XML Compression via Directed Acyclic Graphs

Noname manuscript No.
(will be inserted by the editor)

XML Compression via Directed Acyclic Graphs

Mireille Bousquet-Mélou · Markus Lohrey · Sebastian Maneth · Eric Noeth

Received: date / Accepted: date

Abstract Unranked node-labeled trees can be represented using their minimal dag (directed acyclic graph). For XML this achieves high compression ratios due to their repetitive mark up. Unranked trees are often represented through first child/next sibling (fcns) encoded binary trees. We study the difference in size (= number of edges) of minimal dag versus minimal dag of the fcns encoded binary tree. One main finding is that the size of the dag of the binary tree can never be smaller than the square root of the size of the minimal dag, and that there are examples that match this bound. We introduce a new combined structure, the *hybrid dag*, which is guaranteed to be smaller than (or equal in size to) both dags. Interestingly, we find through experiments that last child/previous sibling encodings are much better for XML compression via dags, than fcns encodings. We determine the average sizes of unranked and binary dags over a given set of labels (under uniform distribution) in terms of their exact generating functions, and in terms of their asymptotical behavior.

Keywords XML, tree compression, directed acyclic graph

M. Bousquet-Mélou
CNRS, LaBRI, Université de Bordeaux
Tel.: +33-5-40-00-69-06
E-mail: bousquet@labri.fr

M. Lohrey
University of Leipzig
Tel.: +49-341-97-32201
E-mail: lohrey@informatik.uni-leipzig.de

S. Maneth
University of Edinburgh
Tel.: +44-131-651-5642
E-mail: smaneth@inf.ed.ac.uk

E. Noeth
University of Leipzig
Tel.: +49-341-97-32212
E-mail: noeth@informatik.uni-leipzig.de

1 Introduction

The tree structure of an XML document can be conveniently represented as an ordered node-labeled unranked tree [30,26], where the children of a node are linearly ordered and every node is labeled with a symbol.[1] For tree structures of common XML documents *dags* (*directed acyclic graphs*) offer high compression ratios: the number of edges of the minimal dag is only about 10% of the number of edges of the original unranked tree [4] In a minimal dag, each distinct subtree is represented only once. A dag can be exponentially smaller than the represented tree. Dags and their linear average time construction via hashing are folklore in computer science (see e.g. [9]); they are a popular data structure used for sharing of common subexpressions (e.g., in programming languages) and in binary decision diagrams, see [25]. Through a clever pointer data structure, worst-case linear time construction is shown in [8].

Unranked trees of XML tree structures are often represented using binary trees, see [29] for a discussion. A common encoding is the *first child/next sibling encoding* [15] (in fact, this encoding is well-known, see Paragraph 2.3.2 in Knuth's first book [14]). The binary tree fcns(t) is obtained from an unranked tree t as follows. Each node of t is a node of fcns(t). A node u is a left child of node v in fcns(t) if and only if u is the first child of v in t. A node u is the right child of a node v in fcns(t) if and only if u is the next sibling of v in t. From now on, when we speak of the size of a graph we mean its number of edges. Consider the minimal dag of fcns(t) (called *bdag* for *binary dag* in the following) in comparison to the minimal dag of t. It was observed in [5] that the sizes of these dags may differ, in both directions. For some trees the difference is dramatic, which motivates the work of this paper: to study the precise relationship between the two dags, and to devise a new data structure that is guaranteed to be of equal or smaller size than the minimum size of the two dags.

Intuitively, the dag of t shares *repeated subtrees*, while the dag of fcns(t) shares *repeated sibling end sequences*. Consider the tree t_n in the left of Figure 1. Its minimal dag is shown on the right. As can be seen, each repeated

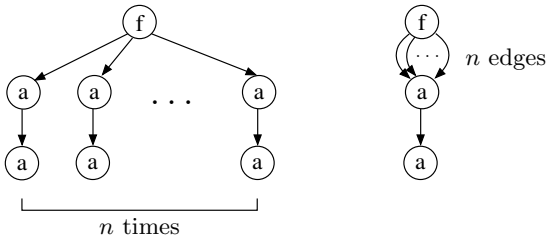

Fig. 1 The unranked tree t_n and dag(t_n).

[1] In the following, we use the term "unranked tree" as a synonym for "ordered node-labeled unranked tree".

subtree is removed in the dag. The dag consists of $n+1$ edges while t_n consists of $2n$ edges. Moreover, fcns(t_n) does not have any repeated subtrees (except for leaves), i.e., the bdag of t_n has $2n$ edges as well. Next, consider the tree s_n

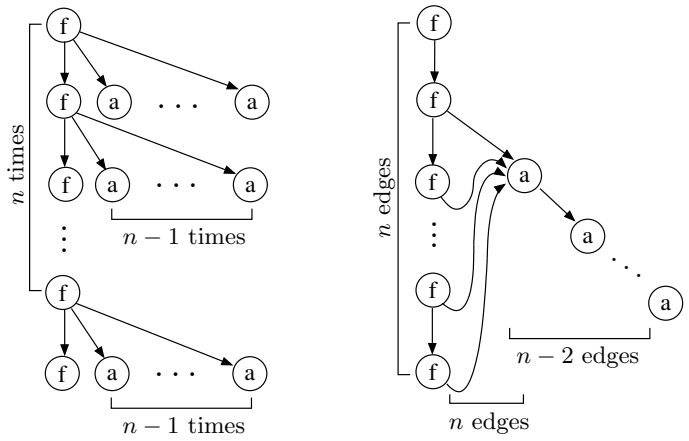

Fig. 2 The unranked tree s_n and bdag(s_n).

in the left of Figure 2. Its bdag is shown on the right, it has $3n-2$ edges. On the other hand, s_n has n^2 edges and the same is true for the dag of s_n since this tree has no repeated subtrees (except for leaves). These two examples show that (i) the size of the dag of an unranked tree can be half the size of the dag of the fcns encoded tree and (ii) the size of the dag of the fcns encoded tree can be quadratically smaller than the size of the dag of the unranked tree. We prove in this paper that these ratios are maximal: The size of the dag of the unranked tree is (i) lower bounded by half of the size of the bdag and (ii) upper bounded by the square of the size of the bdag. Actually, we derive these bounds from stronger statements concerning a combination of the unranked dag and the binary dag, called the *hybrid dag*, which combines both ways of sharing. The idea is as follows. Given an unranked tree, we compute its minimal dag. The dag can be naturally viewed as a regular tree grammar: Introduce for each node v of the dag a nonterminal A_v for the grammar. If a node v is labeled with the symbol f and its children in the dag are $v_1, \ldots, v_n$ in this order, then we introduce the production $A_v \to f(A_{v_1}, \ldots, A_{v_n})$. We now apply the fcns encoding to all right-hand sides of this grammar. Finally, we compute the minimal dag of the forest consisting of all these fcns encoded right-hand sides. See Figure 3 which shows a tree t of size 9. Its unranked and binary dags are each of size 6. The hybrid dag consists of a start tree plus one rule, and is of size 5. For the XML document trees of our corpus, the average size of the hybrid dag is only 76% of the average size of the unranked dag.

We show that the size of the hybrid dag is always bounded by the minimum of the sizes of the unranked dag and the binary dag. Moreover, we show that (i) the size of the hdag is at least half of the size of the binary dag and (ii)

the size of the unranked dag is at most the square of the size of the hdag. The above mentioned bounds for the unranked dag and binary dag are direct corollaries of these bounds.

The tree grammar of a hybrid dag is not a regular tree grammar anymore (because identifier nodes may have a right child). It can be unfolded in three passes: first undoing the sharing of tree sequences, then the binary decoding, and then undoing sharing of subtrees. We show that these grammars can be translated into a well known type of grammars: straight-line linear context-free tree grammars, for short *SLT grammars* (produced by BPLEX [5] or TreeRePair [20]). This embedding increases the size only slightly. One advantage is that SLT grammars can be unfolded into the original tree in one pass. Moreover, it is known that finite tree automata (even with sibling equality constraints) and tree walking automata can be executed in polynomial time over trees represented by SLT grammars [19, 22, 23].

While in the theoretical limit the binary dag can be smaller in comparison than the dag, it was observed in [5] that for common XML document trees t, almost always the dag of t is smaller than the binary dag of t. One explanation is that t contains many small repeated subtrees, which seldomly are part of a repeating sibling end sequence. For each repetition we (possibly) pay a "penalty" of one extra edge in the dag of fcns(t); see the tree t_n which has penalty n. On the other hand, there are very few repeating sibling end sequences in common XML; this is because optional elements typically appear towards *the end* of a child sequence. Hence, the additional feature of sharing sibling sequences is not useful for XML. On real XML documents, we show in experiments that the "reverse binary dag" that arises from the *last child/previous sibling encoding* is typically smaller than the binary dag, and almost as small as the dag. Moreover, for our test corpus, the average size of the *reverse hybrid dag* built from the last child/previous sibling encoding of the dag is only 62% of the average size of the minimal dag.

Observe that in the second sharing phase of the construction of the hybrid dag, only sequences of identifiers (nonterminals of the regular tree grammar corresponding to the dag) are shared. Thus, we are sharing repeated string suffixes in a sequence of strings. We experimented with applying a grammar-based string compressor to this sequence of strings. Such a compressor computes a small straight-line string grammar (SL grammar for short) for a given input word. The resulting object is called an *SL-grammar compressed dag*. It is not difficult to incorporate the output into an SLT grammar. As our experiments show, the obtained grammars are smaller than those of the hybrid dag and almost as small as TreeRePair's grammars. Moreover, they have the advantage that checking equivalence of subtrees is simple (each distinct subtree is represented by a unique identifier), a property not present for arbitrary SLT grammars. For hybrid dags, even equality of sibling end sequences can be checked efficiently.

Average size analysis of dags. Given a tree over n nodes and m labels, what is the average size of its minimal dag? This problem was studied for

unlabeled full binary trees by Flajolet, Sipala, and Steyaert [13]. They present exact expressions and show that the expected node size of the minimal dag of a full binary tree with n nodes is asymptotically

$$\kappa \cdot \frac{n}{\sqrt{\ln n}} \cdot \left(1 + O\left(\frac{1}{\ln n}\right)\right)$$

where the constant κ is explicitly determined. One problem with the paper [13] is that the proof of the result above is rather sketchy, and at certain places contains large gaps. Here we fill these gaps, and extend their results, giving detailed proofs of:

- exact expressions, in terms of their generating functions, for the average node and edge sizes of dags and bdags of unranked trees over n nodes and m labels, and of
- the asymptotic behavior of these averages. We show that these asymptotic behaviors are also of the form $C\frac{n}{\sqrt{\log n}}$, where C is again explicitly determined.

The proofs of these results assume basic knowledge about combinatorial classes, generating functions, and analytic combinatorics. Details on these can be found in textbooks, e.g., the one by Flajolet and Sedgewick [12].

Outline of this paper. Section 2 contains all necessary definitions concerning trees and dags, including the bdag of a tree. Section 3 introduces straight-line tree grammars (briefly, SLT grammars). The hybrid dag of a tree is defined in Section 4, whereas Section 5 deals with the reverse hybrid dag, based on the last child/previous sibling encoding of a tree. In Section 6 we compare the node size and the edge size of $\mathrm{dag}(t)$, $\mathrm{bdag}(t)$, and $\mathrm{hdag}(t)$ for an unranked tree t and prove the worst case bounds mentioned earlier. Next, we consider the average node and edge size of unranked trees in Section 7. Exact formulas are derived in Section 7.1, whereas asymptotic formulas are presented in Section 7.2. The quite technical proofs for the asymptotic formulas are shifted to an appendix at the end of the paper. Section 8 deals with SL-grammar compressed dags. In Section 9 we consider the problem of checking equality of subtrees and sibling sequences in the compressed tree representations from this paper (dags, bdags, hdags, and SL-grammar compressed dags). Finally, Section 10 presents our experimental results.

A preliminary version of this paper (not containing average-case sizes of dags) appeared as [21].

2 Trees and dags

Let Σ be a finite set of node labels. An *ordered Σ-labeled multigraph* is a tuple $M = (V, \gamma, \lambda)$, where

- V is a finite set of nodes
- $\gamma : V \to V^*$ assigns to each node a finite word over the set of nodes

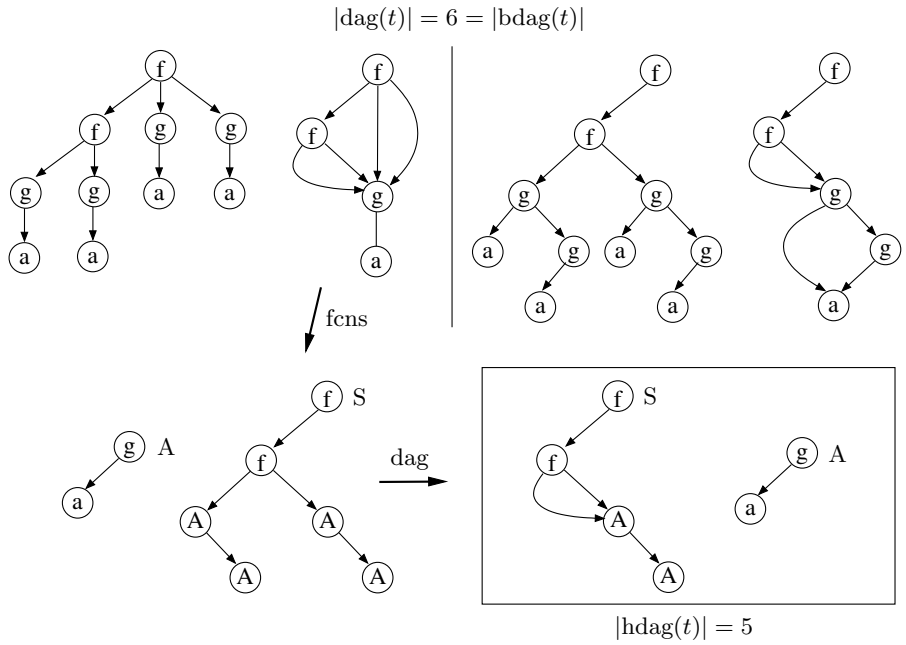

Fig. 3 Top: a tree t, its dag, its fcns encoding and its bdag of t. Bottom: its hybrid dag is shown in the box.

– $\lambda : V \to \Sigma$ assigns to each node a label from Σ.

The idea is that for a node $v \in V$, $\gamma(v)$ is the ordered list of v's successor nodes. The *underlying graph* is the directed graph $G_M = (V, E)$, where $(u, v) \in E$ if and only if v occurs in $\gamma(u)$. The *node size* $\|M\|$ of M and the *edge size* $|M|$ are defined as

$$\|M\| = |V| \quad \text{and} \quad |M| = \sum_{v \in V} |\gamma(v)| \tag{1}$$

(here, $|w|$ denotes the length of the word w). So, $\|M\|$ is the number of nodes of M. The edge size of M is also simply called the *size* of M. Note that the labeling function λ does not influence the size of M. The motivation for this is that the size of M can be seen as the number of pointers that are necessary in order to store M and that these pointers mainly determine the space consumption for M.

Two ordered Σ-labeled multigraphs $M_1 = (V_1, \gamma_1, \lambda_1)$ and $M_2 = (V_2, \gamma_2, \lambda_2)$ are isomorphic if there exists a bijection $f : V_1 \to V_2$ such that for all $v \in V_1$, $\gamma_2(f(v)) = f(\gamma_1(v))$ and $\lambda_2(f(v)) = \lambda_1(v)$ (here we implicitly extend f to a morphism $f : V_1^* \to V_2^*$). We do not distinguish between isomorphic multigraphs. In particular, in our figures a node $v \in V$ is not represented by the symbol v, but by the label $\lambda(v)$.

An *ordered Σ-labeled dag* is a Σ-labeled ordered multigraph $d = (V, \gamma, \lambda)$ such that the underlying graph G_d is acyclic. Note that the node size and edge size of the dag d are defined using the corresponding definitions for Σ-labeled

ordered multigraphs, see (1). The nodes $r \in V$ for which there is no $v \in V$ such that (v, r) is an edge of G_d (r has no incoming edges) are called the *roots* of d. An ordered Σ-labeled *rooted* dag is an ordered Σ-labeled dag with a unique root. In this case every node of d is reachable in G_d from the root node. The nodes $\ell \in V$ for which there is no $v \in V$ such that (ℓ, v) is an edge of G_d (ℓ has no outgoing edges) are called the *leaves* of d. An *ordered Σ-labeled tree* is an ordered Σ-labeled rooted dag $t = (V, \gamma, \lambda)$ such that every non-root node v has exactly one occurrence in the concatenation of all strings $\gamma(u)$ for $u \in V$. In other words, the underlying graph G_t is a rooted tree in the usual sense and in every string $\gamma(u)$, every $v \in V$ occurs at most once. We define $\mathcal{T}(\Sigma)$ as the set of all ordered Σ-labeled trees. We denote ordered Σ-labeled trees by their usual term notation, i.e., for every $a \in \Sigma$, $n \geq 0$, and all trees $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$, we also have $a(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma)$. Note that trees from $\mathcal{T}(\Sigma)$ are *unranked* in the sense that the number of children of a node does not depend on the label of the node. We therefore frequently speak of unranked trees for elements of $\mathcal{T}(\Sigma)$.

Let $d = (V, \gamma, \lambda)$ be an ordered Σ-labeled dag. With every node $v \in V$ we associate a tree $\mathrm{eval}_d(v) \in \mathcal{T}(\Sigma)$ inductively as follows: We set

$$\mathrm{eval}_d(v) = f(\mathrm{eval}_d(v_1), \ldots, \mathrm{eval}_d(v_n)),$$

if $\lambda(v) = f$ and $\gamma(v) = v_1 \cdots v_n$ (where $f(\varepsilon) = f$). Intuitively, $\mathrm{eval}_d(v)$ is the tree obtained by unfolding d starting in the node v. If d is an ordered Σ-labeled rooted dag, then we define $\mathrm{eval}(d) = \mathrm{eval}_d(r)$, where r is the root node of d. Note that if t is an ordered Σ-labeled tree and v is a node of t, then $\mathrm{eval}_t(v)$ is simply the subtree of t rooted at v and is written as $t/v = \mathrm{eval}_t(v)$ in this case. If for nodes $u \neq v$ of t we have $t/u = t/v$, then the tree $t/u = t/v$ is a *repeated subtree* of t.

Let $t = (V, \gamma, \lambda) \in \mathcal{T}(\Sigma)$ and let $G_t = (V, E)$ be the underlying graph (which is a tree). For an edge $(u, v) \in E$, v is a *child* of u, and u is the *parent* of v. If two nodes v and v' have the same parent node u, then v and v' are *siblings*. If moreover $\gamma(u)$ is of the form $u_1 v v' u_2$ for $u_1, u_2 \in V^*$ then v' is the *next sibling* of v, and v is the *previous sibling* of v'. If a node v does not have a previous sibling, it is a *first child*, and if it does not have a next sibling, it is a *last child*.

