of the form $A(x_1, \ldots, x_n) \to t$, where $A \in N$, has rank n, $x_1, \ldots, x_n$ are dummy variables and t is a tree over $\Sigma \cup N \cup \{x_1, \ldots, x_n\}$. The TSLP is called **linear** if, for every rule, each dummy variable x_i (for $1 \leq i \leq n$) appears exactly once in the structure of t.
- each non-terminal $A \in N$ appears on the left-hand side of exactly one production, and the reachability relation $\{(A, B) \mid A(x_1, \ldots, x_n) \to t \in P, B \text{ occurs in } t\}$ is acyclic.

The tree $\mathrm{val}(\mathcal{G})$ is obtained by starting with S and iteratively replacing each non-terminal $A(x_1, \ldots, x_n)$ with the right-hand side t of its production rule, where we substitute each dummy variable x_i with the trees produced by other non-terminals or terminals, until the entire tree contains only terminal symbols from Σ. The size $|\mathcal{G}|$ is the total number of nodes in the right-hand sides of P, excluding non-terminal nodes.

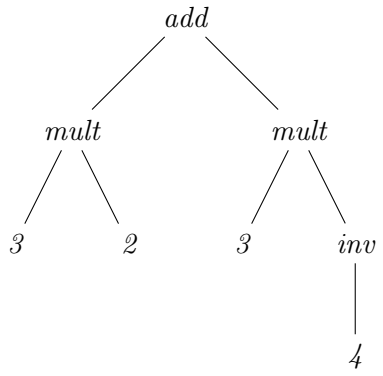
Remark 3.3. The last property guarantees that there are no cycles. For instance, if we have
$A_1 \to t_1$, t_1 contains A_2, $A_2 \to t_2$, t_2 contains A_3, …, $A_k \to t_k$, t_k contains A_1, then this results in an indefinite loop.

Example 3.4. *We extend the tree grammar from the previous example. Consider the TSLP* $\mathcal{G} = \langle N, \Sigma, S, P \rangle$*, where*

- $N = \{S, \mathrm{Mul}\}$ *(non-terminals, with* Mul *of rank 1),*
- $\Sigma = \{add, mult, inv, 2, 3, 4\}$ *(add and mult: rank 2, inv: rank 1,* $2, 3, 4$*: constants),*
- S *is the start non-terminal,*
- P *consists of the productions:*

$$S \to add(\mathrm{Mul}(2), \mathrm{Mul}(inv(4))$$
$$\mathrm{Mul}(x) \to mult(3, x)$$

This TSLP generates the tree $\mathrm{val}(\mathcal{G}) = add(mult(3, 2), mult(3, inv(4)))$ *of size 8.*



As mentioned in the introduction, unfortunately, constructing minimal TSLPs is computationally hard and there is no polynomial-time algorithm that produces the smallest tree grammar to a given tree. The best-known approximation achieves a bound analogous to the one established for string straight-line programs:

Theorem 3.5. *[25] Given a tree t of size n with maximal rank r, a linear TSLP $\mathcal{G}$ with $\mathrm{val}(\mathcal{G}) = t$ of size*

$$O\left(r \cdot g + r \cdot g \cdot \log\left(\frac{n}{r \cdot g}\right)\right),$$

where $g = \mathrm{opt}(t)$ is the size of the smallest TLSP generating t, can be computed $\mathcal{O}(n)$.

Sketch of a Proof: The algorithm iteratively applies three compression steps:

(1) *Chain compression*: Substitute maximal consecutive unary node patterns $a^l(x) = a(a(\cdots a(x)\cdots))$ with a new unary symbol a_l, and add the production rule $a_l(x) \to a^l(x)$, in total adding productions of size $O(k + \sum \log(l_i - l_{i-1}))$ per phase for k distinct chain lengths of patterns where a binary extension is used expressing the chain lengths to reduce representation costs (e.g., $a_8(x) \to a_4(a_4(x))$).

(2) *Pair compression*: Analogously, we merge all pairs $a(b(x))$ into a new unary symbol, and add the associated production rule. A smart partition of the occurring unary pairs maximizes replacements.

Remark: After the first two steps only isolated or no unary nodes at all are left. Since branching nodes (rank larger than 1) and leaf nodes are unchanged we remain with a relatively high number of leaves.

(3) *Leaf compression*: This step removes leaves by modifying nodes with leaf children. For a node $f(t_1, \ldots, t_m)$ of rank $m \leq r$ with leaf children at positions $i_1, \ldots, i_k$, it introduces f' of rank $m - k$, with:

$$f'(y_1, \ldots, y_{m-k}) \to f(t_1, \ldots, t_m),$$

costing $k + 1 \leq r + 1$ (non-parameter nodes). Per phase, $O(|T|)$ nodes are processed, costing $O(r|T|)$, but typically $O(|T|)$ leaves are removed (e.g., $|T|/2$ in a balanced tree).

Each step processes the tree within linear time complexity:

- *Chain Compression*: Identify and replace chains in $\mathcal{O}(n)$ time using a bottom-up traversal.
- *Pair Compression*: For each node, check if it's a 2-chain in constant time. Mark and replace non-overlapping pairs in linear time.
- *Leaf Compression*: Traverse the tree, encoding each node's children list (up to r children) as a sequence of length $\mathcal{O}(r)$. Sort these sequences with Radix sort in linear time.

Each step constructs new productions in a TSLP $\mathcal{G}$, and the process is repeated $\mathcal{O}(\log n)$ times. After each phase it can be shown that the size of the tree reduces by a constant number. Each phase takes $\mathcal{O}(n_i)$ time, where n_i is the current tree size. It can be shown that $n_i \leq \alpha^i n$, for a fixed $\alpha < 1$ (see [25]). Taking this into account, the total time is

$$\sum_{i=0}^{\mathcal{O}(\log(n))} \mathcal{O}(n_i) = \sum_{i=0}^{\infty} \mathcal{O}(\alpha^i n) = \mathcal{O}\left(n \sum_{i=0}^{\infty} \alpha^i\right) = \mathcal{O}\left(n \cdot \frac{1}{1-\alpha}\right) = \mathcal{O}(n).$$

When it comes to the representation costs, the idea is to introduce a credit scheme. In this scheme, each occurrence of a letter in the minimal grammar is assigned a fixed amount of credit, and when a compression operation is applied, the released credit gives the representation cost with the new symbols. For chain compressions, one shows that every chain pattern of the form a^l can be represented by a cost of $\mathcal{O}(1 + \log l)$, totaling in $\mathcal{O}(\log n)$ costs by wisely accounting for powers. Since there are at most $g + (n_0 + n_1)r$ many such rules (where g is the size of the smallest grammar and n_0, n_1 count occurrences of constant and unary symbols), the overall cost for chain compression is $\mathcal{O}((g + (n_0 + n_1)r) \log n)$. A similar (and in fact lower) cost is incurred for the other compression steps (unary pair and leaf compression). By known bounds relating $\mathcal{G}$ and the minimal grammar for the tree, this yields $\mathcal{O}(gr + g \cdot r \cdot \log n)$. So far we did not take into account that the algorithm produces a so-called *reasonable grammar*, for example trivial productions $A \to a$, $|a| = 1$ do not occur. Taking this into account one shows the claimed bound. ■

TSLPs compression outperform DAG compression in the case of random binary trees and all other random trees with node-degree smaller than two:

Theorem 3.6. *[27] For a binary tree t of size n with σ labels, a linear TSLP of size $\mathcal{O}\left(\frac{n}{\log_\sigma n}\right)$ can be computed in linear time.*

Sketch of a Proof: Decompose t into $\mathcal{O}\left(\frac{n}{\log_\sigma n}\right)$ clusters of size $\mathcal{O}(\log_\sigma n)$, each replaced by a non-terminal. Compute the minimal DAG of these clusters (size bounded by $\sqrt{n}$, using Catalan number estimates for binary trees), then combine with a start production. The total size is $\mathcal{O}\left(\frac{n}{\log_\sigma n}\right)$, achieved in $\mathcal{O}(n)$ time via efficient DAG construction. ■

This competes with the $\Omega\left(\frac{n}{\log_\sigma n}\right)$ optimality bound on the minimal TSLP size for uniformly random trees. To see that this bound is optimal take into account that in order to uniquely identify (in lossless compression) a σ-labeled random tree $\log(\sigma^n) = n \log(\sigma)$ bits of information are necessary which leads to about $n/\log_\sigma n$ productions. Essentially, the approach conducted by TSLPs reminds us of the combinatorial strategy "divide and conquer", it divides into different compression

stages, conquers them with compact productions, and combines them through substitution.

3.2 A universal grammar-based code for unlabeled binary trees

In this subsection, we explore the universal compressor for unlabeled plane binary trees from [34], which was the first universal tree source code. In comparison to the TSLPs we discussed previously, the compression technique employed aims for more than efficiency, namely universality. This means we allow for a broad variety of probability distributions on the binary tree sources. Unlike Huffman codes (which we will work with later on) where a penalty for assuming a deviant probability distribution is paid, in this way we can compress sources with pretty much unknown probability distributions or, say, generation process. The goal is lossless compression, meaning the original tree can be perfectly reconstructed from its encoded form. The process consists of two main steps for encoding (ϕ_e) and two corresponding steps for decoding (ϕ_d).

In the following we fix the tree source to be the set of (unlabeled) binary trees $\mathcal{B}$ with a minimum of two leaves where the size is defined as the number of leaves. The probability distribution $\mathbf{P}$ will be specified later on. Furthermore, let $A = \{0, 1\}$ be the set of all finite-length bit strings and $l(x)$ specify the length of each $x \in A^*$.

Definition 3.7. Given a TSLP $\mathcal{G}$, we say $\mathcal{G}$ forms a **representation** of $t \in \mathcal{B}$ if removing the vertex labels of $\mathrm{val}(\mathcal{G})$ results in t.

Encoding Step 1: Constructing the tree grammar $\mathbf{G}_t$
Given a binary tree $t \in \mathcal{B}$, the first step is to transform t into a context-free grammar $\mathcal{G}_t = (N, \Sigma, S, P)$. Since we want the size of $\mathrm{val}(\mathcal{G})$ to return the number of leaves, internal node labels will carry non-terminals. We define $N = \{0, 1, \ldots, F-2\}$ as the set of non-terminal variables, each of rank 2, where $F = F(t)$ is the number of distinct fringe subtrees of t. Note that $F \geq 2$ since we require the tree to have at least two leaves. The alphabet Σ consists of a single variable L of rank 0 and the start non-terminal is $S = 0$.

Before we set up the production rules, we establish a labeling on the internal nodes consisting of non-terminals: The nodes of t are labeled in breadth-first order. The root receives label 0, and each leaf is labeled L. Starting from the root, we label its two children consecutively, then continue with the children of those nodes in the next depth level and so on. Doing so, for each internal node v encountered without a label, we check whether its fringe subtree t_v matches the fringe subtree of a previously labeled vertex v'. If so, v inherits the label ofv'. Otherwise, it receives

the smallest unused integer from $\{1, \ldots, F-2\}$. Each production rule

$$i \to (i_1, i_2)$$

is then extracted from the labeled tree, where i is a non-terminal, that becomes the root node label, and i_1, i_2 are the labels of its left and right children. Trivially, this labeled tree is a representation of t. This slightly modifies the production rules of the TSLP from Definition 3.2.

Encoding Step 2: Generating the codeword for each tree

For each $\mathcal{G}_t$ we create a binary codeword $C(\mathcal{G}_t) \in A^*$ that sets $\phi_e(t)$. Define a sequence $S(t) = (a_1, a_2, \ldots, a_{2F-2})$, where for each non-terminal $i = 0, \ldots, F-2$, the pair (a_{2i+1}, a_{2i+2}) is the right-hand side of the production $i \to (a_{2i+1}, a_{2i+2})$. This sequence, together with the alphabet $A(t) = \{1, 2, \ldots, F-2\} \cup \{L\}$, uniquely identifies $\mathcal{G}_t$. A second sequence, $S_1(t)$, results from $S(t)$ by removing the first appearance of each non-terminal. The codeword $C(\mathcal{G}_t)$ is then given by mapping $S(t)$ to A^*.

$F = 2$: The unique binary tree with two leaves gets mapped to "1".

$F > 2$: The codeword $C(\mathcal{G}_t)$ is the concatenation of four binary strings: B_1, B_2, B_3 and B_4.

