Entropy Bounds for Grammar-Based Tree Compressors

Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner
Universität Siegen
Germany

***Abstract*—The definition of k^{th}-order empirical entropy of strings is extended to node-labeled binary trees. A suitable binary encoding of tree straight-line programs (that have been used for grammar-based tree compression before) is shown to yield binary tree encodings of size bounded by the k^{th}-order empirical entropy plus some lower order terms. This generalizes recent results for grammar-based string compression to grammar-based tree compression. A long version of this paper can be found in [11].**

Keywords. Grammar-based compression, binary trees, empirical entropy, lossless compression

I. Introduction

a) **Grammar-based string compression:** The idea of grammar-based compression is based on the fact that in many cases a word w can be succinctly represented by a context-free grammar that produces exactly w. Such a grammar is called a *straight-line program* (SLP) for w. In the best case, one gets an SLP of size $\mathcal{O}(\log n)$ for a word of length n, where the size of an SLP is the total length of all right-hand sides of the rules of the grammar. A grammar-based compressor is an algorithm that produces for a given word w an SLP $\mathcal{G}_w$ for w, where, of course, $\mathcal{G}_w$ should be smaller than w. Grammar-based compressors can be found at many places in the literature. Probably the best known example is the classical LZ78-compressor of Lempel and Ziv [18]. Indeed, it is straightforward to transform the LZ78-representation of a word w into an SLP for w. Other grammar-based compressors can be found in [1].

Recently, several upper bounds on the compression perfomance of grammar-based compressors in terms of higher order empirical entropy have been shown. For this, the choice of a concrete binary encoding $B(\mathcal{G})$ of an SLP $\mathcal{G}$ is crucial. Kieffer and Yang [13] came up with such a binary encoding B and proved that under certain assumptions on the grammar-based compressor $w \mapsto \mathcal{G}_w$, the combined compressor $w \mapsto B(\mathcal{G}_w)$ yields a universal code with respect to the family of finite-state information sources over finite alphabets. Concretely, it is needed that the size of the SLP $\mathcal{G}_w$ is bounded by $\mathcal{O}(|w|/\log_{\hat{\sigma}} |w|)$ where σ is the size of the underlying alphabet and $\hat{\sigma} = \max\{2, \sigma\}$. This upper bound is met by all grammar-based compressors that only produce so-called irreducible SLPs [13]. Every SLP can be efficiently transformed into an irreducible SLP without increasing the size of the SLP. In their recent paper [16], Navarro and Ochoa used the binary encoding $B(\mathcal{G}_w)$ in order to prove for every word w over an alphabet of size σ the upper bound $|B(\mathcal{G}_w)| \leq |w|H_k(w) + o(|w| \log \hat{\sigma})$ for every $k \in o(\log_{\hat{\sigma}} |w|)$. Here, $H_k(w)$ is the empirical k^{th}-order entropy of w, and the grammar-based compressor $w \mapsto \mathcal{G}_w$ must satisfy the upper bound $|\mathcal{G}_w| \leq \mathcal{O}(|w|/\log_{\hat{\sigma}} |w|)$. Similar but weaker bounds for different binary SLP-encodings were shown in [8], [15].

b) **Grammar-based tree compression:** Grammar-based compression has been generalized from strings to trees by means of linear context-free tree grammars generating exactly one tree. Such grammars are also known as tree straight-line programs, TSLPs for short, see [14] for a survey. TSLPs can be seen as a proper generalization of SLPs and DAGs (directed acyclic graphs, which are a widely used compact representation of trees). Whereas DAGs only have the ability to share repeated subtrees of a tree, TSLPs can also share repeated tree patterns with a hole (so-called contexts). In [5], the authors presented a linear time algorithm that computes for a given binary tree t of size n and with σ node labels a TSLP $\mathcal{G}_t$ of size $\mathcal{O}(n/\log_{\hat{\sigma}} n)$; an alternative algorithm with the same asymptotic size bound can be found in [6]. The reader should notice that the $\mathcal{O}(n/\log_{\hat{\sigma}} n)$-bound cannot be achieved by DAGs: the smallest DAG for an unlabeled binary tree of size n may still contain n edges.

c) **Entropy bounds for grammar-based tree compressors:** In this paper we first consider binary node-labeled trees: every node has a label from a finite set Σ of size σ and every non-leaf node has a left and a right child. For binary unlabeled trees the results of Kieffer and Yang on universal grammar-based compressors have been extended to trees in [10], [17]. Whereas the universal tree encoder from [17] is based on DAGs (and needs a certain assumption on the average DAG size with respect to the input distribution), the encoder from [10] uses TSLPs of size $\mathcal{O}(n/\log n)$. For this, a binary encoding of TSLPs similar to the one for SLPs from [13] is proposed. In this paper we extend the binary TSLP-encoding from [10] to node-labeled binary trees and prove an upper bound similar to the one from [16] for strings. To do this, we first have to come up with a reasonable higher order entropy for binary node-labeled trees (we just speak of binary trees in the following). Several notions of tree entropy can be found in the literature, but all are tailored towards unranked trees and do not yield nontrivial results for binary trees.