For many tree-processing formalisms (e.g. standard tree automata), it is useful to deal with ranked trees, where the number of children of a node is bounded. There is a standard binary encoding of unranked trees, which we introduce next. A *binary Σ-labeled dag* d, or short *binary dag*, can be defined as an ordered $(\Sigma \cup \{\Box\})$-labeled dag $d = (V, \gamma, \lambda)$, where $\Box \notin \Sigma$ is a special dummy symbol such that the following holds:

- For every $v \in V$ with $\lambda(v) \in \Sigma$ we have $|\gamma(v)| = 2$
- for every $v \in V$ with $\lambda(v) = \Box$ we have $|\gamma(v)| = 0$.

For a binary dag, $d = (V, \gamma, \lambda)$, we alter our definitions of node and edge sizes by disregarding all dummy nodes. That is, the node size is now $\|d\| = |\{v \in V \mid \lambda(v) \neq \Box\}|$ and the (edge) size is $|d| = \sum_{v \in V} |\gamma(v)|_\Sigma$, where

$|v_1 v_2 \cdots v_m|_\Sigma = |\{i \mid 1 \leq i \leq m, \lambda(v_i) \neq \Box\}|$. Accordingly, the dummy nodes are not represented in our figures.

A *binary Σ-labeled tree* t, or short *binary tree*, is a binary dag which is moreover an ordered $(\Sigma \cup \{\Box\})$-labeled tree. The symbol $\Box$ basically denotes the absence of a left or right child of a node. For instance, $g(a, \Box)$ denotes the binary tree that has a g-labeled root with an a-labeled left child but no right child (as shown in the bottom of Figure 3). Note that $g(a, \Box)$ and $g(\Box, a)$ are different binary trees.

Let $\mathcal{B}(\Sigma)$ denote the set of binary Σ-labeled trees. We define a mapping $\mathrm{fcns} : \mathcal{T}(\Sigma)^* \to \mathcal{B}(\Sigma)$, where as usual $\mathcal{T}(\Sigma)^*$ denotes the set of all finite words (or sequences) over the set $\mathcal{T}(\Sigma)$, as follows ("fcns" refers to "first child/next sibling"): For the empty word ε let $\mathrm{fcns}(\varepsilon) = \Box$ (the empty binary tree). If $n \geq 1$, $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$ and $t_1 = f(u_1, \ldots, u_m)$ with $m \geq 0$, then

$$\mathrm{fcns}(t_1 t_2 \cdots t_n) = f(\mathrm{fcns}(u_1 \cdots u_m), \mathrm{fcns}(t_2 \cdots t_n)).$$

Note that $\mathrm{fcns}(a) = a(\Box, \Box)$ for $a \in \Sigma$. In the following we always simply write a for $a(\Box, \Box)$. The encoding fcns is bijective, hence the inverse $\mathrm{fcns}^{-1} : \mathcal{B}(\Sigma) \to \mathcal{T}(\Sigma)^*$ is defined. Moreover, for every $t \in \mathcal{T}(\Sigma)$, we have $\|\mathrm{fcns}(t)\| = \|t\|$, see e.g. [14]. The fcns encoding is also known as rotation correspondence (see, e.g. [24]) and as Rabin encoding.

Example 1 Let $t_1 = f(a_1, a_2, a_3)$ and let $t_2 = g(b_1, b_2)$. Then $\mathrm{fcns}(t_1 t_2) = f(a_1(\Box, a_2(\Box, a_3)), g(b_1(\Box, b_2), \Box))$.

As mentioned in the Introduction, one can construct $\mathrm{fcns}(t)$ by keeping all nodes of t and creating edges as follows: For each node u of t, the left child of u in $\mathrm{fcns}(t)$ is the first child of u in t (if it exists) and the right child of u in $\mathrm{fcns}(t)$ is the next sibling of u in t (if it exists).

An ordered tree can be compacted by representing occurrences of repeated subtrees only once. Several edges then point to the same subtree (which we call a *repeated* subtree), thus making the tree an ordered dag. It is known that the minimal dag of a tree is unique and that it can be constructed in linear time (see e.g. [8]). For later purposes it is useful to define the minimal dag $\mathrm{dag}(d)$ for every ordered Σ-labeled dag $d = (V, \gamma, \lambda)$. It can be defined as

$$\mathrm{dag}(d) = (\{\mathrm{eval}_d(u) \mid u \in V\}, \gamma', \lambda')$$

with $\lambda'(f(t_1, \ldots, t_n)) = f$ and $\gamma'(f(t_1, \ldots, t_n)) = t_1 \cdots t_n$. Thus, the nodes of $\mathrm{dag}(d)$ are the different trees represented by the unfoldings of the nodes of d.

The internal structure of the nodes of $\mathrm{dag}(d)$ (which are trees in our definition) has no influence on the size of $\mathrm{dag}(d)$, which is still defined to be the number of its edges. Note that in general we cannot recover d from $\mathrm{dag}(d)$: For instance if d is the disjoint union of two copies of the same tree t, then $\mathrm{dag}(d) = \mathrm{dag}(t)$, but this will not be a problem. Indeed, we use dags only for the compression of forests consisting of different trees. Such a forest can be reconstructed from its minimal dag. Note also that if d is a rooted dag, then $\mathrm{dag}(d)$ is also rooted and we have $\mathrm{eval}(\mathrm{dag}(d)) = \mathrm{eval}(d)$.

Example 2 Consider the tree t_n defined by $t_0 = a$ and $t_n = a(t_{n-1}, t_{n-1})$. While $|t_n| = 2(2^n - 1)$, $|\mathrm{dag}(t_n)| = 2n$. Hence $\mathrm{dag}(t)$ can be exponentially smaller than t.

The *binary dag* of $t \in \mathcal{T}(\Sigma)$, denoted $\mathrm{bdag}(t)$, is defined as

$$\mathrm{bdag}(t) = \mathrm{dag}(\mathrm{fcns}(t)).$$

It is a binary dag as defined above. See Figure 2 in the Introduction for an example (recall that we do not represent dummy nodes in binary dags).

Clearly, the number of nodes of $\mathrm{dag}(t)$ equals the number of different subtrees t/v of t. In order to describe the number of nodes of $\mathrm{bdag}(t)$ the following definition is useful: For a node v of an unranked tree $t = (V, \gamma, \lambda)$ define $\mathrm{sibseq}(v) \in \mathcal{T}(\Sigma)^*$ (the *sibling sequence* of v) as the sequence of subtrees rooted at v and all its right siblings. More formally, if v is the root of t then $\mathrm{sibseq}(v) = t$. Otherwise, let u be the parent node of v and let $\gamma(u) = wvv_1 \cdots v_m$, where $w \in V^*$. Then

$$\mathrm{sibseq}(v) = (t/v)(t/v_1) \cdots (t/v_m).$$

Example 3 The different sibling sequences of the tree $t = f(a, f(b, a), b, a)$ are: t, $af(b, a)ba$, $f(b, a)ba$, ba, and a.

The following lemma follows directly from the definitions of bdag and sibseq:

Lemma 4 *The number of nodes of bdag(t) is equal to the number of different sibling sequences sibseq(v), for all $v \in V$.*

3 Straight-line tree grammars

Straight-line tree grammars are a formalism that in many cases give a more compact tree representation as dags.

Let $\{y_1, y_2, \ldots\}$ be an infinite fixed set of parameters (or variables). A *straight-line tree grammar* (*SLT grammar* for short) is a tuple $\mathcal{G} = (N, \mathrm{rank}, \Sigma, S, \rho)$, where

- N is a finite set of so-called *nonterminal symbols*
- $\mathrm{rank} : N \to \mathbb{N}$ maps every nonterminal to its rank (which may be 0)
- Σ is a finite set of node labels
- $S \in N$ is the start nonterminal and
- ρ is a function that maps every $X \in N$ to an ordered Γ-labeled tree $\rho(X) = (V, \gamma, \lambda)$, where $\Gamma = \Sigma \cup N \cup \{y_1, \ldots, y_{\mathrm{rank}(X)}\}$ and the following conditions hold:
  - for every $1 \leq i \leq \mathrm{rank}(X)$ there is exactly one node $v \in V$ with $\lambda(v) = y_i$, which moreover is a leaf of $\rho(X)$ and
  - for every node $v \in V$ with $\lambda(v) = Y \in N$ we have $|\gamma(v)| = \mathrm{rank}(Y)$, i.e., v has $\mathrm{rank}(Y)$ many children.

Finally, the binary relation $\{(X, Y) \in N \times N \mid Y \text{ appears in } \rho(X)\}$ must be acyclic.

We also write $X \to t$ for $\rho(X) = t$ and call $X \to t$ a *rule* or *production* of $\mathcal{G}$. Moreover, we also write $X(y_1, \ldots, y_{\mathrm{rank}(X)})$ instead of X in the left-hand side of the rules, to emphasize the rank of the nonterminals. The properties of an SLT grammar $\mathcal{G} = (N, \mathrm{rank}, \Sigma, S, \rho)$ allow us to define for every nonterminal $X \in N$ a $(\Sigma \cup \{y_1, \ldots, y_{\mathrm{rank}(X)}\})$-labeled tree $\mathrm{eval}_{\mathcal{G}}(X)$ inductively as follows[2]: Let $\rho(X) = (V, \gamma, \lambda)$. Assume that for every nonterminal Y that appears in $\rho(X)$, the tree $t_Y = \mathrm{eval}_{\mathcal{G}}(Y)$ is already defined. This is a tree that contains for every $1 \le j \le \mathrm{rank}(Y)$ exactly one leaf node that is labeled with y_j. We now replace every node $v \in V$ in $\rho(X)$ that is labeled with a nonterminal by a copy of the tree $t_{\lambda(v)}$. Thereby, the j-th child of v is identified with the y_j-labeled node of $t_{\lambda(v)}$ for every $1 \le j \le \mathrm{rank}(\lambda(v))$, and the parent node of the root of $t_{\lambda(v)}$ becomes the parent node of v. The resulting tree is $\mathrm{eval}_{\mathcal{G}}(X)$. For a completely formal definition, see e.g. [19,22].[3] Finally, let $\mathrm{eval}(\mathcal{G}) = \mathrm{eval}_{\mathcal{G}}(S)$. The term "straight-line tree grammar" comes from the fact that an SLT can be seen as a context-free tree grammar that produces a single tree.

The size of $\mathcal{G} = (N, \mathrm{rank}, \Sigma, S, \rho)$ is defined to be

$$|\mathcal{G}| = \sum_{X \in N} |\rho(X)|.$$

Example 5 Consider the SLT grammar $\mathcal{G}$ with nonterminals S, A, B and rules

$$\begin{aligned} S &\to B(a, b, B(c, d, a)), \\ B(y_1, y_2, y_3) &\to A(y_1, A(y_2, y_3)), \\ A(y_1, y_2) &\to f(g(y_1), y_2). \end{aligned}$$

We have $|\mathcal{G}| = 13$ and $\mathrm{eval}(\mathcal{G}) = f(g(a), f(g(b), f(g(c), f(g(d), a))))$. The same tree is also generated by the SLT grammar with the rules

$$\begin{aligned} S &\to A(a, A(b, A(c, A(d, a)))), \\ A(y_1, y_2) &\to f(g(y_1), y_2). \end{aligned}$$

Its size is only 11.

A k-SLT is an SLT $\mathcal{G} = (N, \mathrm{rank}, \Sigma, S, \rho)$ such that $\mathrm{rank}(X) \le k$ for every $X \in N$. A 0-SLT grammar is also called a *regular SLT* (since it is a regular tree grammar). In such a grammar, nonterminals only occur as leaves in the right-hand sides of the rules.

An ordered Σ-labeled rooted dag $d = (V, \gamma, \lambda)$ can be identified with the 0-SLT grammar

$$\mathcal{G}_d = (V, \mathrm{rank}, \Sigma, S, \rho),$$

[2] We hope that no confusion will arise with the evaluation of a dag defined in the previous section; we will see in fact that the evaluation of a dag can be seen as a special case of the evaluation of an SLT grammar.

[3] The formalisms in [19,22] slightly differ from our definition, since they assume a fixed rank for every node label in Σ. But this is not an essential difference.

where $\mathrm{rank}(v) = 0$ for every $v \in V$, S is the root of d, and $\rho(v) = f(v_1, \ldots, v_n)$ if $\lambda(v) = f$ and $\gamma(v) = v_1 \cdots v_n$. Note that all trees occurring in the right-hand sides of the rules have height 0 or 1. Often in this paper, it will be useful to eliminate in $\mathcal{G}_d$ nonterminals $v \in V$ such that $\gamma(v) = \varepsilon$ (that is, the leaves of the dag d). For this, we define the *reduced* 0-SLT grammar

$$\mathcal{G}_d^{\mathrm{red}} = (V \setminus V_0, \mathrm{rank}, \Sigma, S, \rho),$$

where $V_0 = \{v \in V \mid \gamma(v) = \varepsilon\}$, $\mathrm{rank}(v) = 0$ for every $v \in V \setminus V_0$, S is the root of d, and $\rho(v) = f(w_1, \ldots, w_n)$ if $\lambda(v) = f$, $\gamma(v) = v_1 \cdots v_n$, and $w_i = \lambda(v_i)$ if $v_i \in V_0$ and $w_i = v_i$ otherwise. Note that every right-hand side $\rho(v)$ of $\mathcal{G}_d^{\mathrm{red}}$ is now a tree of height 1. This does not change the evaluation of the grammar, and simplifies some technical details in Section 6. Of course, we should exclude the case that $V_0 = V$. For this, we simply exclude dags consisting of a single node from further considerations. Finally, observe that the evaluation of the grammar $\mathcal{G}_d$ coincides with the evaluation of the dag d defined in the previous section.

If $\mathcal{G}_d^{\mathrm{red}} = (V \setminus V_0, \mathrm{rank}, \Sigma, S, \rho)$ with $V \setminus V_0 = \{A_1, \ldots, A_n\}$ and $\rho(A_i) = f_i(\alpha_{i,1}, \ldots, \alpha_{i,k_i})$ (where $\alpha_{i,j} \in (V \setminus V_0) \cup \Sigma$) then the words $\alpha_{i,1} \cdots \alpha_{i,k_i} \in ((V \setminus V_0) \cap \Sigma)^+$ are called the *child sequences* of the dag d.

Example 6 For $d = \mathrm{dag}(t)$ in Figure 3, $\mathcal{G}_d^{\mathrm{red}}$ consists of the rules

$$S \to f(B, A, A), \qquad B \to f(A, A), \quad A \to g(a).$$

Hence the child sequences of the dag are BAA, AA, and a.

Algorithmic problems on SLT grammar-compressed trees were considered in [19,22]. Of particular interest in this context is a result from [22] stating that a given SLT $\mathcal{G}$ can be transformed in polynomial time into an equivalent 1-SLT $\mathcal{G}_1$ such that $\mathrm{eval}(\mathcal{G}_1) = \mathrm{eval}(\mathcal{G})$. In combination with results from [19] it follows that for a given nondeterministic tree automaton A (even with sibling constraints) and an SLT grammar $\mathcal{G}$ one can test in polynomial time whether A accepts $\mathrm{eval}(\mathcal{G})$. Compression algorithms that produce from a given input tree a small SLT grammar are proposed in [5,20].

4 The hybrid dag

While the dag shares repeated subtrees of a tree, the binary dag shares repeated sibling sequences (see Lemma 4). Consider an unranked tree t. As we have seen in the Introduction, the size of $\mathrm{dag}(t)$ can be smaller than the size of $\mathrm{bdag}(t)$. On the other hand, it can also be that the size of $\mathrm{bdag}(t)$ is smaller than the size of $\mathrm{dag}(t)$. We now wish to define a tree representation that combines both types of sharing (trees and tree sequences) and whose size is guaranteed to be smaller than or equal to the minimum of the sizes of $\mathrm{dag}(t)$ and $\mathrm{bdag}(t)$. Our starting point is $d = \mathrm{dag}(t)$. In this dag we now want to share all repeated sibling sequences. As an example, consider the tree

$t = f(f(g(a), g(a)), g(a), g(a))$ shown on the top left of Figure 3. Its size is 9. The dag of this tree consists of a unique occurrence of the subtree $g(a)$ plus two f-labeled nodes, shown to the right of t in the figure. Thus $|d| = 6$. The corresponding reduced 0-SLT grammar $\mathcal{G}_d^{\mathrm{red}}$ consists of the rules

$$\begin{aligned} S &\to f(B, A, A), \\ B &\to f(A, A), \\ A &\to g(a). \end{aligned} \tag{2}$$

In order to share repeated sibling sequences in d we apply the fcns encoding to the right-hand sides of $\mathcal{G}_d^{\mathrm{red}}$. For the above example we obtain the following new "binary tree grammar":

$$\begin{aligned} S &\to f(B(\Box, A(\Box, A)), \Box) \\ B &\to f(A(\Box, A), \Box) \\ A &\to g(a, \Box). \end{aligned} \tag{3}$$

This is not an SLT grammar, since there are occurrences of A with 0 and 2, respectively, children. We view the above rules just as the binary encoding of (2).

We now build the minimal dag of the forest obtained by taking the disjoint union of all right-hand sides of (3). In the example, the subtree $A(\Box, A)$ appears twice and is shared. We write the resulting dag again as a grammar, using the new nonterminal C for the new repeated tree $A(\Box, A)$ (corresponding to the repeated sibling sequence AA in (2)):

$$\begin{aligned} S &\to f(B(\Box, C), \Box) \\ B &\to f(C, \Box) \\ C &\to A(\Box, A) \\ A &\to g(a, \Box) \end{aligned} \tag{4}$$

These rules make up the *hybrid dag* (*hdag* for short) of the initial tree. Its size is the total number of edges in all right-hand side trees; it is 5 in our example (here, as usual, we do not count edges to $\Box$-labeled nodes). Compare this to $\mathrm{dag}(t)$ and $\mathrm{bdag}(t)$, which are both of size 6. Again, note that (4) should not be seen as an SLT-grammar but as a succinct encoding of (2).

In our example, the production $B \to f(A, A)$ in the 0-SLT grammar (2) does not save any edges, since the nonterminal B occurs only once in a right-hand side (namely $f(B, A, A)$). Eliminating this production yields the 0-SLT grammar

$$\begin{aligned} S &\to f(f(A, A), A, A) \\ A &\to g(a) \end{aligned}$$

with the fcns encoding

$$\begin{aligned} S &\to f(f(A(\Box, A), A(\Box, A)), \Box) \\ A &\to g(a, \Box). \end{aligned}$$

Sharing repeated subtrees gives

$$\begin{aligned} S &\to f(f(C,C),\Box) \\ C &\to A(\Box,A) \\ A &\to g(a,\Box), \end{aligned} \tag{5}$$

which corresponds to the framed graph in Figure 3 . The total number of edges to non-$\Box$ nodes in all right-hand sides is still 5, but it has only 3 nonterminals in contrast to 4 for the above hdag. In practice, having fewer nonterminals is preferable. In fact, our implementation avoids redundant nonterminals like B in our example. On the other hand, having only trees of height 1 as right-hand sides of the dag (seen as a reduced 0-SLT grammar) does not influence the number of edges in the final grammar. Moreover, it slightly simplifies the proofs in the next section, where we show that the size of the hdag of a tree t is smaller than or equal to the minimum of the sizes of $\mathrm{dag}(t)$ and $\mathrm{bdag}(t)$.

