Compression of Unordered XML Trees*

Markus Lohrey[1], Sebastian Maneth[2], and Carl Philipp Reh[3]

1 **University of Siegen, Siegen, Germany**
`lohrey@eti.uni-siegen.de`
2 **University of Edinburgh, Edinburgh, UK**
`smaneth@inf.ed.ac.uk`
3 **University of Siegen, Siegen, Germany**
`reh@eti.uni-siegen.de`

Abstract

Many XML documents are data-centric and do not make use of the inherent document order. Can we provide stronger compression for such documents through giving up order? We first consider compression via minimal dags (directed acyclic graphs) and study the worst case ratio of the size of the ordered dag divided by the size of the unordered dag, where the worst case is taken for all trees of size n. We prove that this worst case ratio is $n/\log n$ for the edge size and $n \log\log n/\log n$ for the node size. In experiments we compare several known compressors on the original document tree versus on a canonical version obtained by length-lexicographical sorting of subtrees. For some documents this difference is surprisingly large: reverse binary dags can be smaller by a factor of 3.7 and other compressors can be smaller by factors of up to 190.

1998 ACM Subject Classification E.4 Coding and Information Theory

Keywords and phrases tree compression, directed acyclic graphs, XML

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.18

1 Introduction

Understanding the interplay between ordered and unordered structures is an important topic of database research. For XML this interplay has received considerable attention, see, e.g., [1, 4, 20, 3, 18]. A document is deemed *document-centric*, if the order of elements matters. Examples of such documents include web pages (e.g., in XHTML). In contrast, a document is *data-centric* if the order of elements is unimportant. For instance, the order of author-, title-, and year-elements in a bibliographic entry is unimportant. Of course, there could be mixtures of both, unordered and ordered nodes. For instance, an author-node could be marked "ordered" to contain subtrees for the first author, second author, etc. JSON naturally supports ordered and unordered nodes (cf. the Conclusions).

The absence of order bears many opportunities such as query optimization and set-oriented parallel processing, cf. [1]. Unordered XML has also been studied recently with respect to schema language definitions [4], a topic already considered during the birth years of XML [16]. Here we study the question whether *tree compression* can benefit from unorderedness.

In XML compression, document trees are typically stored (and compressed) separately from the data values, see, e.g., [14]. Let us first consider a very basic tree compression technique: directed acyclic graphs (dags, for short). Let t be an ordered tree. By representing repeated occurrences of the same subtree in t only once, one obtains a unique minimal dag

* The first author was supported by the DFG via project LO 748/10-1.

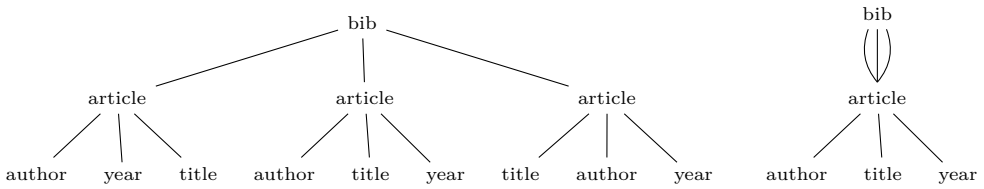

Figure 1 The tree structure of a bibliography on the left and its unordered dag on the right.

for t. In the following, we denote this minimal dag as *the dag of* t. It was observed early on that dags provide high compression ratios for common XML document trees [7] (10% on average for their documents). Moreover, the dag can be produced in linear time [9] or in amortized linear time using hashing (the well-known "hash-consing"), see, e.g., [7]. What happens if we construct the *unordered dag* of the tree t, which is the minimal dag of the unordered version of t?[1] Figure 1 shows an XML document tree consisting of 12 edges. Since there are no repeating subtrees (containing any edges), the minimal dag of this tree has 12 edges as well. In contrast, the unordered dag has only 6 edges. This raises the question how much smaller, at most, the unordered dag can be in comparison to the dag. We answer this question for two size measures: (i) the number of nodes and (ii) the number of edges. For each of these measures we study the maximum of the dag size of t divided by the unordered dag size of t, where the maximum is taken over all node-labelled trees t of size n. We denote these worst case ratios by $\alpha_N(n)$ (for the node size) and $\alpha_E(n)$ (for the edge size). Our main theoretical results provide precise growth rates (up to multiplicative constants) for these values:

(i) for the edge size we obtain $\alpha_E(n) \in \Theta(n/\log n)$ and
(ii) for the node size we obtain $\alpha_N(n) \in \Theta(n \log\log n/\log n)$.

With respect to the upper bound $\alpha_E(n) \in O(n/\log n)$ we show that for every tree t of size n, the unordered dag has at least $e/2 \cdot \ln(n/2)$ many edges (e is Euler's constant and ln is the logarithm to base e). This is shown using the well-known inequality between the geometric and arithmetic mean, see e.g., [17]. The upper bound $\alpha_N(n) \in \Theta(n \log\log n/\log n)$ uses a technique that has been applied in several related contexts, see e.g. [11]: one removes from a tree t all subtrees of size at most m, where m is logarithmic in the size of t. Then one bounds (i) the size of the remaining subtree and (ii) the number of different trees of size at most m. This yields an upper bound on the node size of the dag of t. For the lower bounds in (i) and (ii) one exploits the obvious fact that the list of subtrees of a node of rank r can be permuted in $r!$ many ways without affecting the corresponding unordered tree.