- B_1 is a binary string of length $F-1$ consisting of $F-2$ zeroes followed by a one.
- B_2 is a binary string of length $2F-2$ with exactly $F-2$ ones. The positions of the ones correspond to the first appearances of the members of the set $\{1, 2, \ldots, F-2\}$ in $S(t)$.
- For each $a \in A(t)$, let k_a be the number of times a appears in $S(t)$. B_3 is a binary string of $F-1$ alternate runs of ones and zeros, where the lengths of the runs are $k_1, k_2, \ldots, k_{F-2}, 1$.
- Let $\mathcal{S}_1(t)$ be the set of all permutations of $S_1(t)$, and let

  $$M(t) = \lceil \log_2 (|\mathcal{S}_1(t)|) \rceil.$$

  If $M(t) = 0$, B_4 is the empty string. Otherwise, list all members of $\mathcal{S}_1(t)$ in lexicographical order induced by the ordering of the alphabet $A(t)$ and assign each permutation an integer, according to the order, starting from 0. B_4 then becomes the $M(t)$-bit binary expansion of the integer attributed to $S_1(t)$.

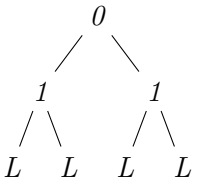
Hence, the binary strings carry the information of distinct fringe subtrees, initial internal node label occurrences, frequency of symbols and the relative ordering of

symbols within their category. Note that

$$|\mathcal{S}_1(t)| = \frac{F!}{k_L!(k_1-1)!\cdots(k_{F-2})!}$$

as S_1 is of length F.

Example 3.8. *Consider the following binary tree:*



The production rules associated to the tree are

$$0 \to (1,1),$$
$$1 \to (L,L).$$

The encoding process is:

- $F = 3$
- $S(t) = (1,1,L,L),\ S_1(t) = (1,L,L),\ S_2(t) = (1)$
- $f_1 = 2,\ f_L = 2$
- $M(t) = \lceil \log_2(3) \rceil = 2$, $S_1(t)$ *has index 0 in* $\mathcal{S}_1 = \{(1,L,L),(L,1,L),(L,L,1)\}$
- $B_1 = 01,\ B_2 = 1000,\ B_3 = 110,\ B_4 = 00$
- *Codeword:* 01100011000

Lemma 3.9. *The encoding map $\phi_e : \mathcal{B} \to A^*$, which assigns to each binary tree $t \in \mathcal{B}$ the codeword $C(\mathcal{G}_t)$, is prefix-free. Furthermore, the decoding map ϕ_d reconstructs the original binary tree $t \in \mathcal{T}$.*

Proof. Let $t \in \mathcal{B}$ be a binary tree with F distinct fringe subtrees, and let $C(\mathcal{G}_t) = B_1B_2B_3B_4$ be its codeword as defined in Encoding Step 2. We prove that the code is prefix-free by showing that $\mathcal{G}_t$ can be uniquely reconstructed from any binary string $a \in A^*$ of which $C(\mathcal{G}_t)$ it is a prefix of.
First, consider the case $F = 2$. Since any other tree $t' \in \mathcal{B}$ has $F(t') > 2$ (implying a codeword length greater than 1, starting with at least one 0 from B_1), $C(\mathcal{G}_t) = 1$ cannot be a prefix of $C(\mathcal{G}_{t'})$.
Now assume $F > 2$. Due to the construction, B_1 and B_2 are immediately identified, from which $F(t)$, the alphabet $A(t)$ and first appearances in S(t) are recovered.

Next, B_3 and thus the frequencies of symbols ($k_L = (2F-2) - \sum k_i$) are discovered since the number of alternating runs, $F-1$, is known, and B_3 ends with a single bit. Convert the remaining B_4 to an integer I, and find $S_1(t)$ as the I-th permutation of $\{1^{k_1-1}, 2^{k_2-1}, \ldots, (F-2)^{k_{F-2}-1}, L^{k_L}\}$ in lexicographical order. Reconstruct $S(t)$ by merging the information from $S_1(t)$ and $A(t)$ according to B_2's one-positions. From this we can construct the representation $\mathcal{G}_t$ of t. This implicitly gives $\phi_d(a)$. ■

Remark 3.10. The decoding map ϕ_d reconstructs t in two steps: first, it extracts $F(t)$, the alphabet $A(t)$, and the first appearances from B_1 and B_2, then it identifies the frequencies and permutation $S_1(t)$ from B_3 and B_4. Second, it merges $S_1(t)$ with $A(t)$ using the one-positions from B_2 to recreate $S(t)$, which enables the reconstruction of $\mathcal{G}_t$.

Lemma 3.11. *Let $t \in \mathcal{B}$ and $\phi_e(t)$ be the binary encoding of t. Furthermore, let*

$$p_a = \frac{k_a}{F}, \quad a \in A$$

be the relative frequency of each symbol in $S_1(t)$ (defined previously) and $p = (p_1, \ldots, p_L)$ its probability distribution. Then,

$$l(\phi_e(t)) \leq 5(F-1) + FH(p).$$

Proof. For $F = 2$, t is the unique tree in $\mathcal{B}$ with two leaves, and $l(\phi_e(t)) = 1$. The right-hand side evaluates to $5(2-1)+2\cdot 0 = 5$, since $S_1(t) = (L, L)$ has zero entropy, so the inequality holds. Now assume $F > 2$. The codeword length is

$$l(\phi_e(t)) = l(B_1)+l(B_2)+l(B_3)+l(B_4) = (F-1)+(2F-2)+l(B_3)+\lceil \log_2(|\mathcal{S}_1(t)|)\rceil.$$

Since B_3 has $F-1$ runs with total length smaller than $2F-2$ (as $k_L > 1$), we obtain the following bound:

$$l(\phi_e(t)) \leq 5(F-1) + \log_2(|\mathcal{S}_1(t)|).$$

In order to bound the cardinality of the set $\mathcal{S}_1$ by the entropy, we assign to the alphabet A the probability distribution p induced by the Encoding Step 1 $\phi_e(t)$. Obtaining an element of $\mathcal{S}_1$ is then given by the probability

$$\begin{aligned} p^F := \prod_{a\in A} p_a^{k_a} &= \exp\left(\sum_{a\in A} k_a \log_2(p_a)\right) \\ &= \exp\left(F\sum_{a\in A} p_a \log_2(p_a)\right) = \exp(-FH(p)). \end{aligned}$$

This gives the total probability of obtaining a sequence $S_1 \in \mathcal{S}_1$ as

$$p^F(\mathcal{S}_1) = |\mathcal{S}_1|\exp(-FH(p)) \implies |\mathcal{S}_1| \leq \exp(FH(p)).$$

Thus, $\log_2(|\mathcal{S}_1(t)|) \leq FH(p)$ completes the proof. ■

Definition 3.12. Define D as the minimal DAG representation of t consisting of F vertices, where the root vertex represents the tree t and there is a unique leaf vertex. Then, the grammar $\mathcal{G}(D)$ consists of the non-terminals $\{0, 1, \ldots, F-2\}$, where the root is assigned label 0, and internal vertices (those with out-degree 2) are labeled in breadth-first order. The unique leaf vertex is labeled with the terminal symbol L, and the corresponding productions are derived accordingly.

Lemma 3.13. *For $t \in \mathcal{B}$, $\mathcal{G}_t$ has the smallest number of variables, among all TSLPs forming a representation of t, equal to $F(t)$.*

Proof. This is immediate from the fact that $\mathcal{G}_t$ corresponds to the minimal DAG. Since the minimal DAG is unique up to isomorphism, any grammar with $F(t)$ variables forming a representation of t is isomorphic to $\mathcal{G}_t$. ■

Optimality and universality for leaf-centric binary tree sources

We now investigate the conditions under which (ϕ_e, ϕ_d) is universal and asymptotically optimal (see Definitions P26 and P27) for *leaf-centric* binary tree sources. In [34] they also show under which conditions (ϕ_e, ϕ_d) is universal for *depth-centric* binary sources.

Definition 3.14. Let $I = \{(i,j) \mid i,j \geq 1, n \geq 2, i+j=n\}$ denote the set of pairs of positive integers summing to n (where n will be the number of leaves in a binary tree). Define the set of functions

$$\Sigma = \left\{ \sigma : \mathbb{N} \times \mathbb{N} \to [0,1] \;\middle|\; \sum_{(i,j)\in I} \sigma(i,j) = 1 \right\},$$

where each σ assigns a probability to pairs (i,j). For each $\sigma \in \Sigma$, we define a **leaf-centric binary source** $(\mathcal{B}, \mathbf{P}_\sigma)$, where $\mathcal{B}$ is the set of all binary trees with a minimum of two leaves, and the probability of a tree $t \in \mathcal{B}$ is

$$\mathbf{P}_\sigma(t) = \prod_v \sigma\left(|t_{v_L}|, |t_{v_R}|\right),$$

with the product taken over all internal nodes v of t. Here, v_L and v_R are the left and right children of v, and $|t_{v_L}|$ and $|t_{v_R}|$ denote the number of leaves in the fringe subtrees rooted at v_L and v_R, respectively. The function $\mathbf{P}_\sigma$ of the tree source is well-defined, as by induction we have

$$\sum_{t\in\mathcal{B}_n} \mathbf{P}_\sigma(t) = \sum_I \sigma(i,j) \sum_{r_L\in\mathcal{B}_i} \mathbf{P}_\sigma(r_L) \sum_{r_R\in\mathcal{B}_j} \mathbf{P}_\sigma(r_R) = 1,$$

where r is the root and $\mathcal{B}_n = \{t \in \mathcal{B} \mid |t| = n\}$ is the set of binary trees with n leaves.

Remark 3.15. Consider a subset of leaf-centric binary tree sources, called the *one-dimensional leaf-centric binary tree sources* $(\mathcal{B}, \mathbf{P}_{\hat{\sigma}})$ where $\hat{\sigma} \in \Sigma$ satisfies

$$\hat{\sigma}(i,j) \neq 0 \quad \text{if and only if} \quad (i,j) \in I \cap \{(1,n-1),(n-1,1)\}.$$

This tree source outputs trees in which each internal node must have at least one child that is a leaf. Furthermore, we can construct an analogous model for binary sequences, a *one-dimensional binary sequence source*, by distinguishing an infinite-length bit sequence a_∞ and assigning a positive probability only to sequences $a \in A^*$ where each bit in a is the opposite of the corresponding bit in a_∞. Each tree $t \in \mathcal{B}_n$ with $\mathbf{P}_{\hat{\sigma}}(t) > 0$ can be assigned to a binary string $b \in A^*_{n-1}$ by encoding the sequence of left or right leaf choices at each internal node.
In [26] Kieffer showed that there cannot exist a universal code for such binary sequence sources. Since the one-dimensional leaf-centric tree sources are isomorphic to this family, they inherit this property: no universal code exists for $(\mathcal{B}, \mathbf{P}_{\hat{\sigma}})$. In consequence, also no universal code for the leaf centric binary tree sources exists. Hence, we need to further restrict the family:

Theorem 3.16. *The code (ϕ_e, ϕ_d) is a universal code for the family of leaf-centric binary trees $(\mathcal{B}, \mathbf{P}_\sigma)$ where $\sigma \in \Sigma$ is subject to the constraint*

$$\sup\left\{ \frac{i+j}{\min(i,j)} : i,j \geq 1,\ \sigma(i,j) > 0 \right\} < \infty. \tag{3.1}$$

Remark 3.17. We call this subclass of leaf-centric binary tree sources *imbalanced-bounded*, since the condition (3.1) bounds the ratio of total leaves $(i+j)$ to the smaller subtree's leaves $(\min(i,j))$ at each internal node.

First, we need to establish a condition of asymptotic optimality over a family of tree sources:

Lemma 3.18. *Any lossless code (ψ_e, ψ_d) on $\mathcal{B}$ is asymptotically optimal for the family $(\mathcal{B}, \mathbf{P})_{\mathbf{P} \in \mathcal{P}}$, if*

$$\limsup_{n \in \mathbb{N}} R(\psi_e, n, \mathbf{P}) \leq 0, \tag{3.2}$$

where $R(\psi_e, n, \mathbf{P})$ is defined in P26.