In general, the hybrid dag is produced by first building the minimal dag, then constructing the fcns encoding of the corresponding reduced 0-SLT grammar, and then building a minimal dag again. More formally, consider $d = \mathrm{dag}(t)$ and assume that the corresponding reduced 0-SLT grammar $\mathcal{G}_d^{\mathrm{red}}$ contains the rules $A_1 \to t_1, \ldots, A_n \to t_n$. Recall that every tree t_i has height 1 and that the trees $t_1, \ldots, t_n$ are pairwise different. Let t_i' be the tree that is obtained from t_i by adding A_i as an additional label to the root of t_i. Then

$$\mathrm{hdag}(t) = \mathrm{dag}(\mathrm{fcns}(t_1'), \ldots, \mathrm{fcns}(t_n')),$$

where the tuple $(\mathrm{fcns}(t_1'), \ldots, \mathrm{fcns}(t_n'))$ is viewed as the dag obtained by taking the disjoint union of the binary trees $\mathrm{fcns}(t_i')$. Clearly $\mathrm{hdag}(t)$ is unique up to isomorphism. In the second step when $\mathrm{dag}(\mathrm{fcns}(t_1'), \ldots, \mathrm{fcns}(t_n'))$ is constructed from the tuple $(\mathrm{fcns}(t_1'), \ldots, \mathrm{fcns}(t_n'))$, only suffixes of child sequences can be shared, since the trees $t_1', \ldots, t_n'$ are pairwise different and of height 1. The size $|\mathrm{hdag}(t)|$ of the hdag is the number of edges (to non-$\Box$-labeled nodes) of $\mathrm{dag}(\mathrm{fcns}(t_1'), \ldots, \mathrm{fcns}(t_n'))$. Note that the additional label A_i at the root of t_i is needed in order to be able to reconstruct the initial tree t. In (4), these additional labels (S, A, and B) are implicitly present as the left-hand sides of the rules. On the other hand, these labels have no influence on the size of the hdag.

The hdag is a particular dag. It is obtained by sharing repeated suffixes of child sequences in the minimal dag (viewed as a 0-SLT grammar). In Section 8 we introduce a further generalization of this idea, where child sequences of the dag are compressed using a straight-line context-free tree grammar. Moreover, we show that such a compressed structure can be easily transformed into an equivalent 1-SLT grammar (Theorem 32). This applies in particular to hdags. Hence, all the good algorithmic properties of (1-)SLT grammars (e.g. polynomial time evaluation of tree automata) also hold for hdags.

5 Using the reverse encoding

Instead of using the fcns encoding of a tree, one may also use the *last child previous sibling encoding* (lcps). Just like fcns, lcps is a bijection from $\mathcal{T}(\Sigma)^*$ to $\mathcal{B}(\Sigma)$ and is defined as follows. For the empty word ε let $\mathrm{lcps}(\varepsilon) = \Box$ (the empty binary tree). If $n \geq 1$, $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$ and $t_n = f(u_1, \ldots, u_m)$ with $m \geq 0$, then

$$\mathrm{lcps}(t_1 t_2 \cdots t_n) = f(\mathrm{lcps}(t_1, \ldots, t_{n-1}), \mathrm{lcps}(u_1 \cdots u_m)).$$

Again, the inverse $\mathrm{lcps}^{-1} : \mathcal{B}(\Sigma) \to \mathcal{T}(\Sigma)^*$ is defined.

Example 7 Let $t_1 = f(a_1, a_2, a_3)$ and $t_2 = g(b_1, b_2)$. Then

$$\mathrm{lcps}(t_1 t_2) = g(f(\Box, a_3(a_2(a_1, \Box), \Box)), b_2(b_1, \Box)).$$

Let $\mathrm{rbdag}(t) = \mathrm{dag}(\mathrm{lcps}(t))$ and

$$\mathrm{rhdag}(t) = \mathrm{dag}(\mathrm{lcps}(t'_1), \ldots, \mathrm{lcps}(t'_n)),$$

where $t'_1, \ldots, t'_n$ are obtained from t as in the definition of the hdag. The reason to consider the lcps encoding is that $\mathrm{rbdag}(t)$ and $\mathrm{rhdag}(t)$ are smaller for trees that have repeated *prefixes* of child sequences. Empirically, as we show in Section 10.3, this is quite common and for most trees t in our XML corpus $|\mathrm{rbdag}(t)| < |\mathrm{bdag}(t)|$ and $|\mathrm{rhdag}(t)| < |\mathrm{hdag}(t)|$.

Example 8 Let $t = f(f(a, a, b), f(a, a, c))$. Then $|\mathrm{rbdag}(t)| = 7$ while $|\mathrm{dag}(t)| = |\mathrm{bdag}(t)| = |\mathrm{hdag}(t)| = |t| = 8$.

Clearly, there are also trees t where $|\mathrm{hdag}(t)| < |\mathrm{rhdag}(t)|$. This raises the question whether there is a scheme which combines the best of both approaches. Obviously one can construct both $\mathrm{hdag}(t)$ and $\mathrm{rhdag}(t)$ of a tree t and discard the larger of both. Yet a scheme which combines both approaches by sharing both suffixes and prefixes of children sequences, faces the problem that the resulting minimal object is not necessarily unique. This can easily be seen by considering trees in which repeated prefixes and suffixes of child sequences overlap. Also it is not clear how a minimal such object can be constructed efficiently. A (non-optimal) approach we have considered was to first share repeated prefixes and then share repeated suffixes. Yet the results in compression achieved were not significantly better than for the rhdag. Moreover, this approach can be further generalized by sharing arbitrary factors of sibling sequences. This is the topic of Section 8.

6 Comparison of worst-case sizes of dag, bdag, and hdag

We want to compare the node size and the edge size of $\mathrm{dag}(t)$, $\mathrm{bdag}(t)$, and $\mathrm{hdag}(t)$ for an unranked tree t. We do not include $\mathrm{rbdag}(t)$ or $\mathrm{rhdag}(t)$, because by symmetry the same bounds holds as for $\mathrm{bdag}(t)$ and $\mathrm{hdag}(t)$, respectively.

6.1 The number of nodes

In this section we consider the number of *nodes* in the dag and bdag of an unranked tree t. We show that $\|\mathrm{dag}(t)\| \leq \|\mathrm{bdag}(t)\|$.

Example 9 Consider the tree $t_n = f(a, a, \ldots, a)$ consisting of n nodes, where $n \geq 2$. Then $\|\mathrm{dag}(t)\| = 2$ and $\|\mathrm{bdag}(t)\| = n$, while $|\mathrm{dag}(t)| = |\mathrm{bdag}(t)| = n - 1$. Note that dags with multiplicities on edges, as defined in [4], can store a tree such as t_n in size $O(\log n)$.

Lemma 10 *Let t be an unranked tree. Then $\|dag(t)\| \leq \|bdag(t)\|$.*

Proof The lemma follows from Lemma 4 and the obvious fact that the number of different subtrees of t (i.e., $\|\mathrm{dag}(t)\|$) is at most the number of different sibling sequences in t: $\mathrm{sibseq}(u) = \mathrm{sibseq}(v)$ implies $t/u = t/v$. □

Lemma 11 *There exists a family of trees $(t_n)_{n\geq 2}$ such that $\|dag(t)\| = 2$ and $\|t_n\| = \|bdag(t)\| = n$.*

Proof Take the family of trees t_n from Example 9. □

Let us remark that the node size of the hdag can be larger than the node size of the bdag and the node size of the dag. The reason is that in $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$, there is a nonterminal for each node of the dag (and hence the height of each right-hand side is at most one). This can be done differently of course; it was chosen to simplify proofs and because our main goal is the reduction of edge size. Note that the total number of edges of $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$ is equal to the number of edges of $\mathrm{dag}(t)$.

6.2 The number of edges

We have just seen that the number of nodes of the (minimal) dag is always at most the number of nodes of the bdag, and that the gap can be maximal ($O(1)$ versus $|t|$). For the number of edges, the situation is different. We show that $\frac{1}{2}|\mathrm{bdag}(t)| \leq |\mathrm{dag}(t)| \leq \frac{1}{2}|\mathrm{bdag}(t)|^2$ for $|t| \geq 2$ and that these bounds are sharp up to the constant factor $1/2$ in the second inequality. In fact, for $|t| \geq 2$ we show the three inequalities

$$\begin{aligned} |\mathrm{hdag}(t)| &\leq \min(|\mathrm{dag}(t)|, |\mathrm{bdag}(t)|), \\ |\mathrm{bdag}(t)| &\leq 2|\mathrm{hdag}(t)|, \text{ and} \\ |\mathrm{dag}(t)| &\leq \frac{1}{2}|\mathrm{hdag}(t)|^2 \end{aligned}$$

which imply

$$\frac{1}{2}|\mathrm{bdag}(t)| \leq |\mathrm{dag}(t)| \leq \frac{1}{2}|\mathrm{bdag}(t)|^2.$$

Before we prove these bounds we need some definitions. Recall that the nodes of $\mathrm{bdag}(t)$ are in 1-1-correspondence with the different sibling sequences of t. In the following, let

$$\mathrm{sib}(t) = \{\mathrm{sibseq}(v) \mid v \text{ a node of } t\}$$

be the set of all sibling sequences of t. To count the size (number of edges) of $\mathrm{bdag}(t)$ we have to count for each sibling sequence $w \in \mathrm{sib}(t)$ the number of outgoing edges in $\mathrm{bdag}(t)$. We denote this number with $e(w)$; it can be computed as follows, where $w = s_1 s_2 \cdots s_m$ ($m \geq 1$) and the s_i are trees:

- $e(w) = 0$, if $m = 1$ and $|s_1| = 0$
- $e(w) = 1$, if either $m = 1$ and $|s_1| \geq 1$ (then w has only a left child) or if $m \geq 2$ and $|s_1| = 0$ (then w has only a right child)
- $e(w) = 2$, otherwise.

With this definition we obtain:

Lemma 12 *For every $t \in \mathcal{T}(\Sigma)$, we have*

$$|bdag(t)| = \sum_{w \in sib(t)} e(w).$$

The size of the hdag can be computed similarly: Consider the reduced 0-SLT grammar $\mathcal{G} = \mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$. Let N be the set of nonterminals of $\mathcal{G}$ and let S be the start nonterminal. Recall that every right-hand side of $\mathcal{G}$ has the form $f(\alpha_1, \ldots, \alpha_n)$, where every α_i belongs to $\Sigma \cup N$. Let $\mathrm{sib}(\mathcal{G})$ be the set of all sibling sequences that occur in the right-hand sides of $\mathcal{G}$. Thus, for every right-hand side $f(\alpha_1, \ldots, \alpha_n)$ of $\mathcal{G}$, the sibling sequences $f(\alpha_1, \ldots, \alpha_n)$ (a sibling sequence of length 1) and $\alpha_i \alpha_{i+1} \cdots \alpha_n$ ($1 \leq i \leq n$) belong to $\mathrm{sib}(\mathcal{G})$. For such a sibling sequence w we define $e(w)$ as above. Here, every α_i is viewed as a tree with a single nodes, i.e., $|\alpha_i| = 0$. Then we have:

Lemma 13 *For every $t \in \mathcal{T}(\Sigma)$, we have*

$$|hdag(t)| = \sum_{w \in sib(\mathcal{G})} e(w).$$

For $w = s_1 \cdots s_m \in \mathrm{sib}(t)$ let $\tilde{w}$ be the string that results from w by replacing every non-singleton tree $s_i \notin \Sigma$ by the unique nonterminal of $\mathcal{G}$ that derives to s_i. Actually, we should write $\tilde{w}_t$ instead of $\tilde{w}$, since the latter also depends on the tree t. But the tree t will be always clear from the context. Here are a few simple statements:

- For every $w \in \mathrm{sib}(t)$, the sibling sequence $\tilde{w}$ belongs to $\mathrm{sib}(\mathcal{G})$, except for the length-1 sequence $\tilde{w} = S$ that is obtained from the length-1 sequence $w = t \in \mathrm{sib}(t)$.
- For every $w \in \mathrm{sib}(t)$, $\tilde{w}$ is a word over $N \cup \Sigma$.
- For every $w \in \mathrm{sib}(t)$, $e(\tilde{w}) \leq e(w)$.
- The mapping $w \mapsto \tilde{w}$ is an injective mapping from $\mathrm{sib}(t) \setminus \{t\}$ to $\mathrm{sib}(\mathcal{G})$.

Using this mapping, the sums in Lemma 12 and 13 can be related as follows:

Lemma 14 *For every $t \in \mathcal{T}(\Sigma)$, we have*

$$|hdag(t)| \;=\; \sum_{w \in sib(\mathcal{G})} e(w) \;=\; |N| + \sum_{w \in sib(t)} e(\tilde{w}).$$

Proof By Lemma 13 it remains to show the second equality. The only sibling sequences in $\mathrm{sib}(\mathcal{G})$ that are not of the form $\tilde{w}$ for $w \in \mathrm{sib}(t)$ are the sequences (of length 1) that consist of the whole right-hand side $f(\alpha_1, \ldots, \alpha_m)$ of a nonterminal $A \in N$. For such a sibling sequence u we have $e(u) = 1$ (since it has length 1 and $f(\alpha_1, \ldots, \alpha_m)$ is not a single symbol). Hence, we have

$$\begin{aligned}\sum_{w \in \mathrm{sib}(\mathcal{G})} e(w) &= |N| + \sum_{w \in \mathrm{sib}(t) \setminus \{t\}} e(\tilde{w}) \\ &= |N| + \sum_{w \in \mathrm{sib}(t)} e(\tilde{w}),\end{aligned}$$

where the last equality follows from $e(\tilde{t}) = e(S) = 0$. □

Theorem 15 *For every $t \in \mathcal{T}(\Sigma)$, we have*

$$|hdag(t)| \leq \min(|dag(t)|, |bdag(t)|).$$

Proof Since $\mathrm{hdag}(t)$ is obtained from $\mathrm{dag}(t)$ by sharing repeated suffixes of child sequences, we immediately get $|\mathrm{hdag}(t)| \leq |\mathrm{dag}(t)|$. It remains to show $|\mathrm{hdag}(t)| \leq |\mathrm{bdag}(t)|$. By Lemma 12 and 14 we have to show

$$|N| + \sum_{w \in \mathrm{sib}(t)} e(\tilde{w}) \;\leq\; \sum_{w \in \mathrm{sib}(t)} e(w),$$

where N is the set of nonterminals of $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$. To see this, note that:

- $e(\tilde{w}) \leq e(w)$ for all $w \in \mathrm{sib}(t)$ and
- for every nonterminal $A \in N$ there must exist a sibling sequence $w \in \mathrm{sib}(t)$ such that $\tilde{w}$ starts with A. For this sequence we have $e(w) = e(\tilde{w}) + 1$ (note that the right-hand side of A does not belong to Σ, and hence w starts with a tree of size at least 1).

Choose for every $A \in N$ a sibling sequence $w_A \in \mathrm{sib}(t)$ such that $\tilde{w}_A$ starts with A. Let $R = \mathrm{sib}(t) \setminus \{w_A \mid A \in N\}$. We get

$$\begin{aligned}|N| + \sum_{w \in \mathrm{sib}(t)} e(\tilde{w}) &= |N| + \sum_{A \in N} e(\tilde{w}_A) + \sum_{w \in R} e(\tilde{w}) \\ &= \sum_{A \in N} (e(\tilde{w}_A) + 1) + \sum_{w \in R} e(\tilde{w}) \\ &\leq \sum_{A \in N} e(w_A) + \sum_{w \in R} e(w) \\ &= \sum_{w \in \mathrm{sib}(t)} e(w).\end{aligned}$$

This proves the theorem. □

Theorem 16 *For every $t \in \mathcal{T}(\Sigma)$ with $|t| \geq 2$, we have*

$$|\mathit{dag}(t)| \leq \frac{1}{2}|\mathit{hdag}(t)|^2.$$

Proof Let $f_i(\alpha_{i,1}, \ldots, \alpha_{i,n_i})$ for $1 \leq i \leq k$ be the right-hand sides of $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$. W.l.o.g. assume that $1 \leq n_1 \leq n_2 \leq \cdots \leq n_k$. Every $\alpha_{i,j}$ is either from Σ or a nonterminal. Moreover, all the trees $f_i(\alpha_{i,1}, \ldots, \alpha_{i,n_i})$ are pairwise different. We have $|\mathrm{dag}(t)| = \sum_{i=1}^{k} n_i$.