- The k^{th}-order label entropy from [3] is based on the empirical probability that a node v is labeled with a

certain symbol conditioned on the k first labels from the parent node of v to the root of the tree.
- The tree entropy from [12] is the 0^{th} order entropy of the node degrees.
- Recently, two combinations of the two previous entropy measures were proposed in [7]. The first combination is based on the empirical probability that a node v is labeled with a certain symbol conditioned on (i) the k first labels from the parent node of v to the root and (ii) the node degree of v. The second combination uses the empirical probability that a node v has a certain degree conditioned on (i) the k first labels from the parent node of v to the root and (ii) the node label of v.

Tree entropy [12] is not useful in the context of binary trees, since a binary tree with n leaves has $n-1$ nodes of degree 2, which shows that the tree entropy divided by the number of nodes $(2n-1)$ converges to 1 when n increases. On the other hand, the k^{th}-order label entropy is not useful for unlabeled trees. For the special case of unlabeled binary trees, also the combinations of [7] do not lead to useful entropy measures.

Our first contribution is the definition of a reasonable entropy measure for binary trees that can also be used for the unlabeled case. For this we define the k-history of a node v in a binary tree t by taking the last k edges on the unique path from the root to v. For each edge (v_1, v_2) traversed on this path we write down the node label of v_1 and a 0 (resp., 1) if v_2 is a left (resp., right) child of v_1. Thus, the k-history of a node is a word of length $2k$ that alternatingly consists of symbols from Σ and directions that are encoded by 0 or 1. For nodes at depth smaller than k we pad the history with 0's and a default node label $\square \in \Sigma$ in order to get length exactly k. [1] For each k-history z we then enumerate all nodes $v_1, \ldots, v_l$ of t whose k-history is z and form the string $w(t,z)$ over the alphabet $\Sigma \times \{0,2\}$ by writing down for each node v_i the pair consisting the label of v_i (a symbol from Σ) and the degree (either 0 or 2) of v_i. We define the k^{th}-order empirical entropy of t, $H_k(t)$ for short, as the sum of the (0^{th}-order) unnormalized empirical entropies of these strings $w(t,z)$. This definition is similar to the definition of the k^{th} oder empirical entropy of a string. Our main result states that

$$|B(\mathcal{G}_t)| \leq H_k(t) + O\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + O\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma,$$

where t is a binary tree with n leaves, the grammar-based compressor $t \mapsto \mathcal{G}_t$ produces TSLPs of size $\mathcal{O}(n/\log n)$ for binary trees of size n, and B is the binary TSLP-encoding from [10]. If $k \leq o(\log_\sigma n)$ then this bound can be simplified to $|B(\mathcal{G}_t)| \leq H_k(t) + o(n \log \hat{\sigma})$. The assumption $k \leq o(\log_{\hat{\sigma}} n)$ can be also found in [16]. In fact, Gagie argued in [4] that k^{th}-order empirical entropy for strings stops being a reasonable complexity measure for almost all strings of length n over alphabets of size σ when $k \geq \log_{\hat{\sigma}} n$.

The k^{th}-order empirical entropy $H_k(t)$ is a lower bound on the coding length of a tree encoder that encodes for each node the relevant information (the label of the node and the binary information whether the node is a leaf or internal) depending on the k-history of the node. In the long version [11] of this paper we establish a relationship between $H_k(t)$ and a certain class of k^{th}-order tree processes: $H_k(t)$ coincides with the minimal self-information of t with respect to all k^{th}-order tree processes. This result is an extension of a result of Gagie [4] according to which the k^{th}-order empirical entropy of a string w is the minimal self-information of w with respect to all k^{th}-order Markov processes.