Sketch of a Proof: Fix any probability inducing function $\mathbf{P}$. By Kraft's inequality, there exists a distribution Q such that $l(\psi_e(t)) \geq -\log_2 Q(t)$. With known methods on estimating the divergence and "error costs" for assuming a deviant probability distribution (employed in [34]) one can show that,

$$\liminf_{n \in \mathbb{N}} R(\psi_e, n, \mathbf{P}) \geq \liminf_{n \in \mathbb{N}} \sum_{t \in \mathcal{T}_n} |t|^{-1} \mathbf{P}(t) \log_2 \left(\frac{\mathbf{P}(t)}{Q(t)} \right) = 0. \tag{3.3}$$

Hence, by Definition P26, the instances (3.2) and (3.3) imply asymptotic optimality. ■

Definition 3.19. Let Λ be the set of mappings $\lambda : \mathcal{B} \cup \{\square\} \to (0, 1]$ satisfying

(a) $\lambda(t) \leq \lambda(t_L)\lambda(t_R)$, $t \in \mathcal{B}$,

(b) $1 \leq \sum_{t \in \mathcal{B}_n} \lambda(t) \leq n^{K(\lambda)}$ for some $K(\lambda) > 0$.

A source $(\mathcal{B}, \mathbf{P})$ has the **Domination Property** if some $\lambda \in \Lambda$ satisfies $\mathbf{P}(t) \leq \lambda(t)$ for all $t \in \mathcal{B}$.
Furthermore we say that the source has the **Representation Ratio Negligibility (RRN) Condition** if

$$\lim_{n \in \mathbb{N}} \sum_{t \in \mathcal{B}_n} r(t)\mathbf{P}(t) = 0,$$

where $r(t) = F/|t| \leq 1$ measures the information content (symbols in tree grammar) per leaf.

Theorem 3.20. *The code (ϕ_e, ϕ_d) is asymptotically optimal for any source $(\mathcal{B}, \mathbf{P})$ if it satisfies the Domination Property and the RRN Condition.*

Proof. First of all for each $\lambda \in \Lambda$ and $t \in \mathcal{B}$ the inequality

$$\frac{l(\phi_e(t) + \log_2(\lambda(t))}{|t|} \leq C_\lambda \gamma(r(t)),$$

holds, where

$$\gamma(x) = \begin{cases} -\frac{x}{2} \log_2 \left(\frac{x}{2}\right), & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$$

Due to space limitations and the nature of the applied methods, we omit the proof of this, but it can be found in [34]. Since the source satisfies the Domination Property, there exists a $\lambda \in \Lambda$ such that

$$\begin{aligned} R(\phi_e, n, \mathbf{P}) &\leq \sum_{t \in \mathcal{B}_n, \mathbf{P(t)} > \mathbf{0}} \frac{l(\phi_e(t) + \log_2(\lambda(t))}{|t|} \mathbf{P}(t) \\ &\leq C_\lambda \sum_{t \in \mathcal{B}_n, \mathbf{P(t)} > \mathbf{0}} \gamma(r(t))\mathbf{P(t)}. \end{aligned}$$

Since $\gamma(x)$ is concave on $[0, 1]$ we can apply Jensen's inequality and further obtain

$$C_\lambda \sum_{t \in \mathcal{B}_n, \mathbf{P(t)} > \mathbf{0}} \gamma(r(t))\mathbf{P(t)} \leq C_\lambda \gamma \left(\sum_{t \in \mathcal{B}_n, \mathbf{P(t)} > \mathbf{0}} r(t)P(t) \right)$$

The proof is finished using the RRN Property and Lemma 3.18:

$$\lim_{n \in \mathbb{N}} R(\phi_e, n, \mathbf{P}) \leq \lim_{n \in \mathbb{N}} C_\lambda \gamma \left(\sum_{t \in \mathcal{B}_n, \mathbf{P(t)} > \mathbf{0}} r(t)\mathbf{P}(t) \right) = 0. \qquad \blacksquare$$

Proof of Theorem 3.16. To prove this, we show that each source $(\mathcal{B}, \mathbf{P}_\sigma)$ satisfies the Domination Property and the RRN Condition.

Step 1: Domination Property
Let $C_n = |\mathcal{B}_n|$ be the $(n-1)$th Catalan number. Define $\lambda(\square) = 1$ and

$$\lambda(t) = \max(C_{|t|}^{-1},\ \mathbf{P}_\sigma(t))$$

for $t \in \mathcal{B}$.

(a) For $t \in \mathcal{B}$, let $n = |t|$, $i = |t_L|$, $j = |t_R|$, so $i + j = n$. Since $C_n^{-1} \leq C_i^{-1} C_j^{-1}$ by properties of the binomial coefficient, and $\mathbf{P}_\sigma(t) \leq \mathbf{P}_\sigma(t_L)\mathbf{P}_\sigma(t_R)$, we have $\lambda(t) \leq \lambda(t_L)\lambda(t_R)$.

(b) $\sum_{t \in \mathcal{B}_n} \lambda(t) \leq \sum_{n \in \mathbb{N}} C_n^{-1} + \sum_{t \in \mathcal{B}_n} \mathbf{P}_\sigma(t) = 1+1 = 2 \leq n^{K(\lambda)}$ for some $K(\lambda) \geq 1$ (e.g., $K(\lambda) = 1$ for $n \geq 2$).

Since $\lambda \in \Lambda$ and $\mathbf{P}_\sigma(t) \leq \lambda(t)$, the Domination Property holds.

Step 2: RRN Condition
Let $t \in \mathcal{B}_n$ for an arbitrary $n \geq 2$ with $\mathbf{P}_\sigma(t) > 0$. Consider the truncated tree t' of t with F leaves and F distinct fringe subtrees. Such a t' exists and can be constructed by reducing the grammar that forms a representation of t to use each production exactly once, leading to $F - 1$ internal nodes. Let $L_1, \ldots, L_F$ be the leaves of t', and v_i the parent of L_i, $i = 1, \ldots, F$. Let S be the supremum from (3.1). Since $\mathbf{P}_\sigma(t) > 0$ we get

$$\sum_{i=1}^{F} \frac{|t_{v_i}|}{|L_i|} \leq S \cdot F \leq Sn.$$

Hence,

$$\sum_i |t_{v_i}| \leq Sn.$$

Each v_i has at most two leaf children, so the number of distinct v_i's is at least $F/2$. Thus, there are at least $\lceil F/2 \rceil$ distinct subtrees $f_1, f_2, \ldots, f_k$ (where $k = \lceil F/2 \rceil$) with $\sum_{i=1}^{k} |f_i| \leq Sn$.

Order all trees in $\mathcal{B}$ by size: $t(1) \in \mathcal{B}_2$, $t(2), t(3) \in \mathcal{B}_3$, etc., with $m_i = |t(i)|$. Define

$$k(M) = \max\{k \geq 1 : m_1 + \cdots + m_k \leq M\}, \quad M \geq 2.$$

Since $C_n \sim 4^n/(n^{3/2}\sqrt{\pi})$ grows exponentially,

$$k(M)/M = O(1/\log M),$$

so $\lim_{M \to \infty} k(M)/M = 0$. Here, $k(Sn) \geq \lceil F/2 \rceil$, so $F/n \leq 2k(Sn)/n$.

Now,

$$\sum_{t\in\mathcal{B}_n} r(t)\mathbf{P}_\sigma(t) \leq \sum_{t\in\mathcal{B}_n} \frac{2k(Sn)}{n}\mathbf{P}_\sigma(t) = \frac{2k(Sn)}{n}\sum_{t\in\mathcal{B}_n}\mathbf{P}_\sigma(t) = \frac{2k(Sn)}{n}.$$

As $n \to \infty$,

$$\frac{k(Sn)}{Sn} \to 0,$$

so $k(Sn)/n \to 0$. Thus, the RRN Property holds. ■

In 2022 Ganardi et al. [21] extended this universal grammar-based tree compression for binary trees established by Kieffer et al. [34] to other binary tree sources and improved performance bounds by also bounding the worst-case redundancy.

4 Entropy-based compression

Another approach to finding the optimal lossless tree compression is to align it with the information-theoretic limit, the so-called entropy $H(T)$ of the tree source T. In this section we will establish the entropy $H(T)$ for certain tree sources and develop compression algorithms that achieve compression bounds in terms of that entropy. Compressed representations were first classified by Jacobson in [24] according to the amount of space required to encode a tree chosen uniformly at random from the source T:

- **Implicit:** if it requires $H(T) + O(1)$ bits of space.
- **Succinct:** if it requires $H(T) + o(H(T))$ bits of space.
- **Compact:** if it requires $O(H(T))$ bits of space.

These classifications are subject to the condition that operations and navigations must remain efficient on the compressed version. Achieving implicit compression for trees is challenging, as the order must be preserved. An example of an implicit data structure would be a list, where the only additional required information besides the data is its length, which clearly can be described with a constant amount of bits. However, the compression algorithms presented in this thesis mostly lie between implicit and succinct representations, where the support for efficient operational queries is relaxed or left for further improvement.

4.1 Entropy of plane trees

For the models considered below, when calculating entropy, we assume a distribution based on the standard probability model. Each tree source $\mathcal{T}$ generates a labeled tree uniformly at random. We then erase the labels, effectively considering the equivalence classes of trees under root-preserving permutations, and compute the entropy of the unlabeled class. However, for each model, the resulting probability distribution over the unlabeled trees is non-uniform. The following entropy results for m-ary search trees and increasing trees originate from [22].

Remark 4.1. We will make use of the fact that an unlabeled tree t is uniquely determined by the sizes $k_1, \ldots, k_l$ of all its l subtrees dangling from the root - referred to as the **root split** - as well as by the subtrees $t_1, \ldots, t_l$ themselves. Hence

$$\mathbb{P}(T = t) = \mathbb{P}\left(\mathbf{R}_T = \mathbf{k}\right) \prod_{j=1}^{l} \mathbb{P}\left(T_j = t_j\right)$$

where $\mathbf{k} = (k_1, \ldots, k_l)$ and $\mathbf{R}_T = (R_1, \ldots, R_l)$ denotes the random variable of the root split for a random tree in T. This affects the entropy as follows, according to (1.5):

$$H(T) = H(\mathbf{R}_T, T_1, \ldots, T_l) = H(\mathbf{R}_T) + H(T_1, \ldots, T_l | \mathbf{R}_T)$$

$$= H(\mathbf{R}_T) + l \sum_{k} H(T | \#T = k) \sum_{\|\mathbf{k}'\| = \|\mathbf{k}\| - k} \mathbb{P}\left(\mathbf{R}_T = \left(k, \mathbf{k}'\right)\right) \tag{4.1}$$

where k takes on every permissible subtree size value and $\mathbf{k}' = (k_2, \ldots, k_l)$ gives the root split of the remaining subtrees when the first subtree size is fixed. The equality holds because, first, the subtrees become independent of each other once their respective sizes are fixed, and second, the distribution of one subtree is entirely determined by its size. us
With regards to the encoding schemes operating on trees of size n we need to establish the entropy rate:

Definition 4.2. Let T_n be the random variable that takes on a tree with n nodes from the tree source $\mathcal{T}$. The **entropy rate** associated to a tree source $\mathcal{T}$ is given by

$$H(\mathcal{T}) = \lim_{n \to \infty} \frac{H(N_1, \ldots, N_n)}{n} = \lim_{n \to \infty} \frac{H(T_n)}{n}$$

and returns the entropy per node.

4.2 Entropy of increasing search trees

Definition 4.3. An **m-ary search tree**, $m \geq 2$, on n keys from a totally ordered set like $\{1, \ldots, n\}$ is a rooted plane tree that is defined recursively as follows:

- If $n = 0$, the tree is empty.
- If $1 \leq n \leq m - 1$, the tree consists only of a root, where all keys are stored in the root.
- If $n \geq m$, then $m - 1$ keys are selected as pivots and stored in the root. These pivots divide the remaining $n - m + 1$ keys into m subtrees $M_1, \ldots, M_m$: If

the pivots are $p_1 < p_2 < \ldots < p_{m-1}$, then the subtrees consist, respectively, of the following keys:

$$I_1 := \{p_i \mid p_i < p_1\}, \quad I_2 := \{p_i \mid p_1 < p_i < p_2\}, \quad \ldots, \, I_m := \{p_i \mid p_{m-1} < p_i\}.$$

For each subtree M_i an m-ary search tree is then constructed recursively.

The standard probability model assumes that every permutation of the keys is equally likely. Furthermore, the specification of the pivots here is deterministic. To obtain the corresponding unlabeled m-ary search tree we remove the keys. As a result, the unlabeled representatives are non-uniformly distributed. We will illustrate this phenomenon for the next tree class in Figure 4.1.