If $n_k = 1$, then t is a linear chain. In this case, we get

$$|\mathrm{dag}(t)| = |t| \leq \frac{1}{2}|t|^2 = \frac{1}{2}|\mathrm{hdag}(t)|^2$$

since $|t| \geq 2$. Let us now assume that $n_k \geq 2$. Recall that we compute $\mathrm{hdag}(t)$ by taking the minimal dag of the forest consisting of the binary encodings of the trees $f_i(\alpha_{i,1}, \ldots, \alpha_{i,n_i})$. The binary encoding of $f_i(\alpha_{i,1}, \ldots, \alpha_{i,n_i})$ has the form $f_i(t_i, \Box)$, where t_i is a chain of $n_i - 1$ many right pointers. Let d be the minimal dag of the forest consisting of all chains t_i. Since all the trees $f_i(\alpha_{i,1}, \ldots, \alpha_{i,n_i})$ are pairwise distinct, we have $|\mathrm{hdag}(t)| = k + |d|$. Since the chain t_i consists of n_i many nodes, we have $|d| \geq \max\{n_i \mid 1 \leq i \leq k\} - 1 = n_k - 1$. Hence, we have to show that $\sum_{i=1}^{k} n_i \leq \frac{1}{2}(k + n_k - 1)^2$. We have

$$\sum_{i=1}^{k} n_i \leq k \cdot n_k \leq (k-1)n_k + \frac{1}{2}n_k^2 = \frac{1}{2}(2(k-1)n_k + n_k^2) \leq \frac{1}{2}(k - 1 + n_k)^2,$$

which concludes the proof. For the second inequality note that $n_k \leq \frac{1}{2}n_k^2$, since $n_k \geq 2$. □

Consider the tree s_n from Figure 2. We have $|\mathrm{dag}(s_n)| = |s_n| = n^2$ and $|\mathrm{hdag}(s_n)| = |\mathrm{bdag}(s_n)| = 3n - 2$. Hence, we get

$$|\mathrm{dag}(s_n)| = n^2 > \frac{1}{9}(3n-2)^2 = \frac{1}{9}|\mathrm{hdag}(s_n)|^2.$$

This shows that up to a constant factor, the bound in Theorem 16 is sharp. The constant $1/9$ can be slightly improved:

Theorem 17 *There is a family of trees $(s_n)_{n \geq 1}$ such that*

$$|\mathit{dag}(s_n)| > \frac{1}{6}|\mathit{hdag}(s_n)|^2.$$

Proof We specify s_n by the reduced 0-SLT grammar $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(s_n)}$. Let $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(s_n)}$ contain the following productions for $0 \leq i \leq n$:

$$A_i \to f(A_{i+1}, \ldots, A_n, \underbrace{a, \ldots, a}_{n \text{ many}}).$$

This is indeed the grammar obtained from the minimal dag for a tree s_n (of size exponential in n). We have

$$|\mathrm{dag}(s_n)| = \sum_{i=n}^{2n} i = n(n+1) + \sum_{i=0}^{n} i = n(n+1) + \frac{n(n+1)}{2} = \frac{3n(n+1)}{2}.$$

The hybrid dag of s_n consists of the child sequence $A_1 A_2 \cdots A_n a^n$ together with $n+1$ many left pointers into this sequence. Hence, we have

$$|\mathrm{hdag}(s_n)| = 2n - 1 + n + 1 = 3n.$$

We obtain

$$\frac{1}{6}|\mathrm{hdag}(s_n)|^2 = \frac{1}{6}9n^2 = \frac{3}{2}n^2 < \frac{3n(n+1)}{2} = |\mathrm{dag}(s_n)|.$$

This proves the theorem. □

Next let us bound $|\mathrm{bdag}(t)|$ in terms of $|\mathrm{hdag}(t)|$:

Theorem 18 *For every $t \in \mathcal{T}(\Sigma)$, we have*

$$|bdag(t)| + n \leq 2|hdag(t)|,$$

where n is the number of non-leaf nodes of $dag(t)$.

Proof We use the notations introduced before Theorem 15. Note that $n = |N|$ is the number of nonterminals of the 0-SLT grammar $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$. By Lemma 12 we have $|\mathrm{bdag}(t)| = \sum_{w \in \mathrm{sib}(t)} e(w)$. By Lemma 14 we have $|\mathrm{hdag}(t)| = |N| + \sum_{w \in \mathrm{sib}(t)} e(\tilde{w})$. Hence, we have to show that

$$|N| + \sum_{w \in \mathrm{sib}(t)} e(\tilde{w}) \;\geq\; \frac{1}{2} \sum_{w \in \mathrm{sib}(t)} e(w) \;+\; \frac{1}{2}|N|.$$

In order to prove this, we show the following for every sibling sequence $w \in \mathrm{sib}(t)$: Either $e(\tilde{w}) \geq \frac{1}{2}e(w)$ or $e(\tilde{w}) = 0$ and $e(w) = 1$. In the latter case, the sibling sequence w consists of a single tree s of size at least one (i.e., s does not consist of a single node), and $\tilde{w}$ consists of a single nonterminal $A \in N$. So, let $w = t_1 \cdots t_n \in \mathrm{sib}(t)$ and let $\tilde{w} = \alpha_1 \cdots \alpha_n$ with $\alpha_i \in \Sigma \cup N$. We consider the following four cases:

Case 1. $n > 1$ and $t_1 = \alpha_1 \in \Sigma$. We have $e(w) = e(\tilde{w}) = 1$.

Case 2. $n > 1$ and $|t_1| \geq 1$. We have $e(w) = 2$ and $e(\tilde{w}) = 1$.

Case 3. $n = 1$ and $t_1 = \alpha_1 \in \Sigma$. We have $e(w) = e(\tilde{w}) = 0$.

Case 4. $n = 1$ and $|t_1| \geq 1$. We have $e(w) = 1$, $e(\tilde{w}) = 0$, and $\tilde{w}$ consists of a single nonterminal $A \in N$. □

For the tree t_m from Figure 1 we have $|\mathrm{bdag}(t_m)| = |t_m| = 2m$, $|\mathrm{hdag}(s_m)| = |\mathrm{dag}(t_m)| = m + 1$, and $n = |N| = 2$. Hence, Theorem 18 is optimal.

From Theorems 15, 16, and 18 we immediately get:

Corollary 19 *For every $t \in \mathcal{T}(\Sigma)$ with $|t| \geq 2$, we have*

$$\frac{1}{2}|bdag(t)| \leq |dag(t)| \leq \frac{1}{2}|bdag(t)|^2.$$

7 The average-case sizes of dag and bdag

Let $m \geq 1$. We use the terminology *m-labeled tree*, instead of *$\{1, \ldots, m\}$-labeled tree*. In this section, we analyze the average sizes (both node size and edge size) of the dags and bdags of m-labeled unranked trees of size n. Currently, we are not able to make such an analysis for the hdag. While Section 7.1 provides exact expressions for the average sizes, Section 7.2 deals with their asymptotic behavior. The results are mostly an extension of [13], where the authors treat the average node size of binary trees over a singleton alphabet ($m = 1$). However, we give here complete proofs, whereas the proof was merely sketched in [13].

Let $\mathcal{B}_m$ denote the set of non-empty m-labeled binary trees and let $\mathcal{T}_m$ denote the set of non-empty m-labeled unranked trees. Here, "non-empty" means that our trees have at least one node. For $\mathcal{U} \in \{\mathcal{B}, \mathcal{T}\}$ and $n \geq 0$, we define

$$\mathcal{U}_{m,n} = \{t \in \mathcal{U}_m \mid |t| = n\}.$$

We seek expressions for the accumulated quantities

$$N^{\mathcal{U}}_{m,n} = \sum_{t \in \mathcal{U}_{m,n}} \|\mathrm{dag}(t)\| \quad \text{and} \quad E^{\mathcal{U}}_{m,n} = \sum_{t \in \mathcal{U}_{m,n}} |\mathrm{dag}(t)|$$

as well as for the average sizes

$$\bar{N}^{\mathcal{U}}_{m,n} = \frac{N^{\mathcal{U}}_{m,n}}{|\mathcal{U}_{m,n}|} \quad \text{and} \quad \bar{E}^{\mathcal{U}}_{m,n} = \frac{E^{\mathcal{U}}_{m,n}}{|\mathcal{U}_{m,n}|}.$$

Recall that the fcns-encoding yields a bijection between m-labeled unranked trees of edge size n and m-labeled binary trees of edge size n, where the root only contains a left child. Therefore, the average node size (resp. edge size) of the bdag of m-labeled unranked trees of size n is one plus the average node size (resp. edge size) of the minimal dag of m-labeled binary trees of size $n-1$.

One key tool used in this section is that of *generating functions*. If F_n is a sequence of numbers, then its (ordinary) generating function is defined as

$$\mathbf{F}(z) = \sum_{n \geq 0} F_n z^n$$

and $[z^n]\mathbf{F}(z)$ denotes the coefficient of z^n in $\mathbf{F}(z)$ (i.e., F_n). If for a set $\mathcal{S}$ a size function $f : \mathcal{S} \to \mathbb{N}$ is defined such that for every n the set $\mathcal{S}_n = \{s \in \mathcal{S} \mid f(s) = n\}$ is finite, we can associate to the set $\mathcal{S}$ the generating function

$$\mathbf{S}(z) = \sum_{n \geq 0} |\mathcal{S}_n| z^n,$$

which is said to *count the objects of $\mathcal{S}$ by their size*. Such sets $\mathcal{S}$ are sometimes called *combinatorial classes* [12, p.16]. When a class has a simple recursive structure, it is often possible to obtain an explicit expression for $\mathbf{S}(z)$. This will be the case for our two basic generating functions, $\mathbf{B}_m(z)$ and $\mathbf{T}_m(z)$,

which count respectively the trees of $\mathcal{B}_m$ and $\mathcal{T}_m$ by their size (see Lemmas 20 and 23).

Let again $\mathcal{U} \in \{\mathcal{B}, \mathcal{T}\}$. For $u \in \mathcal{U}_m$ and $n \geq 0$, define $C^{\mathcal{U}}_{m,n}(u)$ as the number of $\mathcal{U}_m$-trees of size n that contain u as a subtree. Let $v \in \mathcal{U}_m$ be another tree such that $|v| = |u|$. For every $n \geq 0$ there is a bijection between the set of trees of size n that contain u and the set of trees of size n that contain v (it is obtained by replacing every occurrence of u by a copy of v, and vice versa). Therefore $C^{\mathcal{U}}_{m,n}(u) = C^{\mathcal{U}}_{m,n}(v)$ and so we also write $C^{\mathcal{U}}_{m,n}(p)$ (with $p = |u|$) instead of $C^{\mathcal{U}}_{m,n}(u)$. The corresponding generating function is

$$\mathbf{C}^{\mathcal{U}}_{m,p}(z) = \sum_{n \geq 0} C^{\mathcal{U}}_{m,n}(p) z^n.$$

This series will be determined in Lemma 21 for binary trees and in Lemma 24 for unranked trees. Let us now explain how the accumulated sizes $N^{\mathcal{U}}_{m,n}$ and $E^{\mathcal{U}}_{m,n}$ (or, equivalently, the associated generating functions) can be expressed in terms of these series.

Let $\mathrm{sub}(t)$ denote the set of subtrees occurring in the tree t. Since $\|\mathrm{dag}(t)\|$ is the number of different subtrees of t, we have ($\mathbb{1}_{u \in \mathrm{sub}(t)}$ is 1 if $u \in \mathrm{sub}(t)$ and 0 otherwise)

$$\begin{aligned} N^{\mathcal{U}}_{m,n} = \sum_{t \in \mathcal{U}_{m,n}} \|\mathrm{dag}(t)\| &= \sum_{t \in \mathcal{U}_{m,n}} \sum_{u \in \mathcal{U}_m} \mathbb{1}_{u \in \mathrm{sub}(t)} \\ &= \sum_{u \in \mathcal{U}_m} C^{\mathcal{U}}_{m,n}(u) = \sum_{p \geq 0} |\mathcal{U}_{m,p}|\, C^{\mathcal{U}}_{m,n}(p). \end{aligned}$$

Hence the corresponding generating function is

$$\mathbf{N}^{\mathcal{U}}_{m}(z) = \sum_{n \geq 0} N^{\mathcal{U}}_{m,n} z^n = \sum_{p \geq 0} |\mathcal{U}_{m,p}|\, \mathbf{C}^{\mathcal{U}}_{m,p}(z). \tag{6}$$

We now want to obtain an expression for the accumulated number of edges $E^{\mathcal{U}}_{m,n}$ and the associated generating function. Let us denote by $U^{(d)}_{m,n}$ (with $U = B$ or T) the number of trees from $\mathcal{U}_{m,n}$ that have root degree d (i.e., the root has d children). Then, in the same spirit as for the number of nodes, we get for the number of edges:

$$\begin{aligned} E^{\mathcal{U}}_{m,n} = \sum_{t \in \mathcal{U}_{m,n}} |\mathrm{dag}(t)| &= \sum_{t \in \mathcal{U}_{m,n}} \sum_{u \in \mathrm{sub}(t)} \deg(\mathrm{root}(u)) \\ &= \sum_{u \in \mathcal{U}_m} \deg(\mathrm{root}(u)) C^{\mathcal{U}}_{m,n}(u) = \sum_{p,d \geq 0} d\, U^{(d)}_{m,p} C^{\mathcal{U}}_{m,n}(p). \end{aligned}$$

The associated generating function is

$$\mathbf{E}^{\mathcal{U}}_{m}(z) = \sum_{n \geq 0} E^{\mathcal{U}}_{m,n} z^n = \sum_{p,d \geq 1} d\, U^{(d)}_{m,p} \mathbf{C}^{\mathcal{U}}_{m,p}(z). \tag{7}$$

Indeed, we can ignore trees of size $p = 0$ (or, equivalently, root degree $d = 0$).

7.1 Exact counting

In this section, we determine explicit expressions for the generating functions $\mathbf{N}_m^{\mathcal{U}}(z)$ and $\mathbf{E}_m^{\mathcal{U}}(z)$, whose coefficients record the accumulated number of nodes (resp. edges) in the dag of m-labeled trees of size n. We start with binary trees (that is, $\mathcal{U} = \mathcal{B}$).

7.1.1 Binary trees

Lemma 20 *The generating function $\mathbf{B}_m(z)$ of m-labeled binary trees, counted by their edge number, is*

$$\mathbf{B}_m(z) = \frac{1 - 2mz - \sqrt{1-4mz}}{2mz^2}. \tag{8}$$

Equivalently, the number of m-labeled binary trees of size p is

$$B_{m,p} = \frac{1}{p+2}\binom{2p+2}{p+1} m^{p+1}. \tag{9}$$

Of course the case $m = 1$ recovers the (shifted) Catalan numbers.

Proof The proof of Lemma 20 follows a general principle called the *symbolic method* in [12, Chapter 1]: If a combinatorial class $\mathcal{H}$ is built by disjoint union from two combinatorial classes $\mathcal{F}$ and $\mathcal{G}$ with the respective generating functions $\mathbf{F}(z)$ and $\mathbf{G}(z)$, then the generating function $\mathbf{H}(z)$ of the combinatorial class $\mathcal{H}$ is $\mathbf{H}(z) = \mathbf{F}(z) + \mathbf{G}(z)$. Similarily, if $\mathcal{H}$ is build via Cartesian product from the classes $\mathcal{F}$ and $\mathcal{G}$, then $\mathbf{H}(z) = \mathbf{F}(z) \cdot \mathbf{G}(z)$.

An m-labeled binary tree is either a single node with a label from $\{1, \dots, m\}$, or a root node with a single subtree (left or right) or a root node with two subtrees. The above principles give for the generating function $\mathbf{B}_m(z)$ the equation

$$\mathbf{B}_m(z) = m + 2mz\mathbf{B}_m(z) + m\left(z\mathbf{B}_m(z)\right)^2.$$

Solving this equation for $\mathbf{B}_m(z)$ proves equation (8) (taking the other root for $\mathbf{B}_m(z)$ would give a series with negative powers of z). Equation (9) follows from (8) by a Taylor expansion. □

Lemma 21 *The generating function $\mathbf{C}_{m,u}^{\mathcal{B}}(z)$ of m-labeled binary trees that contain a given tree $u \in \mathcal{B}_m$ of size p is*

$$\mathbf{C}_{m,p}^{\mathcal{B}}(z) = \frac{1}{2mz^2}\left(\sqrt{1-4mz+4mz^{p+2}} - \sqrt{1-4mz}\right).$$

Proof We first determine the generating function $\mathbf{A}_{m,p}^{\mathcal{B}}(z)$ counting m-labeled binary trees that do *not* contain (or *avoid*) u. A non-empty binary tree t that avoids u is either reduced to a single node, or a root node with a u-avoiding tree attached (this may be the left or the right child), or a root node to which two u-avoiding trees are attached. However, we must still exclude the tree

$t = u$, which is included in the above recursive description. We thus get the following equation:

$$\mathbf{A}^{\mathcal{B}}_{m,p}(z) = m + 2mz\mathbf{A}^{\mathcal{B}}_{m,p}(z) + m\left(z\mathbf{A}^{\mathcal{B}}_{m,p}(z)\right)^2 - z^p,$$

which yields

$$\mathbf{A}^{\mathcal{B}}_{m,p}(z) = \frac{1 - 2mz - \sqrt{1 - 4mz + 4mz^{p+2}}}{2mz^2}.$$

Using $\mathbf{C}^{\mathcal{B}}_{m,p}(z) = \mathbf{B}_m(z) - \mathbf{A}^{\mathcal{B}}_{m,p}(z)$, this proves the lemma. □

We now obtain expressions for the generating functions $\mathbf{N}^{\mathcal{B}}_m(z)$ and $\mathbf{E}^{\mathcal{B}}_m(z)$ given by (6) and (7).

Theorem 22 *The generating function of the accumulated number of nodes of minimal dags of m-labeled binary trees is*

$$\mathbf{N}^{\mathcal{B}}_m(z) = \frac{1}{2mz^2}\sum_{p\geq 0} B_{m,p}\left(\sqrt{1 - 4mz + 4mz^{p+2}} - \sqrt{1 - 4mz}\right),$$

where the numbers $B_{m,p}$ are given by (9).
The generating function of the accumulated number of edges of dags of m-labeled binary trees is

$$\mathbf{E}^{\mathcal{B}}_m(z) = \frac{3}{2mz^2}\sum_{p\geq 1} \frac{p}{2p+1} B_{m,p}\left(\sqrt{1 - 4mz + 4mz^{p+2}} - \sqrt{1 - 4mz}\right).$$

Equation (3) in [13] can be obtained from the above expression for $\mathbf{N}^{\mathcal{B}}_m(z)$ by setting $m = 1$ and by shifting the index (since the size is defined as the number of nodes in [13]).

Proof The expression for $\mathbf{N}^{\mathcal{B}}_m(z)$ follows directly from (6) and Lemma 21. To express the series $\mathbf{E}^{\mathcal{B}}_m(z)$, we first need to determine (according to (7)) the number $B^{(d)}_{m,p}$ of m-labeled binary trees of size $p \geq 1$ with root degree d. Note that d can only be 1 or 2. Clearly, $B^{(1)}_{m,p} = 2mB_{m,p-1}$, and thus $B^{(2)}_{m,p} = B_{m,p} - 2mB_{m,p-1}$. Hence, for $p \geq 1$,

$$\begin{aligned}\sum_{d\geq 1} d \cdot B^{(d)}_{m,p} &= 2mB_{m,p-1} + 2(B_{m,p} - 2mB_{m,p-1}) \\ &= 2(B_{m,p} - mB_{m,p-1}) \\ &= \frac{3p}{2p+1} B_{m,p},\end{aligned}$$

where the last equation follows from (9). The expression for $\mathbf{E}^{\mathcal{B}}_m(z)$ now follows, using (7) and Lemma 21. □

7.1.2 Unranked trees

Lemma 23 *The generating function $\mathbf{T}_m(z)$ of m-labeled unranked trees is*

$$\mathbf{T}_m(z) = \frac{1-\sqrt{1-4mz}}{2z}. \tag{10}$$

Equivalently, the number of m-labeled unranked trees of size p is

$$T_{m,p} = \frac{1}{p+1}\binom{2p}{p} m^{p+1}. \tag{11}$$

Again, we obtain the Catalan numbers when $m = 1$.

Proof An m-labeled unranked tree is a root node to which a sequence of m-labeled unranked trees is attached. We can now use another construction from [12, Chapter 1]: If $\mathcal{G}$ is a combinatorial class that does not contain an element of size 0, and the class $\mathcal{F}$ is defined as

$$\mathcal{F} = \{\epsilon\} + \mathcal{G} + (\mathcal{G} \times \mathcal{G}) + (\mathcal{G} \times \mathcal{G} \times \mathcal{G}) + \cdots,$$

then the generating function of $\mathcal{F}$ is

$$\mathbf{F}(z) = \frac{1}{1-\mathbf{G}(z)}$$

where $\mathbf{G}(z)$ is the generating function of $\mathcal{G}$.