In the final section of the paper we present a simple extension of our entropy notion to node-labeled unranked trees. In an unranked tree the number of children of a node is arbitrary. Unranked trees are important in the area of XML, where the hierarchical structure of a document is represented by a node-labeled unranked tree. For such a tree t we define the k^{th}-order empirical entropy as the k^{th}-order empirical entropy of the *first-child next-sibling* (fcns for short) encoding of t. The fcns-encoding of t is a binary tree which contains all nodes of t. If a node v of t has the first (i.e., left-most) child v_1 and the right sibling v_2 then v_1 (resp., v_2) is the left (resp., right) child of v in the fcns-encoding of t. If v has no child or no right sibling then one adds dummy leaves to the fcns encoding in order to obtain a full binary tree. Our choice of defining the k^{th}-order empirical entropy of an unranked tree via the fcns-encoding is motivated by the fact that in XML document trees the label of a node v usually depends on the labels of the ancestors and the labels of the left siblings of v. This information is contained in the history of v in the fcns-encoding. In the long version [11] we also present experimental results for real XML document trees. For $k = 1, 2, 4, 8$ we computed the quotient of the k^{th}-order empirical entropy and the worst case bit length (which is $(2 + \log \sigma)n$, where n is the number of nodes and σ is the number of node labels [9]). For $k = 4$ the average quotient is about 5.3% (the maximal value is about 19%).

II. Preliminaries

We use the standard $\mathcal{O}$-notation. If $b > 0$ is a constant, then we write $\mathcal{O}(\log n)$ for $\mathcal{O}(\log_b n)$. We make the convention that $0 \cdot \log(0) = 0$.

Let $w = a_1 a_2 \cdots a_l \in \Gamma^*$ be a word over a finite alphabet Γ, i.e., $a_1, a_2, \ldots, a_l \in \Gamma$. With $|w| = l$ we denote the length of w. The empty word is denoted by ε. For $a \in \Gamma$ we denote with $|w|_a = |\{i \mid 1 \leq i \leq l, a_i = a\}|$ the number of occurrences of a in w. The *unnormalized empirical entropy* of w is

$$H(w) = -\sum_{a \in \Gamma} |w|_a \log_2 \left(\frac{|w|_a}{|w|}\right). \tag{1}$$

It is a lower bound for the bit length of $E(w)$, where $E : \Gamma \to \{0,1\}^*$ is an encoding function with a prefix-free image in the sense that there do not exist $w, w' \in \Gamma^*$ with $w \neq w'$ such that $E(w)$ is a prefix of $E(w')$. Note that $H(\varepsilon) = 0$.

[1] This is an ad hoc decision to simplify definitions. Alternatively, one could allow histories of length shorter than k; this would not change our results.

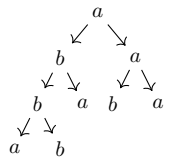

Fig. 1. A binary tree.

III. Binary trees and their entropy

A. Binary trees

Let Σ denote a finite non-empty alphabet of size $|\Sigma| = \sigma$. Later, we will need a fixed distinguished symbol from Σ that we will denote with $\square \in \Sigma$. We will also need the value $\hat{\sigma} = \max\{2, \sigma\}$. With $\mathcal{T}(\Sigma)$ we denote the set of *labeled binary trees* over the alphabet Σ. Formally, it is inductively defined as the smallest set of terms over Σ such that (i) $\Sigma \subseteq \mathcal{T}(\Sigma)$ and (ii) if $t_1, t_2 \in \mathcal{T}(\Sigma)$ and $a \in \Sigma$, then $a(t_1, t_2) \in \mathcal{T}(\Sigma)$. With $|t|$ we denote the number of leaves of t, which can be inductively defined by $|a| = 1$ and $|a(t_1, t_2)| = |t_1| + |t_2|$ for $a \in \Sigma$ and $t_1, t_2 \in \mathcal{T}(\Sigma)$. Note that $2|t| - 1$ is the number of occurrences of symbols from Σ in t. Let $\mathcal{T}_n(\Sigma) = \{t \in \mathcal{T}(\Sigma) \mid |t| = n\}$ for $n \geq 1$. Since Σ will not change in this paper, we use the abbreviations $\mathcal{T}$ and $\mathcal{T}_n$ for $\mathcal{T}(\Sigma)$ and $\mathcal{T}_n(\Sigma)$, respectively. A *tree encoder* is an injective mapping $E : \mathcal{T} \to \{0,1\}^*$ such that the range $E(\mathcal{T})$ is a prefix-free set of bit strings.

Occasionally, we will consider a binary tree as a graph with nodes and edges in the usual way, where each node is labeled with a symbol from Σ. Note that $t \in \mathcal{T}_n$ has $2n-1$ nodes in total: n leaves and $n-1$ internal nodes. It is convenient to define a node v of $t \in \mathcal{T}$ as a bit string that describes the path from the root to the node (0 means left, 1 means right). Formally, we define the node set $V(t) \subseteq \{0,1\}^*$ of $t \in \mathcal{T}$ by

- $V(a) = \{\varepsilon\}$ for every $a \in \Sigma$,
- $V(a(t_0, t_1)) = \{iw \mid i \in \{0,1\}, w \in V(t_i)\} \cup \{\varepsilon\}$ for every $a \in \Sigma$.

The leaves of t are those strings in $V(t)$ that are maximal with respect to the prefix relation.