Lemma 4.4. *[22] Let $n \in \mathbb{N}$ and S_n be the random variable that takes on an unlabeled m-ary search tree with n keys according to the mentioned non-uniform probability distribution. Then the entropy of S_n is recursively given by*

$$H(S_n) = \log_2 \binom{n}{m-1} + \frac{m}{\binom{n}{m-1}} \sum_{k=0}^{n-m+1} \binom{n-k-1}{m-2} H(S_k), \quad n \geq m$$

and

$$H(S_n) = 0, \quad n = 0, \ldots, m-1.$$

Proof. For $n = 0, \ldots, m-1$ the search tree is either empty or all keys get stored in the node, so the entropy is obviously zero. For $n \geq m$ equation (4.1) becomes

$$H(S_n) = H(\mathbf{R}_{S_n}) + m \sum_{k=0}^{n-m+1} H(S_k) \sum_{\|\mathbf{k}'\|=n-m+1-k} \mathbb{P}\left(\mathbf{R}_{S_n} = \left(i, \mathbf{k}'\right)\right) \tag{4.2}$$

The probability of the root split is

$$\mathbb{P}\left(\mathbf{R}_{S_n}\right) = \frac{1}{\binom{n}{m-1}}.$$

since the choice of the pivots fixes the subtree sizes. By plugging into the entropy formula (1.4) of Section 3 we get

$$\begin{aligned} H\left(\mathbf{R}_{S_n}\right) &= \frac{1}{\binom{n}{m-1}} \sum_{\|\mathbf{k}\|=n-m+1} \log_2 \binom{n}{m-1} \\ &= \frac{\log \binom{n}{m-1}}{\binom{n}{m-1}} \sum_{\|\mathbf{k}\|=n-m+1} 1 = \log_2 \binom{n}{m-1}. \end{aligned}$$

Having fixed the size of the first subtree and thereby determined the first pivot, we are left to select the remaining $m-2$ pivots. Thus,

$$\sum_{\|\mathbf{k}'\|=n-m+1-k} \mathbb{P}\left(\mathbf{R}_{S_n} = \left(i, \mathbf{k}'\right)\right) = \frac{\binom{n-k-1}{m-2}}{\binom{n}{m-1}}.$$

Substituting this into (4.2) gives the desired recursion. ■

The asymptotic solution of such a recurrence from Lemma 4.4 is well known and was established in [17] and [6]:

Theorem 4.5. *(Asymptotic Transfer Theorem)*
Given the recurrence

$$a_n = b_n + \frac{m}{\binom{n}{m-1}} \sum_{k=0}^{n-m+1} \binom{n-k-1}{m-2} a_k, \quad n \geq m \tag{4.3}$$

with the initial condition $a_n = b_n (= 0),\ n = 0, \ldots, m-1$. *If*

$$b_n = o(n) \quad \textit{and} \quad K := \sum_{n \geq 0} \frac{b_n}{(n+1)(n+2)} \ \textit{converges}$$

then

$$a_n = \frac{K}{\mathcal{H}_m - 1} n + o(n)$$

where $\mathcal{H}_m$ *denotes the* m*-th harmonic number.*

The following lemma will be used in the proof of the Asymptotic Transfer Theorem:

Lemma 4.6. *[17] Let*

$$F(z) = (1-z)^{-\alpha} \int_0^z \overline{B}(t)(1-t)^{\alpha} \, \mathrm{d}t$$

where $B(z) = \sum_{n \geq 0} b_n z^n$ *and* $\alpha \in \mathbb{C}$. *Then the coefficients have the asymptotics*

$$[z^n]F(z) \sim \frac{n^{\alpha-1}(1 + \mathcal{O}(n^{-1})}{(k+1)^{\alpha-1}(1 + \mathcal{O}((k+1)^{-1})}$$

Proof. Define

$$C(z) = \sum_{n \geq 0} c_n z^n = (1-z)^{-\alpha} \int_0^z \overline{B}(t)(1-t)^{\alpha} \, \mathrm{d}t.$$

Differentiating this leads to the differential equation

$$C'(z) = \sum_{n \geq 0} (n+1) c_{n+1} z^n = \alpha \frac{C(z)}{1-z} + \frac{B(z)}{1-z}, \quad C(0) = 0.$$

Which yields the following recurrence for the coefficients:

$$c_n = \frac{\alpha}{n} \sum_{k=0}^{n-1} c_k + \frac{1}{n} \sum_{k=0}^{n-1} b_k, \quad n \geq 1.$$

The recursion can be transformed into a first-order linear recurrence by applying adjacent differences:

$$n c_n - (n-1) c_n = \alpha c_{n-1} + b_{n-1} \implies c_n = \frac{(\alpha + n - 1) c_{n-1} + b_{n-1}}{n}.$$

Unfolding the recurrence leads to

$$c_n = \sum_{k=0}^{n-1} \frac{b_k}{k+1} \prod_{j=k+2}^{n} \left(1 + \frac{\alpha - 1}{j}\right), \quad n \geq 0. \tag{4.4}$$

Here,

$$\prod_{j=k+2}^{n} \left(\frac{j+\alpha-1}{j}\right) = \frac{(k+1)!}{n!} \frac{(n+\alpha-1)!}{(k+\alpha)!} = \frac{\Gamma(k+2)\Gamma(n+\alpha)}{\Gamma(n+1)\Gamma(k+\alpha+1)}.$$

which holds true for all $\alpha \notin \mathbb{Z}_{\leq 0}$. Now we make use of Stirling's formula for factorials, which also holds for complex numbers:

$$\Gamma(z+1) = z! \sim z^z e^{-z} \sqrt{2\pi z} \left(1 + \mathcal{O}\left(|z|^{-1}\right)\right),$$

for $-\pi + \delta < \arg z < \pi - \delta,\ 0 < \delta < \pi, \quad$ as $|z| \to \infty$. Recall that Stirling's formula for factorials in logarithmic form is given by

$$\log n! = \left(n + \frac{1}{2}\right) \log n - n + \frac{1}{2} \log(2\pi) + \mathcal{O}(n^{-1}).$$

Hence,

$$\begin{aligned}
\frac{\Gamma(n+1+\alpha)}{\Gamma(n+1)} &= (n+\alpha)\log(n+\alpha) - n\log n - \alpha + \frac{1}{2}\left(\log(n+\alpha-\log n)\right) + \mathcal{O}(n^{-1}) \\
&= (n+\alpha)\log\left(1+\frac{\alpha}{n}\right) + \alpha\log n - \alpha + \frac{1}{2}\log\left(1+\frac{\alpha}{n}\right) + \mathcal{O}(n^{-1}) \\
&= (n+\alpha)\left(\frac{\alpha}{n} + \mathcal{O}(n^{-2})\right) + \alpha\log n - \alpha + \frac{\alpha}{2n} + \mathcal{O}(n^{-2}) + \mathcal{O}(n^{-1}) \\
&= \alpha\log n + \mathcal{O}(n^{-1}).
\end{aligned}$$

This gives, for all $\alpha \in \mathbb{C}$,

$$\frac{\Gamma(n+\alpha)}{\Gamma(n)} = n^{\alpha-1}\left(1 + \mathcal{O}(n^{-1})\right)$$

and

$$\frac{\Gamma(k+2)}{\Gamma(k+\alpha+1)} = (k+1)^{\alpha-1}\left(1 + \mathcal{O}\left((k+1)^{-1}\right)\right)$$

analogously. ■

Proof of the Asymptotic Transfer Theorem. [17]

1. The generating function A(z): We solve the non-elementary recursion using generating functions:

$$A(z) := \sum_{n\geq 0} a_n z^n \quad \text{and} \quad B(z) := \sum_{n\geq 0} b_n z^n.$$

In the following we will use the standard notation $x^{\underline{k}} := x\cdot(x-1)\cdots(x-k+1)$ for the falling factorials, while the rising factorials $x^{\overline{k}}$ are defined analogously. Additionally, we denote the m-th derivative of the function $A(z)$ by $A^{(m)}(z)$.
Multiplying (4.3) by $n^{\underline{m-1}}$ simplifies the first fraction on the right-hand side by canceling the denominator. Then we get

$$\begin{aligned}
&\sum_{n\geq m} a_n n^{\underline{m-1}} z^{n-m+1} = \sum_{n\geq m} b_n n^{\underline{m-1}} z^{n-m+1} + m! \sum_{n\geq m} \sum_{k=0}^{n-m+1} \binom{n-k-1}{m-2} a_k \; z^{n-m+1} \\
&\implies A^{(m-1)}(z) = B^{(m-1)}(z) + m! \sum_{n\geq 0} \sum_{k=0(m)}^{n+1} \binom{n+m-k-1}{m-2} a_k \; z^{n+1} \\
&\implies A^{(m-1)}(z) = B^{(m-1)}(z) + m! \sum_{n\geq 0} \sum_{k=0(m)}^{n} \binom{m+(n-k)-2}{m-2} a_k \; z^{n} \\
&\implies A^{(m-1)}(z) = B^{(m-1)}(z) + m!(1-z)^{-(m-1)} A(z)
\end{aligned}$$

which is an Eulerian differential equation. Alternatively to [17], where they reduce the problem step by step to one with constant coefficients, one can make the known ansatz $A(z) = (1-z)^{-\alpha}$, see Section 1.3, and obtain

$$\frac{d^{m-1}}{dz} \frac{1}{(1-z)^{\alpha}} = \frac{m!}{(1-z)^{(m-1)+\alpha}}$$

which gives

$$(-1)^{m-2} \frac{(-\alpha)^{\underline{m-1}}}{(1-z)^{\alpha+m-1}} = \frac{m!}{(1-z)^{m-1+\alpha}}.$$

This further provides the indicial equation

$$\phi(\alpha) = \alpha^{\overline{m-1}} - m! = 0.$$

which has $m-1$ distinct, non-zero solutions $\alpha_1, \ldots, \alpha_{m-1}$ since a solution α_i satisfies

$$\phi'(\alpha_i) = \sum_{j=0}^{m-2} \frac{m!}{\alpha_i + j} \neq 0, \tag{4.5}$$

as proven in [29]. Hence the general homogeneous solution is given by the following sum of linearly independent solutions:

$$\sum_{k=1}^{m-1} c_k (1-z)^{-\alpha_k}.$$

By the method of variation of constants, we find a particular solution. Adjusted to the initial conditions, this yields the complete solution

$$A(z) = \sum_{k=1}^{m-1} c_k (1-z)^{-\alpha_k} + \sum_{k=1}^{m-1} \frac{(1-z)^{-\alpha_k}}{\phi'(\alpha_k)} \int_0^z B^{(m-1)}(t)(1-t)^{\alpha_k+m-2} \, \mathrm{d}t \tag{4.6}$$

where $\phi(\alpha) = \alpha^{\overline{m-1}} - m!$. The coefficients c_k are given by a linear combination of the first m terms $b_0, \ldots, b_{m-1}$ and vanish due to the initial conditions. Repeated partial integration of (4.6), using the initial conditions and denoting $\overline{B}(z)$ as the generating function starting at index $n = m$, gives

$$\begin{aligned}
A(z) &= \sum_{l=1}^{m-2} B^{(l)}(1-z)^l \sum_{k=1}^{m-1} \frac{1}{\phi'(\alpha_k)} + \sum_{k=1}^{m-1} \frac{(1-z)^{-\alpha_k}}{\phi'(\alpha_k)} (\alpha_j + 1)^{\overline{m-2}} \int_0^z B'(t)(1-t)^{\alpha_k} \,\mathrm{d}t \\
&= \sum_{k=1}^{m-1} \frac{(1-z)^{-\alpha_k}}{\phi'(\alpha_k)} \frac{\phi(\alpha_j) + m!}{\alpha_j} \int_0^z B'(t)(1-t)^{\alpha_k} \,\mathrm{d}t. \\
&= m!\overline{B}(z) \sum_{k=1}^{m-1} \frac{1}{\phi'(\alpha_k)\alpha_k} + m! \sum_{k=1}^{m-1} \frac{(1-z)^{-\alpha_k}}{\phi'(\alpha_k)} \int_0^z \overline{B}(t)(1-t)^{\alpha_k - 1} \,\mathrm{d}t \\
&= \overline{B}(z) + m! \sum_{k=1}^{m-1} \frac{(1-z)^{-\alpha_k}}{\phi'(\alpha_k)} \int_0^z \overline{B}(t)(1-t)^{\alpha_k - 1} \,\mathrm{d}t
\end{aligned}$$

where we used the following identities of the indicial polynomial

$$\sum_{k=1}^{m-1} \frac{1}{\phi'(\alpha_k)} = 0 \quad \text{and} \quad \sum_{k=1}^{m-1} \frac{1}{\alpha_k \phi'(\alpha_k)} = \frac{1}{m!}$$

derived in [29].