In our case, $\mathbf{G}(z) = z\mathbf{T}_m(z)$ counts trees with root degree 1 and root label 1, and we thus obtain

$$\mathbf{T}_m(z) = \frac{m}{1-z\mathbf{T}_m(z)}.$$

Solving this for $\mathbf{T}_m(z)$ yields (10). We then obtain equation (11) by a Taylor expansion. □

Lemma 24 *The generating function of m-labeled unranked trees that contain a given tree u of size p is*

$$\mathbf{C}^{\mathcal{T}}_{m,p}(z) = \frac{z^{p+1} + \sqrt{1-4mz+2z^{p+1}+z^{2p+2}} - \sqrt{1-4mz}}{2z}.$$

Proof We first determine the generating function $\mathbf{A}^{\mathcal{T}}_{m,p}(z)$ counting m-labeled unranked trees that do *not* contain (or avoid) u. A tree that avoids u is a root node to which a sequence of u-avoiding trees is attached. As in the binary case, we still need to subtract z^p to avoid counting u itself. This gives

$$\mathbf{A}^{\mathcal{T}}_{m,p}(z) = \frac{m}{1-z\mathbf{A}^{\mathcal{T}}_{m,p}(z)} - z^p,$$

which can be solved for $\mathbf{A}^{\mathcal{T}}_{m,p}(z)$:

$$\mathbf{A}^{\mathcal{T}}_{m,p}(z) = \frac{1}{2z}\left(1 - z^{p+1} - \sqrt{1-4mz+2z^{p+1}+z^{2p+2}}\right).$$

Using $\mathbf{C}^{\mathcal{T}}_{m,p}(z) = \mathbf{T}_m(z) - \mathbf{A}^{\mathcal{T}}_{m,p}(z)$, this proves the lemma. □

Proposition 25 *The generating function of the accumulated node size of minimal dags of m-labeled unranked trees is*

$$\mathbf{N}_m^{\mathcal{T}}(z) = \frac{1}{2z} \sum_{p \geq 0} T_{m,p} \left(z^{p+1} + \sqrt{1 - 4mz + 2z^{p+1} + z^{2p+2}} - \sqrt{1 - 4mz} \right),$$

where the numbers $T_{m,p}$ are given by (11).

The generating function of the accumulated edge size of minimal dags of m-labeled unranked trees is

$$\mathbf{E}_m^{\mathcal{T}}(z) = \frac{3}{2z} \sum_{p \geq 0} \frac{pT_{m,p}}{p+2} \left(z^{p+1} + \sqrt{1 - 4mz + 2z^{p+1} + z^{2p+2}} - \sqrt{1 - 4mz} \right).$$

Proof The expression of $\mathbf{N}_m^{\mathcal{T}}(z)$ follows directly from (6) and Lemma 24. To express the series $\mathbf{E}_m^{\mathcal{T}}(z)$, we first need to determine (according to (7)) the number $T_{m,p}^{(d)}$ of m-labeled unranked trees of size p and root degree d, or, more precisely, the sum

$$\sum_{d \geq 1} d \cdot T_{m,p}^{(d)}$$

for any $p \geq 1$. This is done in [7, Corollary 4.1] in the case $m = 1$. It suffices to multiply by m^{p+1} to obtain the general case:

$$\sum_{d \geq 1} d \cdot T_{m,p}^{(d)} = \frac{3pT_{m,p}}{p+2}.$$

Combining (7) and Lemma 24 now gives the expression of $\mathbf{E}_m^{\mathcal{T}}(z)$. □

7.2 Asymptotic results

In this section we state asymptotic results for the average node and edge sizes of the dag of m-labeled binary trees, and of m-labeled unranked trees. The proofs are rather involved and assume some knowledge in analytic combinatorics [12]. Therefore, the proofs are are given in the Appendix.

7.2.1 Binary trees

Theorem 26 *The average number of nodes in the minimal dag of an m-labeled binary tree of size n satisfies*

$$\bar{N}_{m,n}^{\mathcal{B}} = 2\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n} \right) \right) \quad \textit{with} \quad \kappa_m = \sqrt{\frac{\ln(4m)}{\pi}}. \tag{12}$$

The proof is an application of the *singularity analysis* of Flajolet and Odlyzko, described in [12, Ch. VI]. One first determines the singular behavior of the series $\mathbf{N}_m^{\mathcal{B}}(z)$ given by Theorem 22 in the neighborhood of its *dominant* singularities (that is, singularities of minimal modulus).

Theorem 27 *The generating function* $\mathbf{N}_m^{\mathcal{B}}(z)$ *is analytic in the domain* D *defined by* $|z| < \frac{1}{2m}$ *and* $z \notin [\frac{1}{4m}, \frac{1}{2m}]$. *As* z *tends to* $\frac{1}{4m}$ *in* D, *one has*

$$\mathbf{N}_m^{\mathcal{B}}(z) = \frac{8\, m\, \kappa_m}{\sqrt{(1-4mz)\ln((1-4mz)^{-1})}} + O\left(\frac{1}{\sqrt{(1-4mz)\ln^3((1-4mz)^{-1})}}\right),$$

where κ_m *is defined as in Theorem 26.*

Granted this proposition, one can use the Transfer Theorem VI.4 of [12, p. 393], combined with the estimates of the coefficients of elementary series (see [12, Thm. VI.2, p. 385]) to obtain the asymptotic behavior of the accumulated node size of minimal dags of m-labeled binary trees of size n:

$$N_{m,n}^{\mathcal{B}} = [z^n]\mathbf{N}_m^{\mathcal{B}}(z) = \frac{2\kappa_m}{\sqrt{\pi}} \frac{4^{n+1} m^{n+1}}{\sqrt{n \ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right).$$

Since the numbers $B_{m,n}$, given by (9), satisfy

$$B_{m,n} = \frac{4^{n+1} m^{n+1}}{\sqrt{\pi} n^{3/2}} \left(1 + O\left(\frac{1}{n}\right)\right),$$

this gives Theorem 26. The proof of Theorem 27 can be found in the Appendix, Section A.1 (for $m = 1$) and Section A.2 (for general values of m).

For the edge size, one obtains in a similar fashion the following result.

Theorem 28 *The average number of edges in the minimal dag of an* m*-labeled binary tree of size* n *satisfies*

$$\bar{E}_{m,n}^{\mathcal{B}} = 3\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right) \tag{13}$$

with κ_m *as in Theorem 26.*

The proof is a simple adaptation of the proof of Theorem 26 and can be found in Section A.3 of the Appendix. Note the factor 3/2 between the node and edge sizes, which could be predicted by comparing the expressions of $\mathbf{N}_m^{\mathcal{B}}(z)$ and $\mathbf{E}_m^{\mathcal{B}}(z)$ in Theorem 22.

7.2.2 Unranked trees

Theorem 29 *The average number of nodes in the minmal dag of an* m*-labeled unranked tree of size* n *satisfies*

$$\bar{N}_{m,n}^{\mathcal{T}} = \kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right), \tag{14}$$

with κ_m *as in Theorem 26.*

	$\mathcal{B}_m$	$\mathcal{T}_m$
$\bar{N}_{m,n}$	$2\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right)$	$\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right)$
$\bar{E}_{m,n}$	$3\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right)$	$3\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right)$

Table 1 Overview over the different asymptotics. Recall that $\kappa_m = \sqrt{\frac{\ln 4m}{\pi}}$.

Thus the average node size of compressed unranked trees is about half the node size of compressed binary trees of the same size. Note that the same ratio holds between the heights of these trees [6,10,24].

The proof of Theorem 29 is very similar to the proof of Theorem 26. The required changes are described in Section A.4.

Theorem 30 *The average number of edges in the minimal dag of an m-labeled unranked tree of size n satisfies*

$$\bar{E}_n^{\mathcal{T}} = 3\kappa_m \frac{n}{\sqrt{\ln n}} \left(1 + O\left(\frac{1}{\ln n}\right)\right), \tag{15}$$

with κ_m as in Theorem 26.

In other words, asymptotically the edge size of compressed binary trees is equal to the edge size of compressed unranked trees. The proof of Theorem 30 is given in Section A.5.

Table 1 contains an overview of the results of this section.

8 Dag and string compression

As for the hdag, consider the forest $\mathrm{fcns}(t_1), \ldots, \mathrm{fcns}(t_n)$ of the binary encodings of the right-hand sides $t_1, \ldots, t_n$ of the reduced 0-SLT grammar $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$ for an unranked tree t. In the construction of the hdag we build the minimal dag of this forest. Therefore we only share repeated suffixes of child sequences, i.e., "right branching" trees in the binary encodings. Such trees can in fact be considered as *strings*. We now want to generalize the sharing of suffixes. Instead of only sharing suffixes of child sequences, we now apply an arbitrary grammar-based string compressor to (a concatenation of) the child sequences. Such a compressor infers a small straight-line context-free grammar for the given string.

Formally, a *straight-line context-free string grammar, SL grammar* for short, is a triple $G = (N, \Sigma, \rho)$, where

- N is a finite set of nonterminals
- Σ is a finite set of terminal symbols

– $\rho : N \to (N \cup \Sigma)^*$ is a mapping such that the binary relation $\{(X, Y) \mid X, Y \in N, \rho(X) \in (N \cup \Sigma)^* Y (N \cup \Sigma)^*\}$ is acyclic.

We do not need a start nonterminal for our purpose. From every word $u \in (N \cup \Sigma)^*$ we can derive exactly one terminal string $\mathrm{eval}_G(u)$ using the mapping ρ. Formally, we extend ρ to a morphism $\rho : (N \cup \Sigma)^* \to (N \cup \Sigma)^*$ by $\rho(a) = a$ for $a \in \Sigma$. Due to the above acyclicity condition, for every $u \in (N \cup \Sigma)^*$, there exists an $n \geq 1$ with $\rho^n(u) \in \Sigma^*$, and $\mathrm{eval}_G(u)$ is this string. We define the size of G as $\sum_{X \in N} |\rho(X)|$. As for SLTs we also write $X \to u$ if $\rho(X) = u$.

An *SL grammar-compressed Σ-labeled dag* is a tuple $D = (V, \gamma, \lambda, G)$ such that the following holds:

– $G = (N, V, \rho)$ is an SL grammar with terminal alphabet V
– $\gamma : V \to (V \cup N)^*$
– $\lambda : V \to \Sigma$ and
– the triple $d = (V, \gamma', \lambda)$ with $\gamma'(v) = \mathrm{eval}_G(\gamma(v)) \in V^*$ is an ordered Σ-labeled rooted dag.

We define $\mathrm{eval}(D) = \mathrm{eval}(d)$. We define the size $|D|$ of D as $|G| + \sum_{v \in V} |\gamma(v)|$. We say that D is *minimal* if d is the minimal dag for $\mathrm{eval}(D)$. Note that there are many minimal SL grammar-compressed dags for a given tree, since we do not make any restrictions on the SL grammar part G. In particular, G does not have to be size minimal.

Example 31 Here is an example of an SL grammar-compressed Σ-labeled dag $D = (V, \gamma, \lambda, G)$ with $\Sigma = \{a, b, c, f, g, h\}$ and

$$V = \{A_1, A_2, A_3, A_4, A, B, C\}.$$

The mappings γ and λ are shown below in the left and in the middle column in form of a 0-SLT grammar. For instance, $A_1 \to f(A, D, A_4, D, C)$ stands for $\lambda(A_1) = f$ and $\gamma(A_1) = ADA_4DC$. The SL grammar G is shown in the right column; it contains the nonterminals D and E.

$$\begin{array}{lll} A_1 \to f(A, D, A_4, D, C) & A \to a & D \to A_2 A_3 \\ A_2 \to g(E, A) & B \to b & E \to AA \\ A_3 \to h(E, B) & C \to c & \\ A_4 \to f(D) & & \end{array}$$

The size of this SL grammar-compressed dag is 14 and it represents the dag d with the following 0-SLT grammar $\mathcal{G}_d$:

$$\begin{array}{ll} A_1 \to f(A, A_2, A_3, A_4, A_2, A_3, C) & A \to a \\ A_2 \to g(A, A, A) & B \to b \\ A_3 \to h(A, A, B) & C \to c \\ A_4 \to f(A_2, A_3). & \end{array}$$

Also note that D is minimal.

By the following theorem, a given SL grammar-compressed dag for a tree t can be efficiently transformed into a 1-SLT grammar that produces the binary encoding of t.

Theorem 32 *An SL grammar-compressed Σ-labeled dag $D = (V, \gamma, \lambda, G)$ can be transformed in time $O(|D|)$ into a 1-SLT grammar G_1 such that $eval(G_1) = fcns(eval(D))$ and $|G_1| \leq |D| + 2(|V| + |N|)$.*

Proof Let $G = (N, V, \rho)$. Let $\hat{V} = \{\hat{v} \mid v \in V\}$, $V' = \{v' \mid v \in V\}$, $\hat{N} = \{\hat{X} \mid X \in N\}$, and $N' = \{X' \mid X \in N\}$ be disjoint copies of the sets V and N, respectively. The set of nonterminals of the 1-SLT grammar G_1 is $N \cup N' \cup \hat{N} \cup V \cup \hat{V} \cup V'$. Nonterminals in $N \cup V \cup V'$ have rank 0 and nonterminals in $\hat{N} \cup N' \cup \hat{V}$ have rank 1. The idea is that $\hat{\alpha}$ (for $\alpha \in N \cup V$) represents a copy of α that appears at positions in the fcns encoding having exactly one child (a right child), whereas the original α will only appear in leaf positions. This distinction is necessary since in an SLT grammar every nonterminal has a fixed rank. Nonterminals in $N' \cup V'$ are used in order to keep G_1 small.

The right-hand side mapping of G_1 is defined as follows: For every $v \in V$ with $\lambda(v) = f$ and $\gamma(v) = \alpha_1 \cdots \alpha_k$ ($k \geq 0$, $\alpha_1, \ldots, \alpha_k \in V \cup N$) we set:

- If $k = 0$, then $v \to f$ and $\hat{v}(y) \to f(\Box, y)$; the nonterminal v' is not needed in this case.
- If $k \geq 1$, then $v \to f(v', \Box)$, $\hat{v}(y) \to f(v', y)$, and $v' \to \hat{\alpha}_1(\cdots \hat{\alpha}_{k-1}(\alpha_k) \cdots)$.

Note that the total size of these productions is at most $k + 2$ (recall that we do not count edges to $\Box$-labeled nodes). Removing the nonterminal v' in the case $k \geq 1$ would result in a total size of $2k + 1$.

For the every $X \in N$ with $\rho(X) = \beta_1 \cdots \beta_m$ ($\beta_1, \ldots, \beta_m \in N \cup V$, $m \geq 2$ without loss of generality) we set $X \to X'(\beta_m)$, $\hat{X}(y) \to X'(\hat{\beta}_m(y))$, and $X'(y) \to \hat{\beta}_1(\cdots \hat{\beta}_{m-1}(y) \cdots)$. These rules have total size $m + 2$. Hence, the size of the resulting 1-SLT grammar G_1 is $|D| + 2(|V| + |N|)$ and the time needed to construct it is clearly bounded by $O(|G_1|) = O(|D|)$. It is easy to see that G_1 produces fcns(eval(D)). □

Theorem 32 implies that results for 1-SLT grammars carry over to SL grammar-compressed dags. For instance, finite tree automata [19] (with sibling constraints [22]) and tree-walking automata [22] can be evaluated in polynomial time over 1-SLT grammars and hence over SL grammar-compressed dags.

To construct an SL grammar-compressed dag for a tree t, we first construct dag(t) in linear time. Then we apply a grammar-based string compressor (e.g., RePair [16] or Sequitur [27]) to the child sequences of the dag. In this second phase we want to derive a small SL grammar for a set of strings and not a single string. To do this, we concatenate all child sequences of the dag separated by unique symbols. For instance, for the dag at the end of Example 31 we obtain the string

$$AA_2A_3A_4A_2A_3C\$_1AAA\$_2AAB\$_3A_2A_3.$$

An application of RePair to this string yields the grammar

$$S \to ADA_4DC\$_1EA\$_2EB\$_3D, \qquad D \to A_2A_3, \qquad E \to AA.$$

Then, the right-hand side $ADA_4DC\$_1EA\$_2EB\$_3D$ contains the right-hand sides of the γ-mapping of the SL grammar-compressed dag, whereas the two remaining productions $D \to A_2A_3$ and $E \to AA$ make up the SL grammar part.

The following example shows that our construction may compress $\mathrm{dag}(t)$ exponentially.

Example 33 Consider the tree $f(a, a, \ldots, a)$ with 2^n many a-leaves. Its dag has 2^n many edges. We apply a grammar-based string compressor to the string a^{2^n}. The string compressor may produce the string grammar

$$\begin{aligned} S' &\to A_1A_1 \\ A_i &\to A_{i+1}A_{i+1} \text{ for } 1 \le i \le n-2 \\ A_{n-1} &\to aa \end{aligned}$$

of size $2n$. Actually, RePair would produce such a grammar. The construction from the proof of Theorem 32 yields the following 1-SLT grammar, where we eliminate productions that do not reduce the total grammar size:

$$\begin{aligned} S &\to f(\hat{A}_1(A_1), \Box) \\ A_i &\to \hat{A}_{i+1}(A_{i+1}) \text{ for } 1 \le i \le n-2 \\ \hat{A}_i(y) &\to \hat{A}_{i+1}(\hat{A}_{i+1}(y)) \text{ for } 1 \le i \le n-2 \\ A_{n-1} &\to a(\Box, a) \\ \hat{A}_{n-1}(y) &\to a(\Box, a(\Box, y)) \end{aligned}$$

The total size of this grammar is $3n - 1$. Hence we obtain a 1-SLT grammar for the fcns encoding of $f(a, a, \ldots, a)$ of size $O(n)$.