Example 1: Consider the tree $t = a(b(b(a,b),a),a(b,a))$ with $\Sigma = \{a,b\}$ depicted in Figure 1. We have $V(t) = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001\}$.

Consider a tree $t \in \mathcal{T}$. Let $\lambda_t : V(t) \to \Sigma \times \{0,2\}$ denote the function mapping a node $v \in V(t)$ to a pair (a,i) where $a \in \Sigma$ is the label of v and $i \in \{0,2\}$ is the number of children of v. We can define this function inductively as follows:

- $\lambda_a(\varepsilon) = (a,0)$,
- $\lambda_t(\varepsilon) = (a,2)$ for $t = a(t_0,t_1)$,
- $\lambda_t(iw) = \lambda_{t_i}(w)$ for $t = a(t_0,t_1)$ and $iw \in V(t)$.

B. Histories

We define the set of *histories* as the language $\mathcal{L} = (\Sigma\{0,1\})^*$ consisting of all strings of the form $a_1 i_1 \cdots a_n i_n$, where $n \geq 0$, $a_k \in \Sigma$, and $i_k \in \{0,1\}$ for $1 \leq k \leq n$. For an integer $k \geq 1$ let $\mathcal{L}_k = \{w \in \mathcal{L} \mid |w| = 2k\}$ and let $\ell_k : \mathcal{L} \to \mathcal{L}_k$ be the partial function mapping a history $z \in \mathcal{L}$ with $|z| \geq 2k$ to the suffix of z of length $2k$, i.e., $\ell_k(a_1 i_1 \cdots a_n i_n) = a_{n-k+1} i_{n-k+1} \cdots a_n i_n$.

For a tree t and a node $v \in V(t)$, we inductively define its *history* $h(v) \in \mathcal{L}$ by (i) $h(\varepsilon) = \varepsilon$ and (ii) $h(wi) = h(w)ai$ for $i \in \{0,1\}$ and $wi \in V(t)$. Here, a is the symbol that labels the node w, i.e., $\lambda_t(w) = (a,2)$. That is, in order to obtain $h(v)$, while walking downwards in the tree from the root node to the node v we alternately concatenate symbols from Σ with binary numbers in $\{0,1\}$ such that the symbol from Σ corresponds to the label of the current node and the binary number 0 (resp., 1) states that we move on to the left (resp. right) child node. Note that the symbol that labels v is not part of the history of v. The k-history of a tree node $v \in V(t)$ is $h_k(v) = \ell_k((\square 0)^k h(v)) \in \mathcal{L}_k$, i.e. the suffix of length $2k$ of the word $(\square 0)^k h(v)$, where $\square$ is a fixed dummy symbol in Σ (the choice is arbitrary). This means that if $|v| \geq k$ then $h_k(v)$ describes the last k directions and node labels along the path from the root to node v. If $|v| < k$, we pad the history of v with $\square$'s and zeros such that $h_k(v) \in \mathcal{L}_k$. For $z \in \mathcal{L}_k$ we denote with $V_z(t) = \{v \in V(t) \mid h_k(v) = z\}$ the set of nodes in t with k-history z.

C. Higher-order entropy of a tree

We define the *k^{th}-order (unnormalized) empirical entropy* $H_k(t)$ of a tree $t \in \mathcal{T}_n$ as follows: Take a k-history $z \in \mathcal{L}_k$ and let $V_z(t) = \{v_1, v_2, \ldots, v_j\}$. Define the string $w(t,z) = \lambda_t(v_1)\lambda_t(v_2)\cdots\lambda_t(v_j) \in (\Sigma \times \{0,2\})^*$ (the order in which we enumerate the nodes in $V_z(t)$ has no influence on the following definition). We then define $H_k(t)$ as

$$H_k(t) = \sum_{z \in \mathcal{L}_k} H(w(t,z)),$$

where the empirical entropy $H(w(t,z))$ is defined according to (1). Since $\Sigma \times \{0,2\}$ consists of 2σ many symbols and the lengths of the strings $w(t,z)$ sum up to $2n-1$ (the number of nodes of t), one gets $0 \leq H_k(t) \leq (2n-1)(1 + \log_2 \sigma)$. This upper bound on the entropy matches the information theoretic bound for the worst-case output length of any tree encoder on $\mathcal{T}_n$. Using well-known asymptotic bounds for the number of binary trees with n leaves, one sees that for any tree encoder there must exist a tree $t \in \mathcal{T}_n$ which is encoded with $2\log_2(2\sigma)n - o(n) = 2(\log_2 \sigma + 1)n - o(n)$ bits. The k^{th}-order empirical entropy $H_k(t)$ is a lower bound on the coding length of a tree encoder that encodes for each node the relevant information (the label of the node and the binary information whether the node is a leaf or internal) depending on the k-history of the node.