2. We proceed by showing that

$$[z^n]\ (1-z)^{-\alpha_i} \int_0^z \overline{B}(t)(1-t)^{\alpha_i} dt = o(n), \quad \forall i \in \{2, \ldots, m-1\}.$$

Lemma 4.6 provides us with an asymptotics for the coefficients. Another result of [29] is that the roots of ϕ are $\alpha_1 = 2$ and $\alpha_2, \ldots, \alpha_{m-1}$ have real part no larger than 2. Thus, $n^{\alpha_i}, i = 2, \ldots, m-1$ will be of smaller magnitude than $n^{\alpha_1} = n^2$. Since for all $\epsilon > 0$ it holds $n^{1-\epsilon} = o(n)$ we can further estimate (4.4) by

$$\begin{aligned}
c_n &= \mathcal{O}\left(\sum_{k=0}^{n-1} \frac{b_k}{k+1} \left(\frac{n}{k+1} \right)^{\Re(\alpha)-1} \right) \\
&= \mathcal{O}\left(n^{\Re(\alpha)-1} \sum_{k=0}^{n-1} (k+1)^{-\Re(\alpha)} \right) = o\left(n\right).
\end{aligned}$$

3. Putting everything together:
For $\alpha = 2$ the coefficients in (4.4) are of the simple form

$$\begin{aligned}
&\sum_{k=0}^{n-1} \frac{b_k}{k+1} \prod_{j=k+2}^{n} \left(\frac{j+1}{j} \right) = (n+1) \sum_{k=0}^{n-1} \frac{b_k}{(k+1)(k+2)} \\
&= n \sum_{k=m}^{n-1} \frac{b_k}{(k+1)(k+2)} + o(n).
\end{aligned}$$

Summarizing the results so far, we have

$$[z^n]A(z) = [z^n]\ \overline{B}(z) + n \cdot m! \frac{1}{\phi'(2)} \sum_{k=0}^{n-1} \frac{b_k}{(k+1)(k+2)} + o(n)$$

We evaluate $\phi'(2)$ by plugging $\lambda = 2$ into (4.5) and obtain:

$$\phi'(2) = m! \sum_{j=2}^{m} \frac{1}{j} = m!(\mathcal{H}_m - 1).$$

Finally, with the convergence assumption of the series over b_n we get the desired asymptotic estimate for the coefficients a_n:

$$a_n = \frac{n}{\mathcal{H}_m - 1} \sum_{k=0}^{\infty} \frac{b_k}{(k+1)(k+2)} + o(n). \qquad \blacksquare$$

Corollary 4.7. *The entropy rate of the unlabeled m-ary search tree is given by*

$$h_u = \lim_{n\to\infty} \frac{H(S_n)}{n} = \frac{1}{\mathcal{H}_m - 1} \sum_{k\geq m} \frac{\log \binom{k}{m-1}}{(k+1)(k+2)}$$

Proof. The result is immediately derived from the Asymptotic Transfer Theorem by applying it to

$$a_n = H(S_n) \quad \text{and} \quad b_n = \begin{cases} 0, & n < m-1 \\ \log \binom{n}{m-1}, & n \geq m-1 \end{cases}$$

noticing that

$$\log \binom{n}{m-1} \sim m \log n = o(n)$$

and

$$\sum_{n\geq 0} \frac{b_n}{(n+1)(n+2)} = \Theta\left(\sum_{n\geq 1} \frac{\log n}{n^2} \right) < \infty.$$

$\blacksquare$

4.3 Entropy of increasing trees

Definition 4.8. A ***d*-ary plane tree** is a rooted plane tree in which each internal node has exactly d children. In this section we calculate the entropy of **random unlabeled *d*-ary plane increasing trees**, as done in [22], generated according to the following model:

- The generation begins with a single external node (leaf).

- The first step in the growth process is to replace this external node with an internal node that carries the minimal label and has d leaf children.
- Then, with probability $1/d$, one of these d external nodes is selected and replaced by an internal node labeled with the next higher label that has d leaves.
- In each subsequent step, one of the external nodes (chosen with equal probability) is replaced by an internal node carrying the subsequent label with d external children.

The standard probability model assumes that each d-ary plane increasing tree of n nodes is produced with uniform probability. The corresponding unlabeled d-ary representative is obtained by removing the node labels inducing the mentioned non-uniform distribution.

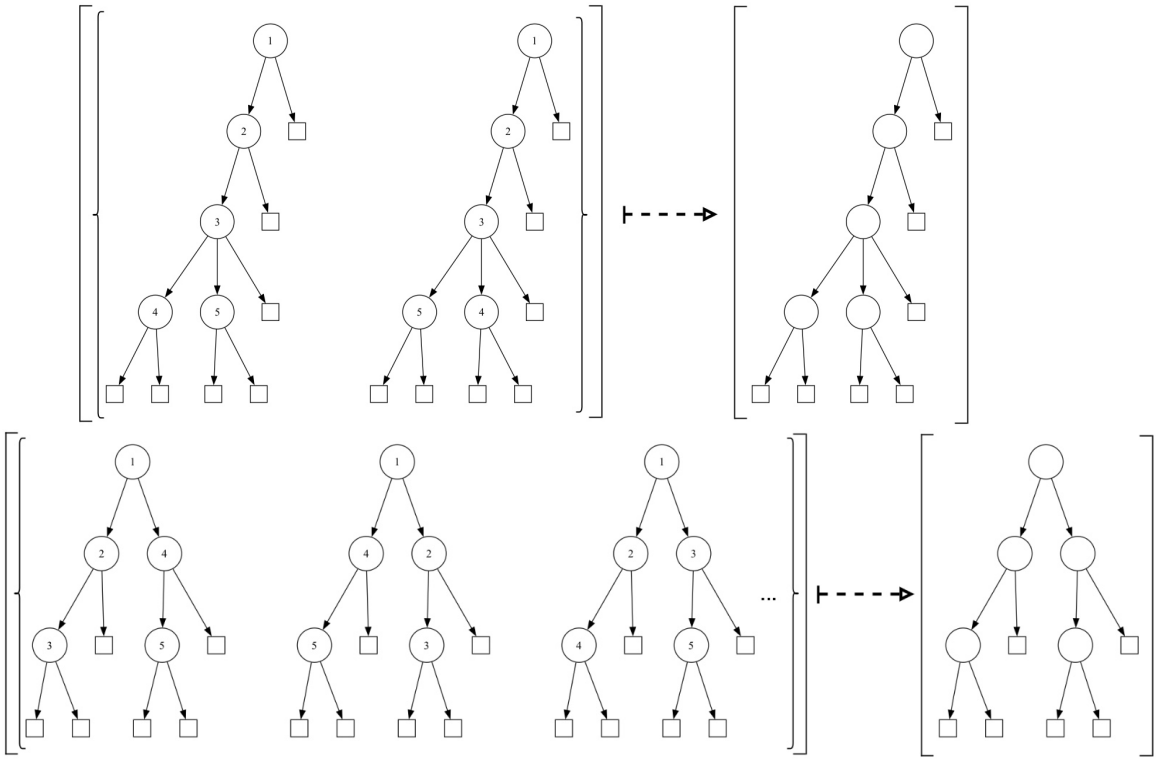


Figure 4.1: Two examples of equivalence classes established on the 2-ary plane increasing trees and their unlabeled representative; this also visualizes the arisen non-uniform distribution on the unlabeled class.

Lemma 4.9. *The family $\mathcal{D}$ of d-ary increasing plane trees has the recursive characterization*

$$\mathcal{D} = \{\epsilon\} + \left(\mathcal{Z}^{\square} \star \mathcal{D}^{d}\right)$$

which gives the nonlinear integral equation for the exponential generating function

$$D(z) = 1 + \int_0^z D(z)^d \, \mathrm{d}t, \quad D(0) = 1.$$

The coefficients are then given by

$$d_n = n! \cdot [z^n]D(z) = (-1)^n (d-1)^n \frac{\Gamma(2 - \frac{d}{d-1})}{\Gamma(2 - \frac{d}{d-1} - n)}.$$

Proof. The derivative of the integral equation for the exponential generating function gives

$$D'(z) = D(z)^d.$$

This is solved by separation of variables, which gives

$$D(z) = (1 - (d-1)z)^{-\frac{1}{d-1}}.$$

Using the binomial theorem we can evaluate the coefficients:

$$\begin{aligned}
d_n = n![z^n]D(z) &= n![z^n] \sum_{n \geq 0} (-1)^n (d-1)^n \binom{-\frac{1}{d-1}}{n} z^n \\
&= n!(-1)^n (d-1)^n \binom{-\frac{1}{d-1}}{n} \\
&= (-1)^n (d-1)^n \left(-\frac{1}{d-1}\right)^{\underline{n}} \\
&= (-1)^n (d-1)^n \frac{\Gamma\left(\frac{1}{1-d} + 1\right)}{\Gamma\left(\frac{1}{1-d} - n + 1\right)} \\
&= (-1)^n (d-1)^n \frac{\Gamma\left(2 - \frac{d}{d-1}\right)}{\Gamma\left(2 - \frac{d}{d-1} - n\right)}.
\end{aligned}$$

■

Lemma 4.10. *Let $n \in \mathbb{N}$ and D_n be the random variable supported on the unlabeled d-ary increasing trees with n nodes $\mathcal{D}_n$. The entropy of the root split $H(\mathbf{R}_{D_n})$ is trivially zero for $n = 0, 1$ and for $n \geq 2$ recursively given by*

$$H(\mathbf{R}_{D_n}) = \log_2 \left(n \frac{d_n}{n!}\right) - d \sum_{k=0}^{n-1} p_{n,k} \log_2 \left(\frac{d_k}{k!}\right).$$

where the d_n are the coefficients derived in Lemma 4.9 and

$$p_{n,k} = \sum_{\|\mathbf{k}'\| = n-1-k} \mathbb{P}(\mathbf{R}_{D_n} = (k, \mathbf{k}')).$$

Proof. Let $\mathbf{k} = (k_1, \ldots, k_d)$.The probability of the root split is

$$\mathbb{P}\left(\mathbf{R}_{D_n} = \mathbf{k}\right) = \binom{n-1}{k_1, \ldots, k_d} \frac{d_{k_1} \ldots d_{k_d}}{d_n}. \tag{4.7}$$

Analogously to (4.1) we derive the following after plugging into the entropy formula:

$$H\left(\mathbf{R}_{D_n}\right) = -d\sum_{k=0}^{n-1} \log_2 \mathbb{P}(\mathbf{R}_{D_n} = k) p_{n,k}.$$

Observe that since k is chosen out of $\{0, \ldots, n-1\}$ we have

$$\mathbb{P}(\mathbf{R}_{D_n} = k) = \frac{1}{n}\frac{d_k}{k!}\frac{n!}{d_n}.$$

Hence,

$$H\left(\mathbf{R}_{D_n}\right) = -d\sum_{k=0}^{n-1} \left(\log_2\left(\frac{d_k}{k!}\right) - \log_2\left(n\frac{d_n}{n!}\right)\right) p_{n,k}.$$

Observe that $\sum_{k=0}^{n-1} p_{n,k} = \frac{1}{d}$ because it expresses the probability of choosing one of the d trees to have exactly k nodes. ■

Lemma 4.11. *Let $n \in \mathbb{N}$ and D_n be supported on the tree source of unlabeled d-ary increasing trees, $d \geq 2$. Then the entropy of D_n is recursively given by*

$$H(D_n) = H\left(\mathbf{R}_{D_n}\right) + d\sum_{k=0}^{n} H(D_k) p_{n,k} \tag{4.8}$$

where $p_{n,k}$ is defined in the previous lemma and given explicitly by

$$p_{n,k} = \frac{(\alpha-1)}{n} \cdot \frac{n!}{k!} \cdot \frac{\Gamma(k+\alpha-1)}{\Gamma(n+\alpha-1)}, \quad \textit{with} \quad \alpha = \frac{d}{d-1}. \tag{4.9}$$

Proof. The recurrence for the entropy is given by (4.1). From (4.7) we get

$$p_{n,k} = \binom{n-1}{k}\frac{d_k}{d_n} \sum_{k_2+\ldots+k_d=n-1-k} \binom{n-1-k}{k_2, \ldots, k_d} d_{k_2} \cdots d_{k_d}.$$