Both $\mathrm{hdag}(t)$ and $\mathrm{rhdag}(t)$ can be seen as particular minimal SL grammar-compressed dags. For instance, $\mathrm{hdag}(t)$ can be seen as a minimal SL grammar-compressed dag $D = (V, \gamma, \lambda, G)$, where the SL grammar $G = (N, V, \rho)$ is *right regular*, i.e., for every nonterminal $X \in N$ we have $\rho(X) \in V^*N \cup V^+$, and similarly, for every $v \in V$ we have $\gamma(v) \in V^*N \cup V^*$. When transforming such an SL grammar compressed dag into a 1-SLT grammar following the proof of Theorem 32, we do not need the copy sets $\hat{N}$ and N', because nonterminals from N always produce suffixes of child sequences in the dag. This implies the following:

Theorem 34 *An hdag that is represented as an SL grammar-compressed Σ-labeled dag $D = (V, \gamma, \lambda, G)$ can be transformed in time $O(|D|)$ into a 1-SLT grammar G_1 such that $\mathit{eval}(G_1) = \mathit{fcns}(\mathit{eval}(D))$ and $|G_1| \le |D| + 2|V|$.*

9 Subtree equality check

In the previous sections we have discussed five different formalisms for the compact representation of unranked trees:

(1) dag
(2) binary dag
(3) hybrid dag
(4) SL grammar-compressed dag
(5) SLT grammars (e.g. produced by BPLEX or TreeRePair)

As mentioned in Section 3, tree automata can be evaluated in polynomial time for SLT grammars, and hence the same holds for the above five formalisms. In this section we consider another important processing primitive: *subtree equality check*. Consider a program which realizes two independent node traversals of an unranked tree, using one of (1)–(5) as memory representation. At a given moment we wish to check if the subtrees at the two nodes of the traversals coincide. How expensive is this check? As it turns out, the formalisms behave quite differently for this task. The dags (1)–(3) as well as SL grammar-compressed dags (4) allow efficient equality check. We show below (Theorem 35) that for an appropriate representation of the two nodes, this test can be performed in time $O(\log N)$, where N is the number of tree nodes. For SLT grammars such a check is much more expensive. Note that we cannot unfold the subtrees and check node by node, as this can take exponential time. For SLT grammars a polynomial time algorithm is known, based on Plandowski's result [28]. A new, fine difference between the dags (1)–(3) on the one hand and (4) and (5) on the other hand is that we can also check equality of sibling sequences in time $O(\log N)$ for (1)–(3) (see Theorem 37). For (4) and (5) we are not aware of such an algorithm.

Let $t = (V, \gamma, \lambda) \in \mathcal{T}(\Sigma)$ be an unranked tree. Recall that the preorder traversal of t ($\mathrm{pre}(t)$ for short) enumerates the nodes of t by first enumerating the root, followed by the preorder traversals of the direct subtrees of the root. Formally, if r is the root of t and $\gamma(r) = v_1 \cdots v_n$, then $\mathrm{pre}(t) = r\,\mathrm{pre}(t/v_1) \cdots \mathrm{pre}(t/v_n)$. The preorder number of a node $u \in V$ is its position in $\mathrm{pre}(t)$. In what follows we identify a preorder number p with the node in t that it represents, and simply speak of "the node p". In particular, we denote with t/p ($1 \leq p \leq \|t\|$) the subtree rooted at node p.

Theorem 35 *Let t be an unranked tree with N nodes. Given $g = dag(t)$ or a minimal SL grammar-compressed dag g with $eval(g) = t$ (this includes the hdag) or $g = bdag(t)$, one can, after $O(|g|)$ time preprocessing, check for given $1 \leq p, q \leq N$ whether $t/p = t/q$ in time $O(\log N)$.*

Proof Let $t = (V, \gamma, \lambda) \in \mathcal{T}(\Sigma)$. First, consider $g = \mathrm{dag}(t) = (U, \gamma', \lambda')$. For $1 \leq p \leq N$ let y_p be the unique node of g such that $\mathrm{eval}_g(y_p) = t/p$. Then, $t/p = t/q$ if and only if $y_p = y_q$. Hence, it suffices to show that the dag-node y_p can be computed from p in time $O(\log N)$ (after $O(|g|)$ time preprocessing). For this, we use techniques from [3]. More precisely, we construct in time $O(|g|)$

an SL string grammar G' for the word $y_1 y_2 \cdots y_N \in U^*$. For this, we introduce for every node $u \in U$ of the dag g a nonterminal $\hat{u}$. If $\gamma'(u) = u_1 \cdots u_n$, then we set $\hat{u} \to u\hat{u}_1 \cdots \hat{u}_n$. It is straightforward to show that this SL string grammar produces the word $y_1 y_2 \cdots y_N$. Note that $|G'| = |g|$.

It now suffices to show that for a given number $1 \leq p \leq N$, the p-th symbol of $\mathrm{eval}(G')$ can be computed in time $O(\log N)$ after $O(|g|) = O(|G'|)$ time preprocessing. This is possible by [3, Theorem 1.1]. Actually, in order to apply this result, we first have to transform G' into Chomsky normal form, which is also possible in time $O(|G'|) = O(|g|)$.

For a minimal SL grammar-compressed dag $g = (U, \gamma, \lambda, G)$ for t, where $G = (N, U, \rho)$, essentially the same procedure as for the dag applies. The set of nonterminals of the SL string grammar G' is $\{\hat{u} \mid u \in U\} \cup N$. For $u \in U$ with $\gamma(u) = \alpha_1 \cdots \alpha_n$ (with $\alpha_i \in U \cup N$) we set $\hat{u} \to u\hat{\alpha}_1 \cdots \hat{\alpha}_n$, where $\hat{\alpha}_i = \hat{v}$ if $\alpha_i = v \in U$ and $\hat{\alpha}_i = \alpha_i$ if $\alpha_i \in N$. The right-hand sides for the G'-nonterminals from N are simply copied from the grammar G with every occurrence of a symbol $u \in U$ replaced by $\hat{u}$. The reader finds an example of the construction in Example 36 below.

Finally, for $g = \mathrm{bdag}(t) = (U, \gamma, \lambda)$ we can proceed similarly. Again we construct in time $O(|g|)$ an SL string grammar G'. The set of nonterminals of G' is $\{\hat{u} \mid u \in U\}$. For every $u \in U$ with $\lambda(u) \neq \square$ and $\gamma(u) = u_1 u_2$ we set $\hat{u} \to u\alpha_1\alpha_2$, where $\alpha_i = \varepsilon$ if $\lambda(u_i) = \square$ and $\alpha_i = \hat{v}$ if $\lambda(u_i) = v \in U$. Note that for given preorder numbers $1 \leq p, q \leq N$, the p-th symbol of $\mathrm{eval}(G')$ is equal to the q-th symbol of $\mathrm{eval}(G')$ if and only if the sibling sequences at nodes p and q of t are equal. But we want to check whether the subtrees rooted at p and q are equal. For this, assume that using [3, Theorem 1.1] we have computed in time $O(\log N)$ the p-th symbol $y_p \in U$ (resp. the q-th symbol $y_q \in U$) of $\mathrm{eval}(G')$. Then, $t/p = t/q$ is equivalent to the following conditions: (i) $\lambda(y_p) = \lambda(y_q)$ and (ii) either y_p and y_q do not have left children in g, or the left children coincide. Since these checks only require constant time, we obtain the desired time complexity. □

Example 36 Consider the following minimal SL grammar-compressed dag from Example 31:

$$\begin{array}{lll} A_1 \to f(A, D, A_4, D, C) & A \to a & D \to A_2 A_3 \\ A_2 \to g(E, A) & B \to b & E \to AA \\ A_3 \to h(E, B) & C \to c & \\ A_4 \to f(D) & & \end{array}$$

Then the construction from the proof of Theorem 35 yields the following SL grammar G':

$$\begin{array}{lll} \hat{A}_1 \to A_1 \hat{A} D \hat{A}_4 D \hat{C} & \hat{A} \to A & D \to \hat{A}_2 \hat{A}_3 \\ \hat{A}_2 \to A_2 E \hat{A} & \hat{B} \to B & E \to \hat{A}\hat{A} \\ \hat{A}_3 \to A_3 E \hat{B} & \hat{C} \to C & \\ \hat{A}_4 \to A_4 D & & \end{array}$$

This grammar produces the string

$$\mathrm{eval}(G') = A_1AA_2A^3A_3A^2BA_4A_2A^3A_3A^2BA_2A^3A_3A^2BC.$$

We observe that for general SLT grammars, a result such as the one of Theorem 35 is not known. To our knowledge, the fastest known way of checking $t/p = t/q$ for a given SLT grammar G for t works as follows: From G we can again easily build an SL string grammar G' for the preorder traversal of t, see, e.g. [5,23]. Assume that the subtree of t rooted in p (resp., q) consists of m (resp., n) nodes. Then we have to check whether the substring of $\mathrm{eval}(G')$ from position p to position $p+m-1$ is equal to the substring from position q to position $q+n-1$. Using Plandowski's result [28], this can be checked in time polynomial in the size of G' and hence in time polynomial in the size of the SLT grammar G. Note that more efficient alternatives than Plandowski's algorithm exist, see, e.g. [18] for a survey, but all of them require at least quadratic time in the size of the SL grammar.

In the context of XML document trees, it is also interesting to check equivalence of two sibling sequences. For the dag, bdag, and hdag, this problem can be solved again very efficiently:

Theorem 37 *Let t be an unranked tree with N nodes. Given $g = dag(t)$ or $g = bdag(t)$ or $g = hdag(t)$ we can, after $O(|g|)$ time preprocessing, check for given $1 \leq p, q \leq N$, whether $sibseq(p) = sibseq(q)$ in time $O(\log N)$.*

Proof The result for the dag follows from the hdag-case, since the hdag can be constructed in linear time from the dag by constructing the minimal dag for the forest consisting of the fcns encodings of the right-hand sides of $\mathcal{G}^{\mathrm{red}}_{\mathrm{dag}(t)}$ (recall that the minimal dag can be constructed in linear time [8]), and this linear time computation is part of the preprocessing. Furthermore, we have already dealt with the bdag in the last paragraph of the proof of Theorem 35. Hence, it remains to consider the hdag. We assume that the $g = \mathrm{hdag}(t)$ is given as a minimal SL grammar-compressed dag $D = (V, \gamma, \lambda, G)$, where the SL grammar $G = (N, V, \rho)$ is *right regular*, i.e., for every nonterminal $X \in N$ we have $\rho(X) \in V^*N \cup V^+$, and similarly, for every $v \in V$ we have $\gamma(v) \in V^*N \cup V^*$; see the end of Section 8. After introducing additional nonterminals, we can assume that for every $X \in N$ we have $\rho(X) \in VN \cup V$, and for every $v \in V$ we have $\gamma(v) \in N \cup \{\varepsilon\}$ (this transformation is possible in time $O(|g|)$). Then, the elements of $\mathrm{sib}(t) \setminus \{t\}$ correspond to the elements of N.

We now construct an SL string grammar G' as follows; see also Example 38 below: The set of nonterminals of G' is $\{\hat{X} \mid X \in N\} \cup V$ and the set of terminals is N. The start nonterminal is the root $r \in V$ of the hdag. For every $v \in V$ we set

$$v \to \begin{cases} \varepsilon & \text{if } \gamma(v) = \varepsilon \\ \hat{X} & \text{if } \gamma(v) = X \in N. \end{cases}$$

For every $X \in N$ we set

$$\hat{X} \to \begin{cases} Xv\hat{Y} & \text{if } \rho(X) = vY, v \in V, Y \in N \\ Xv & \text{if } \rho(X) = v \in V. \end{cases}$$

Then sibseq(p) = sibseq(q) holds for two numbers $1 \le p, q \le \|t\|$ if and only if $p = q = 1$ or $p > 1$, $q > 1$, and the $(p-1)$-th symbol of eval(G') is equal to the $(q-1)$-th symbol of eval(G'). We deal with the case $p = q = 1$ separately because the sibling sequence t corresponding to the root of t is not represented in eval(G') (the latter string has length $N - 1$). □

Example 38 Consider the following hdag (our running example from Section 4), written as an SL grammar-compressed dag of the form used in the proof of Theorem 37.

$$\begin{aligned} S &\to f(X_0), & X_0 &\to BX_1, \\ B &\to f(X_1), & X_1 &\to AX_2, \\ A &\to g(X_3), & X_2 &\to A, \\ C &\to a & X_3 &\to C. \end{aligned}$$

It produces the tree $t = f(f(g(a), g(a)), g(a), g(a))$, see Figure 3. The nonterminal X_0 represents the sibling sequence $f(g(a), g(a))g(a)g(a)$, X_1 represents $g(a)g(a)$, X_2 represents $g(a)$, and X_3 represents a. These are all sibling sequences except for the length-1 sequence t.

According to the construction from the proof of Theorem 37, we obtain the following SLT grammar G':

$$\begin{aligned} S &\to \hat{X}_0, & \hat{X}_0 &\to X_0B\hat{X}_1, \\ B &\to \hat{X}_1, & \hat{X}_1 &\to X_1A\hat{X}_2, \\ A &\to \hat{X}_3, & \hat{X}_2 &\to X_2A, \\ C &\to \varepsilon & \hat{X}_3 &\to X_3C. \end{aligned}$$

It produces the string

$$\text{eval}(G') = X_0X_1X_3X_2X_3X_1X_3X_2X_3.$$

For instance, at preorder positions 3 and 7 the same sibling sequence, namely $g(a)g(a)$ starts in the tree t. This sibling sequence is represented by the symbol X_1. Indeed, the 2nd and 6-th symbol in eval(G') is X_1.

For an SL grammar-compressed dag, the best solution for checking sibseq(p) = sibseq(q) we are aware of uses again an equality check for SL grammar-compressed strings.

File	Edges	mD	aC	mC	dag	bdag
1998statistics	28305	5	22.4	50	1377	2403
catalog-01	225193	7	3.1	2500	8554	6990
catalog-02	2240230	7	3.1	25000	32394	52392
dblp	3332129	5	10.1	328858	454087	677389
dictionary-01	277071	7	4.4	733	58391	77554
dictionary-02	2731763	7	4.4	7333	545286	681130
EnWikiNew	404651	4	3.9	34974	35075	70038
EnWikiQuote	262954	4	3.7	23815	23904	47710
EnWikiVersity	495838	4	3.8	43593	43693	87276
EnWikTionary	8385133	4	3.8	726091	726221	1452298
EXI-Array	226521	8	2.3	32448	95584	128009
EXI-factbook	55452	4	6.8	265	4477	5081
EXI-Invoice	15074	6	3.7	1002	1073	2071
EXI-Telecomp	177633	6	3.6	9865	9933	19808
EXI-weblog	93434	2	11.0	8494	8504	16997
JSTgene.chr1	216400	6	4.8	6852	9176	14606
JSTsnp.chr1	655945	7	4.6	18189	23520	40663
medline	2866079	6	2.9	30000	653604	740630
NCBIgene.chr1	360349	6	4.8	3444	16038	14356
NCBIsnp.chr1	3642224	3	9.0	404692	404704	809394
sprot39.dat	10903567	5	4.8	86593	1751929	1437445
SwissProt	2977030	4	6.7	50000	1592101	1453608
treebank	2447726	36	2.3	2596	1315644	1454520

Table 2 The XML documents in Corpus I, their characteristics, and dag/bdag sizes

10 Experiments

In this section we empirically compare the sizes of different dags of unranked trees, namely dag, bdag, rbdag, hdag, and rhdag. We also include a comparison with SL grammar-compressed dags with RePair [16] as the string compressor, as explained in Section 8, and with TreeRePair [20]. We are interested in the tree structure only, hence we did not compare with XML file compressors like Xmill [17] or XQueC [1].

10.1 Corpora

We use three corpora of XML files for our tests. For each XML document we consider the unranked tree of its element nodes; we ignore all other nodes such as texts, attributes, etc. One corpus (*Corpus I*) consists of XML documents that have been collected from the web, and which have often been used in the context of XML compression research, e.g., in [4,5,20]. Each of these files is listed in Table 2 together with the following characteristics: number of edges, maximum depth (mD), average number of children of a node (aC), and maximum number of children of a node (mC). Precise references to the origin of these files can be found in [20]. The second corpus (*Corpus II*) consists of all well-formed XML document trees with more than 10,000 edges and a depth

File	rbdag	hdag	rhdag	DS	TR
1998statistics	2360	1292	1243	561	501
catalog-01	10303	4555	6421	4372	3965
catalog-02	56341	27457	29603	27242	26746
dblp	681744	358603	362571	149964	156412
dictionary-01	75247	47418	46930	32139	22375
dictionary-02	653982	414356	409335	267944	167927
EnWikiNew	70016	35074	35055	9249	9632
EnWikiQuote	47690	23903	23888	6328	6608
EnWikiVersity	87255	43691	43676	7055	7455
EnWikTionary	1452270	726219	726195	81781	84107
EXI-Array	128011	95563	95563	905	1000
EXI-factbook	2928	3847	2355	1808	1392
EXI-Invoice	2068	1072	1069	96	108
EXI-Telecomp	19807	9933	9932	110	140
EXI-weblog	16997	8504	8504	44	58
JSTgene.chr1	14114	7901	7271	3943	4208
JSTsnp.chr1	37810	22684	19532	9809	10327
medline	381295	466108	257138	177638	123817
NCBIgene.chr1	10816	11466	7148	6283	5166
NCBIsnp.chr1	809394	404704	404704	61	83
sprot39.dat	1579305	1000376	908761	335756	262964
SwissProt	800706	1304321	682276	278915	247511
treebank	1244853	1250741	1131208	1121566	528372

Table 3 The compressed sizes of the documents.

of at least four from the *University of Amsterdam XML Web Collection*[4]. We decided on fixing a minimum size because there is no necessity to compress documents of very small size, and we chose a minimum depth because our subject is tree compression rather than list compression. Note that out of the over 180,000 documents of the collection, only 1,100 fit our criteria and are part of Corpus II (more than $27,000$ were ill-formed and more than $140,000$ had less than $10,000$ edges). The documents in this corpus are somewhat smaller than those in Corpus 1, but otherwise have similar characteristics (such as maximal depth and average number of children) as can be seen in Table 4. The third corpus (*Corpus III*) consists of term rewriting systems[5]. These are stored in XML files, but, are rather atypical XML documents, because their tree structures are trees with small rank, i.e., there are no long sibling sequences. This can be seen in Table 4, which shows that the average number of children is only 1.5 for these files.

10.2 Experimental setup

For the dag, bdag, rbdag, and hdag we built our own implementation. It is written in C++ (g++ version 4.6.3 with O3-switch) and uses Libxml 2.6 for XML parsing. It should be mentioned that these are only rough prototypes

[4] http://data.politicalmashup.nl/xmlweb/

[5] http://www.termination-portal.org/wiki/TPDB

Corpus	Edges	mD	aC	mC
I	$1.9 \cdot 10^6$	6.6	5.7	$8 \cdot 10^4$
II	79465	7.9	6.0	2925
III	1531	18	1.5	13.2

Table 4 Document characteristics, average values.

Corpus	Parse	dag	hdag	DS	TR
I	35	43	46	48	175
II	85	105	120	117	310
III	6.9	8.7	9.2	10.0	14.8

Table 5 Cumulative Running times (in seconds).

and that our code is not optimized at all. The running times listed in Table 5 should be understood with this in mind. For the RePair-compressed dag we use Gonzalo Navarro's implementation of RePair[6]. This is called "DS" in our tables. For TreeRePair, called "TR" in the tables, we use Roy Mennicke's implementation[7] and run with max_rank=1, which produces 1-SLT grammars. Our test machine features an Intel Core i5 with 2.5Ghz and 4GB of RAM.