IV. Tree straight-line programs and compression of binary trees

In this section we introduce tree straight-line programs and use them for the compression of binary trees.

A. Contexts

A *context* is a labeled binary tree c, where exactly one leaf is labeled with the special symbol $x \notin \Sigma$ (called the *parameter*);

all other nodes are labeled with symbols from Σ. Formally, the set of contexts $\mathcal{C}(\Sigma)$ is the smallest set of terms such that (i) $x \in \mathcal{C}(\Sigma)$ and (ii) if $a \in \Sigma$, $c \in \mathcal{C}(\Sigma)$ and $t \in \mathcal{T}(\Sigma)$ then also $a(c,t), a(t,c) \in \mathcal{C}(\Sigma)$. For a tree or context $s \in \mathcal{T}(\Sigma) \cup \mathcal{C}(\Sigma)$ and a context $c \in \mathcal{C}(\Sigma)$, we denote by $c[s]$ the tree or context which results from c by replacing the unique occurrence of the parameter x by s. For example $c = a(b,x)$ and $t = b(b,a)$ yield $c[t] = a(b,b(b,a))$. We write $\mathcal{C}$ for $\mathcal{C}(\Sigma)$.

B. Tree straight-line programs

Let V be a finite alphabet, where each symbol $A \in V$ has an associated rank 0 or 1. The elements of V are called *nonterminals*. We assume that V contains at least one element of rank 0 and that V is disjoint from the set $\Sigma \cup \{x\}$, which are the labels used for binary trees and contexts. We use V_0 (resp., V_1) for the set of nonterminals of rank 0 (resp., of rank 1). The idea is that nonterminals from V_0 (resp., V_1) derive to trees from $\mathcal{T}$ (resp., contexts from $\mathcal{C}$). We denote by $\mathcal{T}_V(\Sigma)$ the set of trees over $\Sigma \cup V$, i.e. each node in a tree $t \in \mathcal{T}_V(\Sigma)$ is labeled with a symbol from $\Sigma \cup V$ and the number of children of a node corresponds to the rank of its label. With $\mathcal{C}_V(\Sigma)$ we denote the corresponding set of all contexts, i.e., the set of trees over $\Sigma \cup \{x\} \cup V$, where the parameter symbol x occurs exactly once and at a leaf position. Formally, we define $\mathcal{T}_V(\Sigma)$ and $\mathcal{C}_V(\Sigma)$ as the smallest sets of terms with the following conditions, where here and in the rest of the paper we use the abbreviations $\mathcal{T}_V$ for $\mathcal{T}_V(\Sigma)$ and $\mathcal{C}_V$ for $\mathcal{C}_V(\Sigma)$:

- $\Sigma \cup V_0 \subseteq \mathcal{T}_V$ and $x \in \mathcal{C}_V$,
- if $a \in \Sigma$, $A \in V_1$, $t_1, t_2 \in \mathcal{T}_V$ then $A(t_1), a(t_1,t_2) \in \mathcal{T}_V$,
- if $a \in \Sigma$, $A \in V_1$, $s \in \mathcal{C}_V$ and $t \in \mathcal{T}_V$ then $A(s), a(s,t), a(t,s) \in \mathcal{C}_V$.

Note that $\mathcal{T} \subseteq \mathcal{T}_V$ and $\mathcal{C} \subseteq \mathcal{C}_V$. A *tree straight-line program* $\mathcal{G}$, or *TSLP* for short, is a tuple (V, A_0, r), where $A_0 \in V_0$ is the start nonterminal and $r : V \to (\mathcal{T}_V \cup \mathcal{C}_V)$ is a function which assigns to each nonterminal its unique right-hand side. It is required that if $A \in V_0$ (resp., $A \in V_1$), then $r(A) \in \mathcal{T}_V$ (resp., $r(A) \in \mathcal{C}_V$). Furthermore, the binary relation $\{(A,B) \in V \times V \mid B \text{ occurs in } r(A)\}$ has to be acyclic. These conditions ensure that exactly one tree is derived from the start nonterminal A_0 by using the rewrite rules $A \to r(A)$ for $A \in V$. To define this formally, we define $\mathsf{val}_{\mathcal{G}}(t) \in \mathcal{T}$ for $t \in \mathcal{T}_V$ and $\mathsf{val}_{\mathcal{G}}(t) \in \mathcal{C}$ for $t \in \mathcal{C}_V$ inductively by the following rules:

- $\mathsf{val}_{\mathcal{G}}(a) = a$ for $a \in \Sigma$ and $\mathsf{val}_{\mathcal{G}}(x) = x$,
- $\mathsf{val}_{\mathcal{G}}(a(t_1,t_2)) = a(\mathsf{val}_{\mathcal{G}}(t_1), \mathsf{val}_{\mathcal{G}}(t_2))$ for $a \in \Sigma$ and $t_1, t_2 \in \mathcal{T}_V \cup \mathcal{C}_V$ (and $t_1 \in \mathcal{T}_V$ or $t_2 \in \mathcal{T}_V$ since there is at most one parameter in $a(t_1,t_2)$),
- $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(r(A))$ for $A \in V_0$,
- $\mathsf{val}_{\mathcal{G}}(A(s)) = \mathsf{val}_{\mathcal{G}}(r(A))[\mathsf{val}_{\mathcal{G}}(s)]$ for $A \in V_1$, $s \in \mathcal{T}_V \cup \mathcal{C}_V$ (note that $\mathsf{val}_{\mathcal{G}}(r(A))$ is a context c, so we can build $c[\mathsf{val}_{\mathcal{G}}(s)]$).

The tree defined by $\mathcal{G}$ is $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(A_0) \in \mathcal{T}$.

Example 2: Let $\Sigma = \{a,b\}$ and $\mathcal{G} = (\{A_0, A_1, A_2\}, A_0, r)$ be a TSLP with $A_0, A_1 \in V_0, A_2 \in V_1$ and $r(A_0) = a(A_1, A_2(b))$, $r(A_1) = A_2(A_2(b))$, and $r(A_2) = b(x,a)$. We get $\mathsf{val}_{\mathcal{G}}(A_2) = b(x,a)$, $\mathsf{val}_{\mathcal{G}}(A_1) = b(b(b,a),a)$ and $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(A_0) = a(b(b(b,a),a),b(b,a))$.

C. Tree straight-line programs in normal form

A TSLP $\mathcal{G} = (V, A_0, r)$ is in *normal form* if the following conditions hold:

- $V = \{A_0, A_1, \ldots, A_{m-1}\}$ for some $m \geq 1$.
- For every $A_i \in V_0$, the right-hand side $r(A_i)$ is a term of the form $A_j(\alpha)$, where $A_j \in V_1$ and $\alpha \in V_0 \cup \Sigma$.
- For every $A_i \in V_1$ the right-hand side $r(A_i)$ is a term of the form $A_j(A_k(x))$, $a(\alpha,x)$, or $a(x,\alpha)$, where $A_j, A_k \in V_1$, $a \in \Sigma$ and $\alpha \in V_0 \cup \Sigma$.
- $\mathsf{val}_{\mathcal{G}}(A_i) \neq \mathsf{val}_{\mathcal{G}}(A_j)$ for $i \neq j$
- For every $A_i \in V$ define the word $\rho(A_i) \in (V \cup \Sigma)^*$ as

$$\rho(A_i) = \begin{cases} A_j\alpha & \text{if } r(A_i) = A_j(\alpha) \\ A_jA_k & \text{if } r(A_i) = A_j(A_k(x)) \\ a\alpha & \text{if } r(A_i) = a(\alpha,x) \text{ or } a(x,\alpha). \end{cases}$$

  Let $\rho_{\mathcal{G}} = \rho(A_0)\cdots\rho(A_{m-1}) \in (\Sigma \cup \{A_1, \ldots, A_{m-1}\})^*$. Then we require that the word $\rho_{\mathcal{G}}$ can be written as $\rho_{\mathcal{G}} = A_1u_1A_2u_2\cdots A_{m-1}u_{m-1}$ with $u_1, \ldots, u_{m-1} \in (\Sigma \cup \{A_1, A_2, \ldots, A_i\})^*$.

We also allow the TSLP $\mathcal{G}_a = (\{A_0\}, A_0, A_0 \mapsto a)$ for every $a \in \Sigma$ in order to get the singleton tree a. In this case, we set $\rho_{\mathcal{G}_a} = \rho(A_0) = a$.

Let $\mathcal{G} = (V, A_0, r)$ be a TSLP in normal form with $V = \{A_0, A_1, \ldots, A_{m-1}\}$ for the further definitions. We define the size of $\mathcal{G}$ as $|\mathcal{G}| = |V| = m$. Thus $2|\mathcal{G}|$ is the length of $\rho_{\mathcal{G}}$. Let $\omega_{\mathcal{G}}$ be the word obtained from $\rho_{\mathcal{G}}$ by removing for every $1 \leq i \leq m-1$ the first occurrence of A_i from $\rho_{\mathcal{G}}$. Thus, if $\rho_{\mathcal{G}} = A_1u_1A_2u_2\cdots A_{m-1}u_{m-1}$ with $u_i \in (\Sigma \cup \{A_1, A_2, \ldots, A_i\})^*$, then $\omega_{\mathcal{G}} = u_1u_2\cdots u_{m-1}$. Note that $|\omega_{\mathcal{G}}| = |\rho_{\mathcal{G}}| - m + 1 = m + 1$. The *entropy* $H(\mathcal{G})$ of the normal form TSLP $\mathcal{G}$ is defined as the empirical unnormalized entropy (see (1)) of the word $\omega_{\mathcal{G}}$: $H(\mathcal{G}) = H(\omega_{\mathcal{G}})$.