From Lemma 4.9 we know the generating function of the d-ary increasing trees $I(z)$ and since

$$\begin{aligned}\sum_{k_2+\ldots+k_d=n-1-k} \binom{n-1-k}{k_2, \ldots, k_d} d_{k_2} \cdots d_{k_d} &= \frac{1}{(n-1-k)!}\left[z^{n-1-k}\right] D(z)^{d-1} \\ = \frac{1}{(n-1-k)!}\left[z^{n-1-k}\right](1-(d-1)z)^{-1} &= (d-1)^{n-1-k}\end{aligned}$$

we have

$$p_{n,k} = \frac{(n-1)!}{k!}\frac{d_k}{d_n}(d-1)^{n-1-k}.$$

Plugging in the coefficients we derived in Lemma 4.9 yields

$$\begin{aligned}p_{n,k} &= \frac{(n-1)!}{k!}(-1)^{k-n}\frac{\Gamma(2-\alpha-n)}{\Gamma(2-\alpha-k)}(d-1)^{-1} \\ &= (-1)^{k-n}\frac{\alpha-1}{n}\frac{n!}{k!}\frac{\Gamma(2-\alpha-n)}{\Gamma(2-\alpha-k)}.\end{aligned}$$

With the reflection formula for the Gamma function

$$\Gamma(z-n) = \frac{(-1)^n \pi}{\Gamma(n+1-z)\,\sin(\pi z)}$$

we arrive at the desired result. ■

Theorem 4.12. *The entropy recurrence of Lemma 4.11 admits the exact solution*

$$H(D_n) = H\left(\mathbf{R}_{D_n}\right) + \alpha(n+\alpha-1)\sum_{k=0}^{n-1} \frac{H\left(\mathbf{R}_{D_k}\right)}{(k+\alpha-1)(k+\alpha)}.$$

The entropy rate h_d of the unlabeled d-ary plane increasing trees is therefore

$$h_d = \alpha \sum_{k=0}^{n-1} \frac{H\left(\mathbf{R}_{D_k}\right)}{(k+\alpha-1)(k+\alpha)}$$

Proof. The recurrence can be solved using the standard technique of finite differences for linear recurrences, along with well-known factorial and Gamma function identities. Since the entropy of a random variable is upper bounded by the logarithm of the cardinality of its image, we have

$$H(\mathbf{R}_{D_n}) \leq \log_2(n^d) = o(n),$$

which implies that it is asymptotically negligible. Therefore, the entropy rate follows as a corollary of the exact solution. ■

The results derived for the unlabeled d-ary plane increasing trees can be generalized to the broader class of **unlabeled degree-unconstrained plane increasing trees**, as shown in [22], with a probability distribution induced by the following generation:

- The generation begins with a single internal node carrying the smallest label.
- Then a child internal node is attached carrying the next highest label.
- The $(n+1)$th node, carrying the next highest label is attached to each previous node with probability $1/n$.

Each tree generated by this process then becomes unlabeled again. Notice the absence of leaves.

Lemma 4.13. *The family $\mathcal{U}$ of degree unrestricted increasing plane trees has the recursive characterization*

$$\mathcal{U} = \left(\mathcal{Z}^{\square} \star Seq_{\geq 0}(\mathcal{U})\right)$$

which gives the nonlinear integral equation for the exponential generating function

$$U(z) = \int_0^z \frac{1}{1-U(z)}\,\mathrm{d}t, \quad U(0) = 0.$$

The coefficients are then given by

$$u_n = n! \cdot [z^n]U(z) = (2n-3)!! = \frac{n!}{n2^{n-1}}\binom{2n-2}{n-1}.$$

Proof. The derivative of the integral equation for the exponential generating function gives

$$U'(z) = \frac{1}{1-U(z)}.$$

This is solved using the method of separation of variables, yielding

$$U(z) = 1 - \sqrt{1-2z}.$$

Again, using the binomial theorem, we evaluate the coefficients:

$$\begin{aligned} u_n = n![z^n]U(z) &= n![z^n]\ 1 - \sum_{n\geq 0}\binom{\frac{1}{2}}{n}(-1)^n 2^n z^n \\ &= n![z^n]\ -\sum_{n\geq 1}\frac{(-1)^{2n}(2n)!}{(1-2n)(n!)^2 4^n}2^n z^n \\ &= n![z^n]\ \sum_{n\geq 1}\frac{(-1)^{2n+2}(2n-2)!}{n((n-1)!)^2 2^{n-1}}z^n \\ &= \frac{1}{n2^{n-1}}\binom{2n-2}{n-1} = (2n-3)!! \end{aligned}$$

■

The entropy rate for the unlabeled degree-unconstrained plane increasing trees $\mathcal{U}$ can be derived from the results for the d-ary family straightforwardly:

1. The entropy recurrence analogue to (4.8) for S_n is given by

$$H(S_n) = \sum_{d=1}^{n-1} H\left(\mathbf{R}_{S_{n,d}}\right)\mathbb{P}(R_n = d) + \sum_{d=1}^{n-1} d \sum_{k=1}^{n-d} H(S_k) q_{n,k,d} \tag{4.10}$$

   where R_n is the random variable representing the root out-degree, that can take on values from $\{1,\ldots,n-1\}$, for a tree $t \in \mathcal{S}_n$ with n nodes and $\mathbf{R}_{S_{n,d}} : \mathcal{S}_{n,d} \to \{1,\ldots,n-d\}^d$ is the root split random variable, that assumes a d-fold root split.

2. An explicit formula for $q_{n,k,d}$, analogous to (4.9), can be derived in a similar way. The same applies to the recurrence relation for the root split.

3. The recurrence (4.10) can be solved explicitly in the same way the recurrence for the d-ary case was solved.

4. The entropy rate h_u for the unlabeled degree-unconstrained plane increasing trees is given by

$$h_u = \frac{1}{2}\sum_{k=2}^{\infty} \frac{\sum_{d=1}^{k-1} H\left(\mathbf{R}_{S_{n,d}}\right) \mathbb{P}(R_n = d)}{(k-\frac{1}{2})(k+\frac{1}{2})}$$

4.4 Compression algorithm for m-ary search trees

We adapt the arithmetic-coding based algorithm for d-ary unlabeled increasing trees ([22, Alg. 1]) to m-ary unlabeled increasing search trees. In doing so, we define a total order on the set of unlabeled m-ary search trees and compute the necessary probabilities for arithmetic coding.

Numerically computed entropy rate of the m-ary search tree

The entropy rate h_u of an m-ary search tree, as defined earlier, represents the average amount of information needed (per node) to describe a tree of $\mathcal{S}$ for large tree sizes. It is well known that in an m-ary search tree the number of nodes N is a random variable related to the number of keys n by $\mathbb{E}[N] \sim \phi_m n$, [22]. Table 4.1 presents the numerically approximated entropy rate h_u, defined earlier, and the normalized ratio h_u/ϕ_m for $m = 2, 3, 4, 15$, where ϕ_m is given by $\phi_m = (2\mathcal{H}_m - 2)^{-1}$.

m	h_u	h_u/ϕ_m
2	1.736	1.736
3	1.5	2.501
4	1.356	2.939
15	0.888	4.119

Table 4.1: Values of h_u and h_u/ϕ_m for $m = 2, 3, 4, 15$

As m increases, ϕ_m becomes smaller, resulting in larger values of h_u/ϕ_m. The decreasing entropy rate for larger values of m is intuitive since more keys are stored in the nodes which reduces the structural uncertainty relative to the size.

Total Ordering on Unlabeled m-ary increasing Search Trees

For a given number of keys n, we define a total order on the set $\mathcal{S}_n$ of unlabeled m-ary search trees with n keys. It compares trees first by their root split and then by their subtree root splits in a left-to-right order. Each tree $s \in \mathcal{S}_n$ has a root with up to m subtrees. The total order is defined recursively as follows:

- For two trees $s_1, s_2 \in \mathcal{S}_n$ with root subtree sizes $(l_1, l_2, \ldots, l_m)$ and $(k_1, k_2, \ldots, k_m)$, respectively, $s_1 < s_2$ if:
  - $(l_1, l_2, \ldots, l_m) < (k_1, k_2, \ldots, k_m)$ in lexicographic order, or
  - $(l_1, l_2, \ldots, l_m) = (k_1, k_2, \ldots, k_m)$, and the first subtree of s_1 is smaller than the first subtree of s_2 (recursively), or
  - $(l_1, l_2, \ldots, l_m) = (k_1, k_2, \ldots, k_m)$, the first subtrees are equal, and the second subtree of $s_1 <$ the second subtree of s_2, and so on, up to the m-th subtree.
- For $n = 0$ (empty tree) or $1 \leq n < m$ (root-only tree), there is a single tree, so the order is trivial.

Next, we map each subtree to a unique sub-interval of $[0, 1)$, based on the total order, using arithmetic coding, a well-known coding scheme in lossless data compression in information theory mentioned in Section 3 that encodes complete messages (trees here) instead of components of messages. Each unlabeled m-ary increasing search tree on n keys gets mapped to a unique subinterval $[a, a + p] \subseteq [0, 1]$, where a represents the relative position of the tree in the total order. The length of the subinterval p represents the probability that the random variable S_n, specified earlier, emits the tree.

The latter, p, is computed recursively, using the previously established recurrence

$$\mathbb{P}(S_n = s) = \mathbb{P}(\mathbf{R}_{U_n} = \mathbf{k}) \prod_{l=1}^{m} \mathbb{P}(S_{n,l} = s_l), \quad \mathbb{P}(\mathbf{R}_{U_n} = \mathbf{k}) = \frac{1}{\binom{n}{m} - 1}.$$

The relative position of a tree $s \in \mathcal{S}_n$ is computed as $\mathbb{P}(S_n < s)$ recursively.

Algorithm 3 Unlabeled m-ary Search Tree Compression

function CompressMTree($s \in \mathcal{S}_n$)
  $[a, b) \leftarrow$ Explore(root of $s, [0, 1)$)
  return first $\lceil -\log_2(b-a) \rceil + 1$ bits of $(a+b)/2$
end function
function Explore($v \in s, [l, r) \subseteq [0, 1)$)
  visited(v) $\leftarrow$ true
  $n \leftarrow$ number of keys in subtree rooted at v
  if $n < m$ **then**
    return $[l, r)$ ▷ Leaf or root-only node, no split
  end if
  $(l_1, \ldots, l_m) \leftarrow$ sizes of subtrees of v
  $a \leftarrow l + (r-l) \cdot \text{CalculateIntervalBegin}(n, l_1, \ldots, l_m)$
  $p \leftarrow (r-l) \cdot \text{CalculateSplitProbability}(n, l_1, \ldots, l_m)$
  $I_{\text{new}} \leftarrow [a, a+p)$
  for all u descendant of v **do**
    if not visited(u) **then**
      $I_{\text{new}} \leftarrow \text{Explore}(u, I_{\text{new}})$
    end if
  end for
  return I_{new}
end function
function CalculateSplitProbability($n, k_1, \ldots, k_m$)
  if $n < m$ **then**
    return 1
  end if
  if $k_1 + \cdots + k_m \neq n - m + 1$ **then**
    return 0
  end if
  return $\dfrac{1}{\binom{n}{m-1}}$
end function
function CalculateIntervalBegin($n, j_1, \ldots, j_m$)
  if $n < m$ **then**
    return 0
  end if
  return $\displaystyle\sum_{\substack{(k_1,\ldots,k_m)<(j_1,\ldots,j_m) \\ k_1+\cdots+k_m=n-m+1}} \frac{1}{\binom{n}{m-1}}^{*}$
end function

Running Time

The algorithm performs a depth-first traversal of the tree, visiting each node once. Let N be the number of nodes in the tree.

- For each node, `Explore` calls `CalculateSplitProbability` and `CalculateIntervalBegin` once.
- `CalculateSplitProbability`: Clearly $O(1)$.
- `CalculateIntervalBegin`: Naively, the expression marked with * sums over all $(k_1, \ldots, k_m) < (l_1, \ldots, l_m)$ with $k_1 + \cdots + k_m = n - m + 1$, leading to $\mathcal{O}(n^{m-1})$ time per call for computing all admissible integer partitions. Instead, we simplify:

$$\sum_{\substack{(k_1,\ldots,k_m)<(j_1,\ldots,j_m)\\ k_1+\cdots+k_m=n-m+1}} 1 = \sum_{i=1}^{m} \sum_{k_i=0}^{j_i-1} \sum_{k_{i+1}+\cdots+k_m=n-m+1-\sum_{l=1}^{i-1} j_l-k_i} 1$$

  where we now subsequently compute the admissible tuples by moving up the positions in the ordering. Note that for a fixed i, $k_1, \ldots, k_{i-1}$ equal $j_1, \ldots, j_{i-1}$ otherwise the lexicographic order would not differ at position i. The inner-most sum is given by (stars and bars):

$$\binom{n-m+1-\sum_{l=1}^{i-1} j_l - k_i + (m-i) - 1}{m-i-1} = \binom{n-\sum_{l=1}^{i-1} j_l - k_i - i}{m-i-1},$$

  except for the boundary case where $i = m$ and this sum equals 1. This reduces time complexity to $\mathcal{O}(m^2 n) = \mathcal{O}(n)$.