10.3 Comparison

Consider first Corpus 1 and the numbers shown in Table 2 and 3. The most interesting file, concerning the effectiveness of the hybrid dag and of the reverse binary encoding, is certainly the medline file. Just like dblp, it contains bibliographic data. In particular, it consists of MedlineCitation elements; such elements have ten children, the last of which varies greatly (it is a MeshHeadingList node with varying children lists) and thus cannot be shared in the dag. This is perfect for the reverse hybrid dag, which first eliminates repeated subtrees, thus shrinking the number of edges to 653,604, and then applies the last child/previous sibling encoding before building a dag again. This last step shrinks the number of edges to impressive 257,138. In contrast, the reverse binary dag has a size of 381,295. Thus, for this document really the combination of both ways of sharing, subtrees and reversed sibling sequences, is essential. We note that in the context of the first attempts to apply dag compression to XML [4] the medline files were particularly pathological cases where dag compression did not work well. We now have new explanations for this: using *reverse* (last child/previous sibling) encoding slashes the size of the dag by almost one half. And using hybrid dags again brings an improvement of more than 30%. The dblp document is similar, but does not make use of optional elements at the end of long sibling lists. Thus, the reverse dags are not smaller for dblp, but the hybrid dag is indeed more than 20% smaller than the dag.

[6] http://http://www.dcc.uchile.cl/~gnavarro/software/

[7] http://code.google.com/p/treerepair/

C.	Input	dag	bdag	rbdag	hdag	G(hd)	rhdag	G(rh)	DS	TR
I	43021	7815	9292	8185	6270	6323	5220	5285	2523	1671
II	90036	13510	15950	14671	10884	11109	9806	10039	5162	3957
III	2095	354	391	390	319	362	320	364	324	310

Table 6 Accumulated sizes (in thousand edges). *C* stands for Corpus, *G(hd)* for the grammar size of the hdag and *G(rh)* for the grammar size of the reverse hybrid dag.

The treebank document, which is a ranked tree and does not contain long lists, gives hardly any improvement of hybrid dag over dag, but the reverse hybrid dag is somewhat smaller than the dag (by 5%). For treebank, TreeRePair is unchallenged and produces a grammar that is less than half the size of DS.

Next, consider the accumulated numbers for the three corpora in Table 6. For Corpus I, the reverse hdag is smaller than the dag by around 38% while the hdag is only around 25% smaller than the dag. As noted in Section 5, the somewhat surprising outcome that the reverse binary encoding enables better compression results from the custom that in many XML-documents optional elements are listed last. This means that there are more common prefixes than suffixes in child sequences. Hence the reverse schemes perform better. When we transform hdags into SLT grammars (according to Section 8), then we get a modest size increase of about 1–2%. For the term rewriting systems of Corpus III, the hdag improves about 10% over the dag. Represented as grammars, however, this improvement disappears and in fact we obtain an accumulated size that is slightly larger than the dag. Note that for this corpus, also TreeRePair (TR) is not much smaller than the dag, and DS is even smaller than TR. Compared to the dag, TreeRePair shares tree patterns (=connected subgraphs). Hence, the trees in Corpus III do not contain many repeated tree patterns which are not already shared by the dag. When we compare DS with TR, then we see on corpora I and II that TreeRePair grammars are on average around 34% smaller than DS, while on Corpus III it is only 23% smaller. On very flat files, such as the EXI documents in Table 2, DS is about as good as TreeRePair. For combined dag and string compression we also experimented with another grammar-based string compressor: Sequitur [27], but found the combined sizes to be larger than with RePair. Concerning running times (see Table 5) note that the dag-variants stay close to the parsing time, while TreeRePair needs considerably more time. Hence, dags should be used when light-weight compression is preferred.

11 Conclusion and future work

We compare the sizes of five different formalisms for compactly representing unranked trees:

(1) dag
(2) binary dag
(3) hybrid dag

(4) combination of dag and SL grammar-based string compression
(5) SLT grammars (e.g. produced by BPLEX or TreeRePair)

For the comparison of (1)–(3) we prove precise bounds: (i) the size of the binary dag of a tree is bounded by twice the size of the hybrid dag of the tree and (ii) the size of the unranked dag of a tree is bounded by the square of the size of the hybrid dag of the tree. As a corollary we obtain that the size of the dag is at least of the size of the binary dag, and at most the square of the size of the binary dag. We also prove that for (1)–(4), checking equality of the subtrees rooted at two given nodes of these structures, can be carried out in $O(\log N)$ time, where N is the number of nodes of the tree. One advantage of binary and hybrid dags is that they also support the efficient checking of equality of (ending) sibling sequences in $O(\log N)$ time.

Our experiments over two large XML corpora and one corpus consisting of term rewriting systems show that dags and binary dags are the quickest to construct. Out of the dags (1)–(3), the reverse hdag (which uses a last child/previous sibling encoding) gives the smallest results. On our XML corpora, using the reverse binary encoding instead of the standard first child/next sibling encoding gives a compression improvement of more than 20%. As a practical yardstick we observe: For applications where sibling sequence check is important, or where the uniqueness of the compressed structures is important, the hybrid dag is a good choice. If strong compression is paramount, then structures (4) and (5) are appropriate. The advantage of (4) over (5) is its support of efficient subtree equality test.

We give generating functions for the exact average sizes of dags and bdags over unranked trees on n nodes and m labels. We show that asymptotically the expected edge sizes of dags and bdags coincide, and that the node size of bdags is twice as large as the node size of dags.

In future work we would like extend our average-case size analysis also to the hybrid dag and to Repair-compressed trees. Further, we would like to apply our compression within other recent compression schemes in databases, such as for instance factorized databases [2].

Acknowledgements The first author acknowledges the hospitality of the Institute of Computer Science, Universität Leipzig, where this work was carried out. The second and fourth author were supported by the DFG grant LO 748/8. The third author was supported by the DFG grant INST 268/239 and by the Engineering and Physical Sciences Research Council project "Enforcement of Constraints on XML streams" (EPSRC EP/G004021/1).

A Appendix: Proofs of the asymptotic results

In this appendix we prove the results from Section 7.2. They rely on methods described in the book *Analytic Combinatorics* [12] from Flajolet and Sedgewick; in particular, the *Transfer Theorem* in Chapter 6.

A.1 Proof of Proposition 27 (case $m = 1$)

For notational simplicity, we first show the proof for unlabeled trees (i.e. a singleton alphabet, $m = 1$). In Section A.2 we adapt the proof to arbitrary alphabets. By Lemma 21 and Proposition 22, the generating function for the cumulated node size of compressed binary trees reads

$$\mathbf{N}(z) := \mathbf{N}_1^{\mathcal{B}}(z) = \sum_{p\geq 0} B_p \mathbf{C}_p(z)$$

with

$$B_p = \frac{1}{p+2}\binom{2p+2}{p+1} \quad \text{and} \quad \mathbf{C}_p(z) = \frac{\sqrt{1-4z+4z^{p+2}} - \sqrt{1-4z}}{2z^2}. \tag{16}$$

We want to prove Proposition 27, which, when $m = 1$, reads as follows.

Proposition 39 *The generating function $\mathbf{N}(z)$ is analytic in the domain D defined by $|z| < \frac{1}{2}$ and $z \notin [\frac{1}{4}, \frac{1}{2}]$. As z tends to $\frac{1}{4}$ in D, one has*

$$\mathbf{N}(z) = \frac{8\kappa}{\sqrt{(1-4z)\ln((1-4z)^{-1})}} + O\left(\frac{1}{\sqrt{(1-4z)\ln^3((1-4z)^{-1})}}\right).$$

where $\kappa = \sqrt{\frac{\ln 4}{\pi}}$.

The proof of this proposition works in several steps. We first show that the function $\mathbf{N}(z)$ is analytic in D. Then we split $\mathbf{N}(z)$ into three parts. The splitting depends on a threshold integer $n \equiv n(z)$. We treat each part independently: we estimate precisely one of them, and simply bound the other two. We then show that these two parts can be neglected for a suitable choice of $n(z)$.

Step 1: $\mathbf{N}(z)$ is analytic in D. We first prove that the series $\mathbf{C}_p(z)$ defined by (16) is analytic in D. This is implied by the following lemma.

Lemma 40 *Let $\Delta_p(z) = 1 - 4z + 4z^{p+2}$. For $p \geq 1$, this polynomial has exactly one root of modulus less than $1/2$. This root is real and larger than $1/4$. When $p = 0$, then $\Delta_p(z) = (1-2z)^2$, with a double root at $1/2$. In particular, $\Delta_p(z)$ does not vanish in D.*

More generally, the rational function $u_p := 4z^{p+2}/(1-4z)$ does not take any value from $(-\infty, -1]$ for $z \in D$.

Proof To prove the existence of a unique root of modulus less than $1/2$, we apply Rouché's theorem (see e.g. [12, p. 270]) by comparing $\Delta_p(z)$ to the auxiliary function $1 - 4z$, and using the integration contour $\{|z| = 1/2\}$. Details are very simple, and are left to the reader.

Since Δ_p has real coefficients, the unique root of modulus less than $1/2$ must be real. One then concludes the proof by observing that $\Delta_p(1/4) > 0$ while $\Delta_p(1/2) < 0$.

The proof of the final statement is similar, upon comparing the polynomials $c(1-4z) + 4z^{p+2}$ and $c(1-4z)$, for $c \geq 1$. □

In order to prove that $\mathbf{N}(z)$ itself is analytic in the domain D, we rewrite it as follows, still denoting $u_p = 4z^{p+2}/(1-4z)$:

$$\begin{aligned}
\mathbf{N}(z) &= \frac{\sqrt{1-4z}}{2z^2} \sum_{p\geq 0} B_p \left(\sqrt{1+u_p} - 1\right) \\
&= \frac{\sqrt{1-4z}}{2z^2} \sum_{p\geq 0} B_p \left(\frac{u_p}{2} + \sqrt{1+u_p} - 1 - \frac{u_p}{2}\right) \\
&= \frac{1}{\sqrt{1-4z}} \sum_{p\geq 0} B_p z^p + \frac{\sqrt{1-4z}}{2z^2} \sum_{p\geq 0} B_p \left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right) \\
&= \frac{1}{\sqrt{1-4z}} \mathbf{B}(z) + \frac{\sqrt{1-4z}}{2z^2} \sum_{p\geq 0} B_p \left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right),
\end{aligned} \tag{17}$$

where

$$\mathbf{B}(z) = \sum_{p\geq 0} B_p z^p = \frac{1-2z-\sqrt{1-4z}}{2z^2} \tag{18}$$

is the generating function of the shifted Catalan numbers.

Since $\sqrt{1-4z}$ and $\mathbf{B}(z)$ are analytic in D, it suffices to study the convergence of the sum over p in (17).

Lemma 41 *For all $u \in \mathbb{C} \setminus (-\infty, -1]$, we have*

$$\left|\sqrt{1+u} - 1 - \frac{u}{2}\right| \leq \frac{|u|^2}{2}.$$

Proof Write $x = \sqrt{1+u}$, so that $\Re(x) > 0$. Then $|u| = |x^2 - 1| = |x-1||x+1|$, so that the above inequality reads

$$\frac{|x-1|^2}{2} \leq \frac{|x-1|^2|x+1|^2}{2},$$

or equivalently $1 \leq |x+1|$, which is clearly true since $\Re(x) > 0$. □

Recall that all points of D have modulus less than $1/2$. Consider a disk included in D. For z in this disk, the quantity

$$|u_p|^2 = \frac{16|z|^{2p+4}}{|1-4z|^2}$$

is uniformly bounded by cr^p, for constants c and $r < 1/4$. By Lemma 40, we can apply Lemma 41 to u_p. Hence

$$\begin{aligned}\sum_{p\geq 0} B_p \left|\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right| &\leq \frac{1}{2}\sum_{p\geq 0} B_p |u_p|^2 \\ &\leq \frac{c}{2}\sum_{p\geq 0} B_p r^p < \infty.\end{aligned}$$

Hence the series occurring in (17) converges uniformly in the disk, and $\mathbf{N}(z)$ is analytic in D. □

Step 2: splitting $\mathbf{N}(z)$. We fix an integer n and write

$$\mathbf{N}(z) = \mathbf{N}^{(1)}(n,z) + \mathbf{N}^{(2)}(n,z) + \mathbf{N}^{(3)}(n,z), \tag{19}$$

where

$$\mathbf{N}^{(1)}(n,z) = \frac{\sqrt{1-4z}}{2z^2}\sum_{p=0}^{n} B_p\left(\sqrt{1+u_p} - 1\right),$$

$$\mathbf{N}^{(2)}(n,z) = \frac{1}{\sqrt{1-4z}}\left(\mathbf{B}(z) - \sum_{p=0}^{n} B_p z^p\right), \tag{20}$$

$$\mathbf{N}^{(3)}(n,z) = \frac{\sqrt{1-4z}}{2z^2}\sum_{p>n} B_p\left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right). \tag{21}$$

One readily checks that (19) indeed holds. Moreover, each $\mathbf{N}^{(i)}(n,z)$ is analytic in D for any i and n.

Step 3: an upper bound on $\mathbf{N}^{(1)}$.

Lemma 42 *For any* $u \in \mathbb{C} \setminus (-\infty, -1]$, *we have* $|\sqrt{1+u} - 1| \leq \sqrt{|u|}$.

Proof The proof is similar to the proof of Lemma 41. Write $x = \sqrt{1+u}$, so that $\Re(x) > 0$. Then the inequality we want to prove reads $|x-1| \leq |x+1|$, which is clearly true. □

Lemma 43 *Let* $y > 0$, *and define*

$$a(n,y) := \sum_{p=0}^{n} B_p y^p.$$

For any $c > 1/4$, *there exists a neighborhood of* c *such that*

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

uniformly in n *and* y *in this neighborhood of* c.

Proof Let us form the generating function of the numbers $a(n,y)$. One finds:

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

Thus

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

where

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

We now estimate $I(n,y)$ using a Cauchy integral:

$$I(n,y) = \frac{1}{2i\pi}\int_{\circlearrowleft} \frac{\sqrt{1-4xy}}{1-x}\frac{dx}{x^{n+3}},$$

where the integral is over a small circle around the origin. Recall that y is taken in a small neighborhood of $c > 1/4$. Hence the dominant singularity of the integrand is at $x = 1/(4y)$. (A second singularity occurs later at $x = 1$.) Writing $x = u/(4y)$ gives

$$I(n,y) = \frac{(4y)^{n+3}}{2i\pi}\int_{\circlearrowleft} \frac{\sqrt{1-u}}{4y-u}\frac{du}{u^{n+3}},$$

the integral being again over a small circle around the origin. We now deform the integration contour as discussed in [12, p. 382]. This gives

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

where $\mathcal{H}$ is the Hankel contour at distance 1 from the real positive axis described in [12, p. 382]. Hence the lemma will be proved if we can show that the integral over $\mathcal{H}$ is $O(1)$, uniformly in n and y. As in our favorite reference [12, p. 383], we split it into two parts, depending on whether $\Re(t) \leq \ln^2 n$ or $\Re(t) \geq \ln^2 n$. On the first part, $4y-1-t/n$ remains uniformly away from 0, while

$$\int_{\mathcal{H}\cap\{\Re(t)\leq \ln^2 n\}} \left|\sqrt{-t}\left(1+\frac{t}{n}\right)^{-n-3}\right| dt$$

can be shown to be $O(1)$, using the techniques of [12, p. 383] or [11]. On the second part, $|4y-1-t/n|$ reaches its minimal value $1/n$ when $\Re(t) = n(4y-1)$. Writing $t = x+i$, we can bound the modulus of the second part by

$$\begin{aligned} 2n\int_{x\geq \ln^2 n} \sqrt{2x}\left(1+\frac{x}{n}\right)^{-n-3} dx &\leq 2n\exp(-\ln^2(n))\int_{x\geq \ln^2 n} \sqrt{2x}\left(1+\frac{x}{n}\right)^{-3} dx \\ &\leq 2n^{3/2}\exp(-\ln^2(n))\int_{u\geq 0} \sqrt{2u}(1+u)^{-3} du \\ &= o(n^{-\alpha}) \end{aligned}$$

for any real α. This completes the proof of the lemma. □

Combining Lemmas 42 and 43 (with $c = 1/2$) gives, for z in D close enough to $1/4$:

$$\begin{aligned} |\mathbf{N}^{(1)}(n,z)| &\leq \frac{\sqrt{|1-4z|}}{2|z^2|}\sum_{p=0}^{n} B_p\sqrt{|u_p|} \\ &\leq \sum_{p=0}^{n} B_p|z|^{(p-2)/2} \\ &= O\left(4^n|z|^{n/2}n^{-3/2}\right), \end{aligned} \tag{22}$$

uniformly in n and z.

Step 4: an upper bound on $\mathbf{N}^{(3)}$. Let us combine Lemma 41 with the expression (21) of $\mathbf{N}^{(3)}(n,z)$. This gives:

$$\begin{aligned} \mathbf{N}^{(3)}(n,z) &\leq \frac{4z^2}{|1-4z|^{3/2}}\sum_{p>n} B_p|z|^{2p} \\ &\leq \frac{4z^2}{|1-4z|^{3/2}}\sum_{p>n} B_n 4^{p-n}|z|^{2p} \qquad \text{(since } B_{p+1}\leq 4B_p\text{)} \\ &\leq \frac{16|z|^{2n+4}}{|1-4z|^{3/2}} B_n \sum_{p>n} 4^{p-n-1}|z|^{2(p-n-1)} \\ &= O\left(\frac{|z|^{2n}}{|1-4z|^{3/2}} B_n\right) \end{aligned} \tag{23}$$

uniformly in n and z, for z in D close enough to $1/4$.