Example 3: Let $\Sigma = \{a,b\}$ and $\mathcal{G} = (V, A_0, r)$ be the normal form TSLP with $V_0 = \{A_0, A_2, A_3\}$, $V_1 = \{A_1, A_4\}$, $r(A_0) = A_1(A_2)$, $r(A_1) = a(x, A_3)$, $r(A_2) = A_4(A_3)$, $r(A_3) = A_4(b)$, and $r(A_4) = b(x,a)$. We have $\mathsf{val}(\mathcal{G}) = a(b(b(b,a),a),b(b,a))$, $\rho_{\mathcal{G}} = A_1A_2aA_3A_4A_3A_4bba$, $|\mathcal{G}| = 10$ and $\omega_{\mathcal{G}} = aA_3A_4bba$.

A *grammar-based tree compressor* is an algorithm ψ that produces for a given tree $t \in \mathcal{T}$ a TSLP $\mathcal{G}_t$ in normal form. It is not hard to show that every TSLP can be transformed with a linear size increase into a normal form TSLP that derives the same tree. For example, the TSLP from Example 2 is transformed into the normal form TSLP described in Example 3. We will not use this fact, since all we need is the following theorem from [5] (recall that $\hat{\sigma} = \max\{2, \sigma\}$):

Theorem 1: There exists a grammar-based compressor ψ (working in linear time) with $\max_{t \in \mathcal{T}_n} |\mathcal{G}_t| \leq \mathcal{O}(n/\log_{\hat{\sigma}} n)$.

D. Binary coding of TSLPs in normal form

In the long version of the paper, we define a binary encoding function E (which builds on similar encodings from [10], [13],

[17]) that maps a TSLP in normal form to a bit string such that the following two lemmas hold:

Lemma 1: The set of code words $B(\mathcal{G})$, where $\mathcal{G}$ ranges over all TSLPs in normal form, is a prefix code.

Lemma 2: For the length of the binary coding $B(\mathcal{G})$ we have: $|B(\mathcal{G})| \leq \mathcal{O}(|\mathcal{G}|) + \sigma + H(\mathcal{G})$.

V. Entropy bounds for binary encoded TSLPs

For this section we fix a grammar-based tree compressor $\psi : t \mapsto \mathcal{G}_t$ such that $\max_{t \in \mathcal{T}_n} |\mathcal{G}_t| \in \mathcal{O}(n/\log_{\hat{\sigma}} n)$. We allow that the alphabet size σ grows with n, i.e., $\sigma = \sigma(n)$ is a function in the tree size n such that $1 \leq \sigma(n) \leq 2n-1$ (a binary tree $t \in \mathcal{T}_n$ has $2n-1$ nodes). We consider the tree encoder $E_\psi : \mathcal{T} \to \{0,1\}^*$ defined by $E_\psi(t) = B(\mathcal{G}_t)$. The main result of the paper is:

Theorem 2: For every $t \in \mathcal{T}_n$ and every $k \geq 0$ we have

$$|E_\psi(t)| \leq H_k(t) + O\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + O\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right) + \sigma.$$

Let us give a brief outline of the proof of Theorem 2 (see [11] for details). With Lemma 2 we get

$$|E_\psi(t)| \leq \mathcal{O}(|\mathcal{G}_t|) + \sigma + H(\mathcal{G}_t) \leq O\left(\frac{n}{\log_{\hat{\sigma}} n}\right) + \sigma + H(\mathcal{G}_t).$$

The main part of the proof is to bound $H(\mathcal{G}_t) = H(\omega_{\mathcal{G}_t})$. We follow here the proof of [16] for strings, but the arguments get more involved. In [16], the authors use a lemma of Gagie [4] that roughly speaking states that the k^{th}-order empirical entropy of a string w is equal to $-\log_2 \mathsf{Prob}[P^w \text{ emits } w]$ (the self information of w with respect to P^w) where P^w is the empirical k^{th}-order Markov process of w (the probability that P^w emits a after $\alpha \in \Sigma^k$ is the probability that a follows an occurrence of α in w). We generalize Gagie's result to trees. For this we define in [11] a suitable class of k^{th}-order tree processes. Such a process determines for each k-history z the probability that a node with k-history z is labeled with a symbol $a \in \Sigma$ and has degree $i \in \{0,2\}$. One can then define for a tree t the empirical k^{th}-order tree process P^t analogously to the string case. Our generalization of Gagie's lemma states