- `Total Complexity`: The overall time complexity therefore sums up to $\mathcal{O}(n^2)$ since there are $\mathcal{O}(n)$ calls to each of the two functions.

Entropy

The code length is $\lceil -\log_2 \mathbb{P}(s) \rceil + 1$, which is at most 2 bits above $-\log_2 \mathbb{P}(s)$. The expected code length is:

$$\begin{aligned} &-\sum \mathbb{P}(S_n = s) \left(\lceil -\log_2(\mathbb{P}(S_n = s) \rceil + 1 \right) \\ &\leq -\sum \mathbb{P}(S_n = s) \left(\log_2(\mathbb{P}(S_n = s) + 2 \right) = H(S_n) + 2 \end{aligned}$$

This shows that the compression is optimal within a constant number of bits of the entropy.

4.5 Hypersuccinct compression for binary trees

In this section, we explore the hypersuccinct tree compression developed in [31] for random unlabeled binary tree sources. The compression method partitions the tree into micro-trees, which are efficiently stored in a lookup table by encoding them using a Huffman code and optimizing the lookup process through the "Four-Russians trick". The authors ironically call this the “Four Russians and One American” technique. The partitioning algorithm, developed in [16], and the Huffman code automatically adapt to the given tree source, making a manual tailoring of the compression scheme to specific tree sources obsolete. Moreover, constant-time query support on the word-RAM model is guaranteed by this succinct data structure. Although we will not cover them here, the hypersuccinct encoding has been shown in [31] to be optimal and universal for a variety of fixed-size sources, including uniformly random Motzkin trees, AVL trees, and random binary search trees.

We denote with $\mathbf{H}$ the hypersuccinct source code $\mathbf{H} : \mathcal{T} \to \{0,1\}^*$ which maps each tree t of n nodes from the tree source $\mathcal{T} = \hat{\mathcal{B}} \cup \{\epsilon\}$ of uniformly random incomplete binary trees (where we allow the empty tree) to a finite-length bit sequence.

Compression Steps

Step 1: Construction of the micro-trees

In the first step, we partition the nodes of a binary tree $t \in \mathcal{T}_n$ into $m = \Theta(n/\log n)$ node-disjoint micro-trees $\sigma_1, \ldots, \sigma_m$, each containing at most $\mu = \lceil \frac{1}{4} \log_2(n) \rceil$ nodes using the Farzan-Munro algorithm ([16], where it is listed in detail). To summarize the procedure: process a tree t bottom-up and declare a subtree (containing all descendants of a node up to placeholder nodes) starting at a node $v \in t$ as a micro-tree if it contains a minimum of $\mu/2$ nodes. The micro subtree is then stored in a separate set Σ_μ, and replaced in t by a single placeholder node v' with a size value equal to the size of this fringe subtree. The parent micro subtree does not contain this node v', but is rather connected to it via a single edge. The algorithm recursively applies the same process to the contracted tree. The recursion continues until the entire tree is partitioned into micro-trees. The binary tree with m nodes, where all micro-trees of t are represented by placeholder nodes, is called its top-tier $\mathcal{Y}_t$.

Each micro-tree shares at most three edges with other micro-trees: one to a parent, one to a left-child, and one to a right-child micro-tree.

Step 2: Huffman Encoding of the micro-trees

The Huffman code $C : \Sigma_\mu \to \{0,1\}^*$ is generated according to Definition P22 with the probabilities p being the empirical occurrence probabilities of the micro-trees. To manage the overhead of encoding rare micro-trees - which get assigned longer lengths according to the principles of Huffman coding - a truncation technique is applied.

Definition 4.14. The **balanced parentheses encoding (BPE)** represents a tree as a sequence of opening and closing parentheses. For binary trees t: $\mathrm{BP} : B \to \{(,)\}^*$, is inductively defined as follows:

- $\mathrm{BP}(t) = \varepsilon$, if t is the empty tree.
- $\mathrm{BP}(t) = (\mathrm{BP}(t_l))\,\mathrm{BP}(t_r)$, otherwise.

Here, ε denotes the empty string, t_l represents the left subtree, and t_r represents the right subtree of t. The balanced parenthesis encoding of a binary tree t obviously has length $2|t|$.

Lemma 4.15. *[31, p. 30] Let $c(\sigma) \in C$ be the codeword for $\sigma \in \Sigma_\mu$ in the Huffman code, $n = |\sigma|$. Define $L = 2n + 2\lfloor \log_2(n+1) \rfloor + 1$, and let a micro-tree σ be* rare *if $|c(\sigma)| \geq L$. Let $b(\cdot)$ and $u(\cdot)$ denote the binary and unary extension functions, respectively. The truncated Huffman code $\hat{C} : \Sigma_\mu \to \{0,1\}^*$ is defined as*

$$\hat{C}(\sigma) = \begin{cases} 0 \cdot u(b(n+1)) \cdot b(n+1) \cdot \mathrm{BP}(\sigma), & \sigma \text{ is rare,} \\ 1 \cdot c(\sigma), & \text{otherwise,} \end{cases}$$

where $\cdot$ denotes string concatenation. Then, $\hat{C}$ is uniquely decodable and prefix-free. The length of a code for m micro-trees, $(\hat{C}(\sigma_i))_{i=1}^m$, is no larger than $|(C(\sigma_i))_{i=1}^m| + m$ and restricts the codeword lengths of a rare micro-tree to $L+1$.

Proof. The unary and binary extensions are necessary for unique decodability because $\mathrm{BP}(\sigma)$, with length $2n$, varies with σ. Using only $0 \cdot \mathrm{BP}(\sigma)$ for rare micro-trees would lead to ambiguity. The binary extension $b(n+1)$, a string of $\lfloor \log_2(n+1) \rfloor + 1$ bits, encodes $n+1$, allowing the decoder to compute the length of $\mathrm{BP}(\sigma)$. Since the length of $b(n+1)$ also varies with σ, the unary extension $u(b(n+1))$, a string of $\lfloor \log_2(n+1) \rfloor$ zeros (followed by a 1), encodes this length, ensuring the decoder can locate the start of $b(n+1)$ and subsequently $\mathrm{BP}(\sigma)$. In the other case, since C is a Huffman code, which is uniquely decodable and prefix-free, $\hat{C}$ is as well. In the first case, where σ is rare, the codeword length is truncated to $1 + \lfloor \log_2(n+1) \rfloor + (\lfloor \log_2(n+1) \rfloor + 1) + 2n = L + 1$. ■

Remark 4.16. Encoding $n+1$ instead of n in the binary expansion follows the convention in the case where the tree might be empty. The technique of encoding a natural number as a concatenation of its unary and binary expansion is known as **Elias encoding**.

Step 3: Encoding the micro-trees top-tier $\mathcal{Y}$

The ordering of micro-trees is recovered by the depth-first ordered BPE of $\mathcal{Y}$, together with the encodings $\hat{C}(\sigma_i)$ for all micro-trees σ_i listed in a matching depth-first order. This ensures that the decoder can reconstruct the hierarchical structure of t. To fully recover the parent-child relationships between different micro-trees we need to transmit the positions of the edges in a parent micro-tree that connect to its left and right child micro-trees, if such children exist. Since each micro-tree can have at most two children and consists of at most μ nodes, we require $2(\lfloor \log_2(\mu+1) \rfloor + 1)$ bits (depth-first order) per micro-tree to encode this information.

In total, the hypersuccinct encoding $\mathbf{H}(t)$ of a tree t consists of the concatenation of the tree and top-tier sizes n and m (in Elias code), followed by the BPE of $\mathcal{Y}$, a list of all codewords in $\hat{C}$ together with their corresponding micro-tree translations (in Elias code concatenated with its BPE), and finally the sequence of ordered micro-tree codes and child occurrence information.

Theorem 4.17. *[31, 16] Consider a binary tree $t \in \mathcal{T}_n$, partitioned into $m = \Theta(n/\log n)$ node-disjoint micro-trees $\sigma_1, \ldots, \sigma_m$, each containing at most $\mu = \lceil \frac{1}{4} \log_2 n \rceil$ nodes as in Step 1. Let Σ_μ denote the set of all possible micro-trees with up to μ nodes, and define the truncated Huffman code $\hat{C} : \Sigma_\mu \to \{0,1\}^*$ as in step 2. Then, there exists a data structure occupying $o(n)$ additional bits that supports a variety of queries in constant-time, and the size of the encoding satisfies:*

$$|\mathbf{H}(t)| \leq 2n + \mathcal{O}\left(\frac{n \log\log n}{\log n}\right).$$

Proof. We first compute the size of $\mathbf{H}(t)$, starting with the sizes n and m. Encoding n in Elias code takes $2\lfloor \log_2 n \rfloor + 1$ bits, and since $m = \Theta(n/\log n)$, encoding both requires at most $2\log_2 n + \mathcal{O}(1)$, which is $\mathcal{O}(\log_2 n)$. The BPE of $\mathcal{Y}$ requires $2m = \Theta(n/\log n)$ bits.

Next, we address the size of the micro-tree codebook. We can find a much tighter upper bound for the number of micro-trees in Σ_μ than $\Theta(n/\log n)$ if we take into account an upper bound for the counting sequence C_n of binary trees, $C_n \leq 4^n$:

$$|\Sigma_\mu| \leq \sum_{s=0}^{\mu} C_s \leq \sum_{s=0}^{\mu} 4^s = \frac{4^{\mu+1}-1}{4-1} < \frac{4}{3} \cdot 4^{\frac{1}{4}\log_2 n + 1} \in \mathcal{O}(\sqrt{n}).$$

By Lemma 4.15 we can bound the size of each codeword by $2\mu + 2\log_2(\mu+1) + 2 \in \mathcal{O}(\mu) = \mathcal{O}(\log n)$. Hence we need $\mathcal{O}(\sqrt{n}\log n)$ bits in space to list all codewords for the micro-trees in Σ_μ. The translation, the encoding of the micro-trees, takes the same amount. Hence, we have $\mathcal{O}(\sqrt{n}\log n)$ bits in total.

For the micro-tree code sequence $\hat{C}(\sigma_1), \ldots, \hat{C}(\sigma_m)$ (in depth-first order), we know by Lemma 4.15 that the code length is restricted to

$$\sum_{i=1}^{m} |C(\sigma_i)| + \Theta(n/\log n).$$

The entropy of random binary trees is approximated by using their counting sequence, the Catalan numbers C_n, together with Stirling's approximation for factorials, and is given by

$$\begin{aligned} H(\mathcal{T}_n) &= \sum_{t \in \mathcal{T}_n} \mathbb{P}(T_n = t) \log_2 \left(\frac{1}{\mathbb{P}(T_n = t)} \right) = \log_2(C_n) \\ &= \log_2 \left(\frac{1}{n+1} \binom{2n}{n} \right) \sim 2n - 2\log_2(n) + \mathcal{O}(1). \end{aligned}$$

due to a high amount of cancellations. By the optimality of the Huffman code we can easily upper bound the micro-tree encoding by

$$\sum_{i=1}^{m} |C(\sigma_i)| \leq 2n + \mathcal{O}\left(\frac{n \log\log n}{\log n} \right).$$

A strict justification, in all details, for this requires the definition of empirical entropy and extensive, but standard, information-theoretic arguments, which can be found in [31, p.32]. The final step requires $2(\lfloor \log_2(\mu+1) \rfloor + 1)$ bits to encode the positions of edges to its children. This is $\mathcal{O}(\log\log n)$ bits per micro-tree, so the total is $m \cdot \mathcal{O}(\log\log n) = \Theta(n/\log n) \cdot \mathcal{O}(\log\log n) = \mathcal{O}\left(\frac{n\log\log n}{\log n}\right)$. Since $\mathcal{O}(\log n),\ \mathcal{O}(\sqrt{n}\log n),\ \mathcal{O}(n/\log n) \in \mathcal{O}\left(\frac{n\log\log n}{\log n}\right)$, we have established the claimed bound.