Step 5: estimate of $\mathbf{N}^{(2)}$. For z fixed, let us determine the generating function of the numbers $\mathbf{N}^{(2)}(n,z)$, given by (20). We find:

$$\sum_{n\geq 0}\mathbf{N}^{(2)}(n,z)x^n = \frac{1}{\sqrt{1-4z}}\frac{\mathbf{B}(z)-\mathbf{B}(xz)}{1-x}$$

where $\mathbf{B}(z)$ is defined by (18). Equivalently,

$$\sum_{n\geq 0}\mathbf{N}^{(2)}(n,z)x^n = \frac{1}{2x^2z^2\sqrt{1-4z}}\left((1+x)\sqrt{1-4z}-1-x+2xz+\frac{4z}{\sqrt{1-4xz}+\sqrt{1-4z}}\right).$$

Thus

$$\mathbf{N}^{(2)}(n,z) = \frac{2}{z\sqrt{1-4z}} I(n,z)$$

where

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

Using the same principles as in the proof of Lemma 43, we now estimate $I(n,z)$ using a Cauchy integral:

$$I(n,z) = \frac{1}{2i\pi} \int_{\circlearrowleft} \frac{1}{\sqrt{1-4xz}+\sqrt{1-4z}} \frac{dx}{x^{n+3}}$$

where the integral is over a small circle around the origin. The unique singularity of the integrand is at $x = 1/(4z)$. Writing $x = u/(4z)$ gives

$$I(n,z) = \frac{(4z)^{n+2}}{2i\pi} \int_{\circlearrowleft} \frac{1}{\sqrt{1-u}+\sqrt{1-4z}} \frac{du}{u^{n+3}}.$$

Changing the integration contour into a Hankel one, gives, as in Lemma 43,

$$I(n,z) = \frac{(4z)^{n+2}}{2i\pi\sqrt{n}} \int_{\mathcal{H}} \frac{1}{\sqrt{-t}+\sqrt{n(1-4z)}} \left(1+\frac{t}{n}\right)^{-n-3} dt.$$

Again, we split the integral into two parts, depending on whether $\Re(t) \leq \ln^2 n$ or $\Re(t) \geq \ln^2 n$. We assume moreover that

$$n \to \infty \quad \text{and} \quad n(1-4z) \to 0. \tag{24}$$

The first part of the integral is then

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

while the second part is found to be smaller than $(4z)^n n^{-\alpha}$, for any real α. Hence

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

so that

$$\mathbf{N}^{(2)}(n,z) = \frac{8(4z)^{n+1}}{\sqrt{\pi}\sqrt{1-4z}\sqrt{n}} \left(1 + O(\sqrt{n(1-4z)}) + O(n^{-1})\right). \tag{25}$$

Step 6: the threshold $n(z)$. We finally want to correlate $n \equiv n(z)$ and z so that, as z tends to $1/4$ (in the domain D), the function $\mathbf{N}(z)$ is dominated by $\mathbf{N}^{(2)}(n,z)$. Given the bounds (22) and (23) on $\mathbf{N}^{(1)}(n,z)$ and $\mathbf{N}^{(3)}(n,z)$, the estimate $B_n \sim 4^n n^{-3/2}$ (up to a multiplicative constant), and the estimate (25) of $\mathbf{N}^{(2)}$, we want

$$\frac{4^n |z|^{n/2}}{n^{3/2}} = o\left(\frac{(4z)^n}{\sqrt{1-4z}\sqrt{n}}\right)$$

and

$$\frac{4^n |z|^{2n}}{n^{3/2}|1-4z|^{3/2}} = o\left(\frac{(4z)^n}{\sqrt{1-4z}\sqrt{n}}\right).$$

We also want (24) to hold. These four conditions hold for

$$n = n(z) = \left\lfloor \frac{\ln|1-4z|}{\ln|z|} \right\rfloor.$$

Indeed, for this choice of n, we have

$$|z|^n = O(1-4z) \quad \text{and} \quad (4z)^n = 1 + O\big(|1-4z| \ln|1-4z|\big),$$

so that

$$\mathbf{N}^{(1)}(n,z) = O\left((1-4z)^{-1/2}\ln^{-3/2}\frac{1}{|1-4z|}\right),$$
$$\mathbf{N}^{(3)}(n,z) = O\left((1-4z)^{-1/2}\ln^{-3/2}\frac{1}{|1-4z|}\right),$$
$$\mathbf{N}^{(2)}(n,z) = \frac{8\sqrt{\ln 4}}{\sqrt{\pi}\sqrt{1-4z}\sqrt{\ln\frac{1}{|1-4z|}}}\left(1+O\left(\ln^{-1}\frac{1}{|1-4z|}\right)\right).$$

Finally, since

$$\ln|1-4z| = \ln(1-4z)\left(1+O\left(\ln^{-1}\frac{1}{|1-4z|}\right)\right),$$

and because z is close to $\frac{1}{4}$, we have at last obtained

$$\mathbf{N}(z) = \frac{8\sqrt{\ln 4}}{\sqrt{\pi}\sqrt{1-4z}\sqrt{\ln\frac{1}{1-4z}}}\left(1+O\left(\ln^{-1}\frac{1}{1-4z}\right)\right),$$

as stated in Proposition 39. □

A.2 Proof of Proposition 27 (case $m \geq 2$)

The proof is a straightforward adaptation of the proof of Theorem 26. We simply describe here the few necessary changes. We start from the expression of Proposition 22:

$$\mathbf{N}_m(z) := \mathbf{N}_m^{\mathcal{B}}(z) = \frac{1}{2z^2}\sum_{p\geq 0} B_p m^p\left(\sqrt{1-4mz+4mz^{p+2}} - \sqrt{1-4mz}\right) \tag{26}$$

where B_p still denotes the $(p+1)^{\text{st}}$-Catalan number.

Let us first prove that the series $\mathbf{N}_m(z)$ is analytic in the domain D defined by $|z| < 1/(2m)$ and $z \notin [1/(4m), 1/(2m)]$. The counterpart of Lemma 40 states that $1-4mz+4mz^{p+2}$ has exactly one root of modulus less than $1/(2m)$, and that this root is larger than $1/(4m)$. This now holds for any $p \geq 0$ (provided $m \geq 2$). Hence each series $\mathbf{C}_{m,p}^{\mathcal{B}}(z)$ is thus analytic in D. We also prove that $u_p := 4mz^{p+2}/(1-4mz)$ avoids the half-line $(-\infty, -1]$ for $z \in D$.

The proof that $\mathbf{N}_m(z)$ is also analytic in D transfers verbatim, once we have written

$$\mathbf{N}_m(z) = \frac{m}{\sqrt{1-4mz}}\mathbf{B}(mz) + \frac{\sqrt{1-4mz}}{2z^2}\sum_{p\geq 0} B_p m^p\left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right).$$

In Step 2, we split $\mathbf{N}_m(z)$ in the three following parts:

$$\mathbf{N}_m^{(1)}(n,z) = \frac{\sqrt{1-4mz}}{2z^2}\sum_{p=0}^{n} B_p m^p\left(\sqrt{1+u_p} - 1\right),$$
$$\mathbf{N}_m^{(2)}(n,z) = \frac{m}{\sqrt{1-4mz}}\left(\mathbf{B}(mz) - \sum_{p=0}^{n} B_p m^p z^p\right),$$
$$\mathbf{N}_m^{(3)}(n,z) = \frac{\sqrt{1-4mz}}{2z^2}\sum_{p>n} B_p m^p\left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right).$$

Now combining Lemmas 42 and 43 gives an upper bound for $\mathbf{N}_m^{(1)}$:

$$\mathbf{N}_m^{(1)}(n,z) = O\left((4m)^n|z|^{n/2}n^{-3/2}\right),$$

uniformly in n and z taken in some neighborhood of $1/(4m)$. The upper bound on $\mathbf{N}_m^{(3)}$ is found to be

$$\mathbf{N}_m^{(3)}(n,z) = O\left(\frac{m^n|z|^{2n}}{|1-4mz|^{3/2}}B_n\right).$$

Finally, since $\mathbf{N}_m^{(2)}(n,z)$ is simply $m\mathbf{N}^{(2)}(n,mz)$ (with $\mathbf{N}^{(2)}$ defined by (20)), we derive from (25) that

$$\mathbf{N}_m^{(2)}(n,z) = \frac{8(4mz)^{n+1}}{\sqrt{\pi}\sqrt{1-4mz}\sqrt{n}}\left(1+O(\sqrt{n(1-4mz)})+O(n^{-1})\right). \tag{27}$$

The threshold function is now

$$n = n(z) = \left\lfloor \frac{\ln|1-4mz|}{\ln|z|} \right\rfloor,$$

and the rest of the proof follows verbatim, leading to

$$\mathbf{N}_m(z) = \frac{8\sqrt{\ln(4m)}}{\sqrt{\pi}\sqrt{1-4mz}\sqrt{\ln\frac{1}{1-4mz}}}\left(1+O\left(\ln^{-1}\frac{1}{1-4mz}\right)\right),$$

which concludes the proof of Proposition 27. □

A.3 Proof of Theorem 28

The series $\mathbf{E}_m^{\mathcal{B}}(z)$ given in Proposition 22 can be written as in (26), upon replacing the numbers B_p by

$$\bar{B}_p = \frac{3p}{2p+1}B_p,$$

with generating function

$$\sum_{p\geq 0}\bar{B}_p z^p = \frac{1-3z-(1-z)\sqrt{1-4z}}{z^2}.$$

One can then adapt the proof of Proposition 27 without any difficulty. The only significant change is in the estimate (25) (and more generally (27)) of $\mathbf{N}_m^{(2)}(n,z)$, which is multiplied by a factor 3/2. This leads to the factor 3 in Theorem 28.

A.4 Proof of Theorem 29

The proof is again a variation on the proof of Theorem 26. Let us describe it directly for a general value of m.

Our first objective is to obtain the following counterpart of Proposition 27: The generating function $\mathbf{N}_m^{\mathcal{T}}(z)$ is analytic in the domain D defined by $|z| < \frac{1}{2m}$ and $z \notin [\frac{1}{4m}, \frac{1}{2m}]$. As z tends to $\frac{1}{4m}$ in D, one has

$$\mathbf{N}_m^{\mathcal{T}}(z) = \frac{m\kappa_m}{\sqrt{(1-4mz)\ln((1-4mz)^{-1})}} + O\left(\frac{1}{\sqrt{(1-4mz)\ln^3((1-4mz)^{-1})}}\right), \tag{28}$$

where κ_m is defined as in Theorem 26.

The transfer theorem from [12] then gives

$$N_{m,n}^{\mathcal{T}} = [z^n]\mathbf{N}_m^{\mathcal{T}}(z) = \frac{\kappa_m}{\sqrt{\pi}}\frac{4^n m^{n+1}}{\sqrt{n\ln n}}\left(1+O(\ln^{-1} n)\right).$$

Since the number of m-labeled unranked trees of size n is

$$T_{m,n} \sim \frac{4^n m^{n+1}}{\sqrt{\pi} n^{3/2}} \left(1 + O(n^{-1})\right),$$

this gives for the average number of nodes the expression of Theorem 29.

So let us focus on the proof of (28), which will mimic the proof of Proposition 27. We start from the expression of Proposition 25:

$$\mathbf{N}_m(z) := \mathbf{N}_m^{\mathcal{T}}(z) = \sum_{p \geq 0} T_p m^{p+1} \mathbf{C}_{m,p}^{\mathcal{T}}(z)$$

where

$$\mathbf{C}_{m,p}^{\mathcal{T}}(z) = \frac{z^{p+1} + \sqrt{1 - 4mz + 2z^{p+1} + z^{2p+2}} - \sqrt{1 - 4mz}}{2z}$$

and $T_p = T_{1,p}$ is the p-th Catalan number, with generating function

$$\mathbf{T}(z) = \sum_{p \geq 0} T_p z^p = \frac{1 - \sqrt{1 - 4z}}{2z} = 1 + z\mathbf{B}(z). \tag{29}$$

Recall that the series $\mathbf{B}(z)$ is defined by (18). We follow the same steps as in the proof of Proposition 27.

Step 1: $\mathbf{N}_m(z)$ is analytic in D. With Rouché's theorem we can prove that $\mathbf{C}_{m,p}^{\mathcal{T}}(z)$ is analytic in the domain D. We then define $u_p = z^{p+1}(2 + z^{p+1})/(1 - 4mz)$ and prove that u_p does not meet the half-line $(-\infty, -1]$ for $z \in D$. We then write

$$\mathbf{N}_m(z) = \frac{m}{2}\mathbf{T}(mz) + \frac{m\mathbf{T}(mz)}{2\sqrt{1-4z}} + \frac{mz\mathbf{T}(mz^2)}{4\sqrt{1-4mz}} + \frac{\sqrt{1-4mz}}{2z} \sum_{p \geq 0} T_p m^{p+1} \left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right)$$

to conclude, with the arguments of Sections A.1 and A.2, that $\mathbf{N}_m(z)$ is also analytic in D.

Step 2: splitting $\mathbf{N}_m(z)$. We fix an integer n and write

$$\mathbf{N}_m(z) = \mathbf{N}_m^{(1)}(n,z) + \mathbf{N}_m^{(2)}(n,z) + \mathbf{N}_m^{(3)}(n,z) + \mathbf{N}_m^{(4)}(n,z),$$

where

$$\begin{aligned}
\mathbf{N}_m^{(1)}(n,z) &= \frac{\sqrt{1-4mz}}{2z} \sum_{p=0}^{n} T_p m^{p+1} \left(\sqrt{1+u_p} - 1\right), \\
\mathbf{N}_m^{(2)}(n,z) &= \frac{m}{2\sqrt{1-4mz}} \left(\mathbf{T}(mz) - \sum_{p=0}^{n} T_p m^p z^p\right), \\
\mathbf{N}_m^{(3)}(n,z) &= \frac{\sqrt{1-4mz}}{2z} \sum_{p>n} T_p m^{p+1} \left(\sqrt{1+u_p} - 1 - \frac{u_p}{2}\right), \\
\mathbf{N}_m^{(4)}(n,z) &= \frac{m}{2}\mathbf{T}(mz) + \frac{zm}{4\sqrt{1-4mz}} \left(\mathbf{T}(mz^2) - \sum_{p=0}^{n} T_p m^p z^{2p}\right).
\end{aligned}$$

One readily checks that (A.4) indeed holds. Moreover, each $\mathbf{N}_m^{(i)}(n,z)$ is analytic in D for any i and n.

Step 3: an upper bound on $\mathbf{N}_m^{(1)}$. Using the same ingredients as before, we find that, as in the binary case,

$$\mathbf{N}_m^{(1)}(n,z) = O\left((4m)^n |z|^{n/2} n^{-3/2}\right)$$

uniformly in n and z taken in a neighborhood of $1/(4m)$.

Step 4: an upper bound on $\mathbf{N}_m^{(3)}$. Again, the behavior remains the same as in the binary case:

$$\mathbf{N}_m^{(3)} = O\left(\frac{m^n |z|^{2n}}{|1-4mz|^{3/2}} T_n\right),$$

uniformly in n and z, for z in D close enough to $1/(4m)$.

Step 5: estimate of $\mathbf{N}_m^{(2)}$. Using (29), we observe that

$$\mathbf{N}_m^{(2)}(n,z) = \frac{m^2 z}{2} \mathbf{N}^{(2)}(n-1, mz),$$

with $\mathbf{N}^{(2)}(z)$ defined by (20). Hence the estimate (25) gives

$$\mathbf{N}_m^{(2)}(n,z) = \frac{m(4mz)^{n+2}}{\sqrt{\pi}\sqrt{1-4mz}\sqrt{n}} \left(1 + O(\sqrt{n(1-4mz)}) + O(n^{-1})\right). \tag{30}$$

Step 6: an upper bound of $\mathbf{N}_m^{(4)}$. Given that $m|z|^2 < 1/(4m) \leq 1/4$, we can write

$$\begin{aligned}\left|\mathbf{N}_m^{(4)}(n,z)\right| &\leq \frac{m}{2}|\mathbf{T}(mz)| + \frac{m|z|}{4\sqrt{|1-4mz|}} \sum_{p>n} T_p m^p |z|^{2p} \\ &= O\left(1 + \frac{T_n m^n z^{2n}}{\sqrt{|1-4mz|}}\right)\end{aligned}$$

for z in a neighborhood of $1/(4m)$. The argument is the same as for the bound (23).

Step 7: the threshold $n(z)$. Using the same threshold function as in the binary case, we see that $\mathbf{N}_m(n,z)$ is dominated by $\mathbf{N}_m^{(2)}(n,z)$, and more precisely, that (28) holds.

A.5 Proof of Theorem 30

Comparing the two series of Proposition 25 shows that it suffices to replace the numbers T_p by

$$\bar{T}_p = \frac{3p}{p+2} T_p,$$

with generating function

$$\sum_{p\geq 0} \bar{T}_p z^p = \frac{1-3z-(1-z)\sqrt{1-4z}}{2z^2},$$

to go from $\mathbf{N}_m^{\mathcal{T}}(z)$ to $\mathbf{E}_m^{\mathcal{T}}(z)$. One can then adapt the proof of Proposition 29 without any difficulty. The only significant change is in the estimate (30) of $\mathbf{N}_m^{(2)}(n,z)$, which is multiplied by a factor 3. This leads to the factor 3 in Theorem 30.

References

1. A. Arion, A. Bonifati, I. Manolescu, and A. Pugliese. XQueC: A query-conscious compressed XML database. *ACM Trans. Internet Techn.*, 7(2), 2007.
2. N. Bakibayev, D. Olteanu, and J. Zavodny. Fdb: A query engine for factorised relational databases. *PVLDB*, 5(11):1232–1243, 2012.
3. P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings. In *SODA*, pages 373–389, 2011.
4. P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *VLDB*, pages 141–152, 2003.
5. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4-5):456–474, 2008.
6. N. G. de Bruijn, D. E. Knuth, and S. O. Rice. The average height of planted plane trees. In *Graph theory and computing*, pages 15–22. Academic Press, New York, 1972.
7. N. Dershowitz and S. Zaks. Enumerations of ordered trees. *Discrete Mathematics*, 31(1):9–28, 1980.
8. P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.
9. A. P. Ershov. On programming of arithmetic operations. *Commun. ACM*, 1(8):3–9, 1958.
10. P. Flajolet and A. Odlyzko. The average height of binary trees and other simple trees. *J. Comput. System Sci.*, 25(2):171–213, 1982.
11. P. Flajolet and A. Odlyzko. Singularity analysis of generating functions. *SIAM J. Discrete Math.*, 3(2):216–240, 1990.
12. P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
13. P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *ICALP*, pages 220–234, 1990.
14. D. E. Knuth. *The Art of Computer Programming, Vol. I: Fundamental Algorithms*. Addison-Wesley, 1968.
15. C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *VLDB*, pages 249–260, 2003.
16. N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *DCC*, pages 296–305, 1999.
17. H. Liefke and D. Suciu. XMILL: An efficient compressor for XML data. In *SIGMOD Conference*, pages 153–164, 2000.
18. M. Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complexity Cryptology*, 4:241–299, 2013.
19. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theor. Comput. Sci.*, 363(2):196–210, 2006.
20. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using repair. *Inf. Syst.*, 38(8):1150–1167, 2013.
21. M. Lohrey, S. Maneth, and E. Noeth. XML compression via dags. In *ICDT*, pages 69–80, 2013.
22. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.
23. S. Maneth and T. Sebastian. Fast and tiny structural self-indexes for XML. *CoRR*, abs/1012.5696, 2010.
24. J.-F. Marckert. The rotation correspondence is asymptotically a dilatation. *Random Structures Algorithms*, 24(2):118–132, 2004.
25. C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.
26. F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002.
27. C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res. (JAIR)*, 7:67–82, 1997.
28. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA*, pages 460–470, 1994.
29. T. Schwentick. Automata for XML - a survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.
30. D. Suciu. Typechecking for semistructured data. In *DBPL*, pages 1–20, 2001.