$$H_k(t) = -\log_2 \mathsf{Prob}[P^t \text{ emits } t]. \tag{2}$$

The main technical result in [11] states that for every k^{th}-order tree process P, the entropy $H(\mathcal{G}_t)$ can be bounded by

$$-\log_2 \mathsf{Prob}[P \text{ emits } t] + O\left(\frac{kn \log \hat{\sigma}}{\log_{\hat{\sigma}} n}\right) + O\left(\frac{n \log\log_{\hat{\sigma}} n}{\log_{\hat{\sigma}} n}\right).$$

Using this bound for P^t together with Lemma 2 and (2) proves Theorem 2. To prove the upper bound of $H(\mathcal{G}_t)$ we define a factorization of t into subtrees and contexts that is similar to the parsing of the string in [16, Lemma 2] (see also [13]).

VI. Extension to unranked tree

So far, we have only considered binary trees. In many applications, Σ-labeled unranked ordered trees appear, where "ordered" means that the children of a node are totally ordered, and "unranked" means that the number of children of a node (also called its degree) can be any natural number. In order to define a k^{th}-order empirical entropy of such a tree t one can use the so-called first-child next-sibling encoding of t, $\mathrm{fcns}(t)$. This is a binary tree with node labels from $\Sigma \uplus \{\Box\}$ for a dummy symbol, which contains all nodes of t. If a node v of t has the first (i.e., left-most) child v_1 and the right sibling v_2 then v_1 (resp., v_2) is the left (resp., right) child of v in the fcns-encoding of t. If v has no child or no right sibling then one adds $\Box$-labeled leaves to the fcns encoding in order to obtain a full binary tree. Details and an example can be found in the long version [11]. We define the k^{th}-order empirical entropy of an unranked tree t as $H_k(t) = H_k(\mathrm{fcns}(t))$.

This above definition of the k^{th}-order empirical entropy of an unranked tree has a practical motivation. Unranked trees occur for instance in the context of XML, where the hierarchical structure of a document is represented as an unranked node-labeled tree. In this setting, the label of a node quite often depends on (i) the labels of the ancestor nodes and (ii) the labels of the (left) siblings. This dependence is captured by our definition of the k^{th}-order empirical entropy.

References

[1] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.

[2] T. M. Cover. Enumerative source encoding. *IEEE Trans. Inf. Theory*, 19(1):73–77, 1973.

[3] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009.

[4] T. Gagie. Large alphabets and incompressibility. *Inf. Process. Lett.*, 99(6):246–251, 2006.

[5] M. Ganardi, D. Hucke, A. Jez, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. *J. Comput. Syst. Sci.*, 86:136–158, 2017.

[6] M. Ganardi and M. Lohrey. A universal tree balancing theorem. *ACM Trans. Comput. Theory*, 11(1):1:1–1:25, 2018.

[7] M. Ganczorz. Entropy bounds for grammar compression. *CoRR*, abs/1804.08547, 2018.

[8] M. Ganczorz. Using statistical encoding to achieve tree succinctness never seen before. *CoRR*, abs/1807.06359, 2018.

[9] R. F. Geary, R. Raman, and V.Raman. Succinct ordinal trees with level-ancestor queries. *ACM Trans. Algorithms*, 2(4):510–534, 2006.

[10] D. Hucke and M. Lohrey. Universal tree source coding using grammar-based compression. In *Proceedings of ISIT 2017*, pages 1753–1757. IEEE Computer Society Press, 2017.

[11] D. Hucke, M. Lohrey and L. Seelbach Benkner. Entropy bounds for grammar-based tree compressors. *arXiv.org*, https://arxiv.org/abs/1901.03155, 2019.

[12] J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees with applications. *J. Comput. Syst. Sci.*, 78(2):619–631, 2012.

[13] J. C.Kieffer and E. h. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

[14] M. Lohrey. Grammar-based tree compression. In *Proceedings of DLT 2015*, LNCS 9168, pages 46–57. Springer, 2015.

[15] G. Navarro and L. Russo. Re-pair achieves high-order entropy. In *Proceedings of DCC 2008*, page 537. IEEE Computer Society, 2008.

[16] C. Ochoa and G. Navarro. RePair and all irreducible grammars are upper bounded by high-order empirical entropy. *IEEE Trans. Inf. Theory*, 2018. doi: 10.1109/TIT.2018.2871452.

[17] J. Zhang, E.-h. Yang, and J. C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Trans. Inf. Theory*, 60(3):1373–1386, 2014.

[18] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978.