In Section 4 of [16] a lookup table that enables constant-time for intra micro-tree queries is constructed using dictionaries for root connections and a pointer network for parent-child edges, both contributing $o(n)$ bits. So all that is left to show, is that we can map an index $i \in \{1, \ldots, m\}$ to $\mathrm{BP}(\sigma_i)$ in constant time. This involves storing the mapping $i \mapsto \hat{C}(\sigma_i)$ as a standard variable-cell array that allows constant time mapping [31, p.25] and takes $\sum_{i=1}^{m} |\hat{C}(\sigma_i)| + O(n\log\log n/\log n)$ bits (already included). Secondly, we need the codebook lookup for $\hat{C}(\sigma_i) \mapsto \mathrm{BP}(\sigma_i)$, which takes $O(\sqrt{n}\ P(\log n))$ bits due to the length restriction of $\hat{C}$, where $P(\log(n))$ is some polynomial in $\log(n)$. Since all time-complexity statements assume the word-RAM model with word size $\Theta(\log n)$ the mapping of these bit strings fits within a constant number of words. ■

Remark 4.18. Together with the lookup table occupying $o(n)$ additional bits, the hypersuccinct encoding scheme is turned into a succinct data structure with constant time queries on a variety of navigational operations, listed in [31, Table 1], including parent, child, depth, ancestor and many more. The balanced parenthesis encoding which uses $2n$ bits is an optimal compressed representation for binary and plane trees since $\log(C_n) \sim 2n$. The compression techniques used for the binary tree source are also used for the hypersuccinct encoding of the other plane trees in [31]. The universality comes from the fact that the Huffman code adapts to the frequencies of micro-trees of the specific tree being compressed independent of its source and that the Farzan-Munro algorithm is by design universal, as well as the other compounds.

A brief outlook on further developments

In this thesis, we have explored several important tree compression methods, such as DAG compression, tree grammars, and entropy based compression methods for a variety of tree classes. However, not all major compression forms were covered in detail. One compression method we left out, developed by Bille et al. in 2015 [1] and later improved by Hübschle-Schneider and Raman in their revisited work [23], is top tree compression.
This method expanded on DAG compression, as mentioned in the introduction. A labeled tree t of size n is transformed into a new tree t', called top tree, where tree pattern repeats (connected subgraphs identical in structure and labels) form clusters. This transformed tree is then compressed into a top DAG using classical DAG compression. It is then shown that this representation supports navigational queries in $O(\log n)$ time. The size of the top DAG found by Billie was $O\left(\frac{n}{\log_\sigma^{0.19} n}\right)$, where σ is the alphabet size. This was improved by Hübschle-Schneider and Raman to $O\left(\frac{n}{\log_\sigma n} \cdot \log\log_\sigma n\right)$ which reduces the gap to the optimal information-theoretic lower bound of $\Omega\left(\frac{n}{\log_\sigma n}\right)$ to $O(\log\log_\sigma n)$.
This improvement stems from its exploitation of tree pattern repetitions, which DAG compression overlooks. Additionally, Bille et al. proved that the size of the resulting top tree DAG is — for any tree — at most a logarithmic factor larger than that of its minimal DAG.

Another example is TreeRePair, introduced by Lohrey et al. in 2013 [28]. This tree grammar, that was not covered in the tree grammar section of this thesis, demonstrated good compression ratios on XML document trees. Building on the RePair algorithm for string compression, TreeRePair constructs TSLPs in linear time, achieving compression by iteratively replacing the most frequent digrams (pairs of a parent node label, a child index, and a child node label) with new non-terminals. While its navigational performance is slower than common succinct compressions the biggest advantage of TreeRePair is its ability to produce one of the smallest known queryable memory representation of plane trees for tree grammars.

The hypersuccinct encoding scheme we briefly covered at the end was even further improved, with a particular focus on AVL trees trees, by the work of [7]. The hypersuccinct paper established a universal tree source code for optimal compressed tree data structures while supporting a wide range of operations in constant time. However, it imposed stringent conditions, including the "tamed source" requirement, which imposes non-trivial restrictions on the fringe subtrees. In [7], Chizewer et al. were able to relax this constraint by one condition, introducing the concept of "weakly tame" tree classes. They also gave a new succinct encoding for these weakly tame classes, which eliminates the need for the pointers to "parent-child edges" used in the hypersuccinct framework, simplifying the data structure, while maintaining constant-time queries in the same word-RAM model.

For AVL trees specifically, [7] rigorously derives the asymptotic for the counting sequence, which has been conjectured in the 1980s (this conjectured number was used in [31]), and thereby derives the information-theoretic lower bound required for encoding this tree source, a log-linearity constant of approximately 0.938 bits per node, with methods of analytic combinatorics.

Bibliography

[1] Philip Bille et al. "Tree Compression with Top Trees". In: *Information and Computation* 243 (2015), pp. 166–177. DOI: 10.1016/j.ic.2014.12.012.

[2] Olivier Bodini et al. "Compaction for Two Models of Logarithmic-Depth Trees: Analysis and Experiments". In: *Random Structures & Algorithms* 61.1 (2022), pp. 31–61. DOI: 10.1002/rsa.21056.

[3] Mireille Bousquet-Mélou et al. "XML Compression via Directed Acyclic Graphs". In: *Theory of Computing Systems* 57.4 (2015), pp. 1322–1371.

[4] Peter Buneman, Martin Grohe, and Christoph Koch. "Path Queries on Compressed XML". English. In: *Proceedings of the 29th Conference on Very Large Data Bases*. 2003, pp. 141–152.

[5] Kim Casel et al. "On the Complexity of Grammar-Based Compression over Fixed Alphabets". In: *Leibniz International Proceedings in Informatics (LIPIcs)*. Vol. 55. 2016, p. 122. DOI: 10.4230/LIPIcs.ICALP.2016.122.

[6] Hsiu-Hung Chern and Hsien-Kuei Hwang. "Phase Changes in Random m-ary Search Trees and Generalized Quicksort". In: *Random Structures & Algorithms* 19.3-4 (2001). Analysis of Algorithms (Krynica Morska, 2000), pp. 316–358. DOI: 10.1002/rsa.1008.

[7] Jeremy Chizewer et al. "Enumeration and Succinct Encoding of AVL Trees". In: *35th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA 2024)*. Ed. by Cécile Mailler and Sebastian Wild. Vol. 302. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 2:1–2:12. ISBN: 978-3-95977-329-4. DOI: 10.4230/LIPIcs.AofA.2024.2. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.AofA.2024.2.

[8] Michalis Christou et al. "Computing All Subtree Repeats in Ordered Ranked Trees". In: *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 338–343. DOI: 10.1007/978-3-642-24583-1_33.

[9] Michalis Christou et al. “Computing All Subtree Repeats in Ordered Trees”. In: *Information Processing Letters* 112.24 (2012), pp. 958–962. DOI: 10.1016/j.ipl.2012.09.017.

[10] Hubert Comon et al. *Tree Automata Techniques and Applications*. Available at HAL: https://hal.archives-ouvertes.fr/hal-03367725. 2008.

[11] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 2nd. Wiley-Interscience, 2006. ISBN: 978-0-471-24195-9.

[12] P. J. Downey, R. Sethi, and R. E. Tarjan. “Variations on the common subexpression problem”. In: *Journal of the ACM* 27 (4 1980), pp. 758–771. DOI: 10.1145/322217.322228.

[13] Michael Drmota. *Random Trees: An Interplay Between Combinatorics and Probability*. 1st. Springer Vienna, 2009. DOI: 10.1007/978-3-211-75357-6.

[14] Michael Drmota and Bernhard Gittenberger. “The Shape of Unlabeled Rooted Random Trees”. In: *European Journal of Combinatorics* 31.8 (2010), pp. 2028–2063. DOI: 10.1016/j.ejc.2010.06.018.

[15] A. P. Ershov. “On Programming of Arithmetic Operations”. In: *Communications of the ACM* 1.8 (1958), pp. 3–9.

[16] Arash Farzan and J. Munro. “A Uniform Paradigm to Succinctly Encode Various Families of Trees”. In: *Algorithmica* 68 (Jan. 2014). DOI: 10.1007/s00453-012-9664-0.

[17] James Allen Fill and Nevin Kapur. “Transfer Theorems and Asymptotic Distributional Results for m-ary Search Trees”. In: *Random Structures & Algorithms* 26.4 (July 2005), pp. 359–391. DOI: 10.1002/rsa.v26:4.

[18] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[19] Philippe Flajolet, Paola Sipala, and Jean-Marc Steyaert. “Analytic Variations on the Common Subexpression Problem”. In: *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 220–234.

[20] Tomáš Flouri et al. “An Optimal Algorithm for Computing All Subtree Repeats in Trees”. In: *Combinatorial Algorithms, 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, Revised Selected Papers*. Vol. 8288. Lecture Notes in Computer Science. Heidelberg: Springer, 2013, pp. 269–282. DOI: 10.1007/978-3-319-03898-8_24.

[21] Moses Ganardi et al. “Universal Tree Source Coding Using Grammar-Based Compression”. In: *IEEE Transactions on Information Theory* 65.10 (2019), pp. 6399–6413. DOI: 10.1109/TIT.2019.2919829.

[22] Zbigniew Gołębiewski, Andrzej Magner, and Wojciech Szpankowski. "Entropy and Optimal Compression of Some General Plane Trees". In: *ACM Transactions on Algorithms (TALG)* 15.1 (2018), pp. 1–23. DOI: 10.1145/3282485.

[23] Lorenz Hübschle-Schneider and Rajeev Raman. "Tree Compression with Top Trees Revisited". In: *International Symposium on Experimental Algorithms*. Springer. 2015, pp. 15–27.

[24] Guy Joseph Jacobson. *Succinct static data structures*. Carnegie Mellon University, 1988.

[25] A. Jeż and M. Lohrey. "Approximation of Smallest Linear Tree Grammars". In: *Proceedings of STACS 2014*. Vol. 25. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 445–457. DOI: 10.4230/LIPIcs.STACS.2014.445.

[26] J.C. Kieffer. "A unified approach to weak universal source coding". In: *Information Theory, IEEE Transactions on* 24 (Dec. 1978), pp. 674–682. DOI: 10.1109/TIT.1978.1055960.

[27] Markus Lohrey. "Grammar-Based Tree Compression". In: *Proceedings of the International Conference on Developments in Language Theory*. Vol. 9168. 2015, pp. 46–57. DOI: 10.1007/978-3-319-21500-6_5.

[28] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. "XML tree structure compression using RePair". In: *Information Systems* 38 (Nov. 2013), pp. 1150–1167. DOI: 10.1016/j.is.2013.06.006.

[29] H. M. Mahmoud. *Evolution of Random Search Trees*. A Wiley-Interscience Publication. New York: John Wiley & Sons Inc., 1992.

[30] A. Meir and J. W. Moon. "On the Altitude of Nodes in Random Trees". In: *Canadian Journal of Mathematics* 30.5 (1978), pp. 997–1015. DOI: 10.4153/CJM-1978-073-8.

[31] J. Ian Munro et al. "Hypersuccinct Trees – New universal tree source codes for optimal compressed tree data structures and range minima". In: *29th Annual European Symposium on Algorithms (ESA 2021)*. Vol. 204. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 70:1–70:18. DOI: 10.4230/LIPIcs.ESA.2021.70. URL: https://arxiv.org/abs/2104.13457.

[32] Dimbinaina Ralaivaosaona and Stephan Wagner. "Repeated Fringe Subtrees in Random Rooted Trees". In: *Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2015)*. SIAM, 2015, pp. 78–88. DOI: 10.1137/1.9781611973761.

[33] Rajeev Raman and S. Srinivasa Rao. “Succinct Representations of Ordinal Trees”. In: *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*. Ed. by Andrej Brodnik et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 319–332. ISBN: 978-3-642-40273-9. DOI: 10.1007/978-3-642-40273-9_20.

[34] Jie Zhang, En-hui Yang, and John C. Kieffer. “A Universal Grammar-Based Code for Lossless Compression of Binary Trees”. In: *IEEE Transactions on Information Theory* 60.3 (2014), pp. 1373–1386. DOI: 10.1109/TIT.2013.2295392.

[35] D. G. Zill and M. R. Cullen. *Differential Equations with Boundary-Value Problems*. 5th. Belmont: Brooks Cole, 2001.