Let us also mention that the unordered dag can be computed in linear time as well. This can be done using the method for unordered tree isomorphism as given in Aho, Hopcraft, and Ullman's book [2].

In the second part of the paper, we contrast our theoretical results by experimental data for two corpora of XML trees. In addition to dags, we are interested in our experiments to gauge the impact of unorderedness of other tree compression methods. These are the dag variants introduced in [5] and the grammar-based tree compressor "TreeRePair" [15]. These are the strongest tree compressors that we are aware of. Instead of introducing (nontrivial) adaptations of each of these compressors to unordered trees, we opted for a

[1] Whenever we solely use the term dag (resp., tree), we always refer to the ordered version. If we want to speak about the unordered versions, we explicitly add the adjective "unordered".

different approach: we compress *canonical* trees. For this, we use the well-known canonization via length-lexicographical sorting of subtree lists, see., e.g., [8]. For an ordered tree t, it is easy to observe that the dag of t's canonical tree is isomorphic to the unordered dag of t. Our experimental results can be summarized as follows: for plain dags as explained above, the largest difference for any document of our collection is that the unordered dag has 60% size of the ordered dag. Then, essentially, the stronger the tree compression method as such, the larger the difference on the canonical tree. For the "hybrid dag" the largest difference is a factor of 3 smaller. For dag-plus-string-compression (called "DS" in [5]) we obtain some astonishing results: for one document tree (coming from Wikipedia data) the compression of the canonical tree is smaller by a factor of 190 than that of the original tree.

2 Preliminaries

With $e = 2,71828\cdots$ we always denote Euler's constant. With $\ln n$ (resp. $\log n$) we denote the logarithm of n to base e (resp., 2). For a positive integer k we denote by $[k]$ the set $\{1, 2, \ldots, k\}$. Let Σ be an alphabet. For a string $w = a_1 a_2 \cdots a_n \in \Sigma^*$ $(a_1, \ldots, a_n \in \Sigma)$ we denote by $\mathrm{alph}(w)$ the set $\{a_1, a_2, \ldots, a_n\}$ of symbols that appear in w. For $a \in \Sigma$ let $|w|_a$ denote the number $|\{i \in [n] \mid a_i = a\}|$ of occurrences of the symbol a in w. For $i \in [n]$ we denote the i-th letter a_i of w by $w[i]$. For $a \in \Sigma$ we denote with a^m the word $a \cdots a$ with m many occurrences of a.

3 Multi-graphs, Dags, and Trees

In this section we formally define node-labelled trees and dags in the ordered and unordered setting. Our definitions are non-standard in the sense that we define trees and dags as multi-graphs, whereas usually they are defined as ordinary graphs. Let Σ be an alphabet. A Σ-*labeled ordered dag* (or briefly dag) is a tuple $d = (V, \gamma, \lambda, v_0)$, where

- V is a finite set of nodes,
- $\gamma : V \to V^*$ assigns to each node a finite sequence of successor nodes,
- $\lambda : V \to \Sigma$ assigns to each node a label from Σ, and
- $v_0 \in V$ is the root node.

Moreover, we require that the *edge relation* $E_d := \{(u, v) \mid v \in \mathrm{alph}(\gamma(v))\}$ satisfies the following two properties:

- E_d is acyclic, i.e., there is no node v with $(v, v) \in E_d^+$ and
- every node v is reachable from v_0, i.e., $(v_0, v) \in E_d^*$.

Often we speak of the *multi-edges* of d. Formally, these are triples $(v, i, \gamma(v)[i])$ for $v \in V$ and $1 \leq i \leq |\gamma(v)|$. We use two size measures for a dag $d = (V, \gamma, \lambda, v_0)$:

- $\|d\| = \sum_{v \in V} |\gamma(v)|$ is the number of multi-edges, and
- $|d| = |V|$ is the number of nodes.

A *path* in d of length n is a sequence $v_1, k_1, v_2, k_2, \ldots, v_{n+1}$ such that (v_i, k_i, v_{i+1}) is a multi-edge of d for all $1 \leq i \leq n$. The *height* $h(d)$ of d is the length of a longest path in d. For a node $v \in V$, $\rho(v) = |\gamma(v)|$ is its rank and the rank of d is $\rho(d) = \max\{\rho(v) \mid v \in V\}$.

A Σ-*labeled ordered tree* (or briefly tree) can be defined as a dag $d = (V, \gamma, \lambda, v_0)$ such that every node $u \in V \setminus \{v_0\}$ has a unique occurrence in the set of strings $\{\gamma(v) \mid v \in V\}$. In other words: $\mathrm{alph}(\gamma(u)) \cap \mathrm{alph}(\gamma(v)) = \emptyset$ for $u \neq v$ and $|\gamma(u)|_v \leq 1$ for all $u, v \in V$. Alternatively, one can use terms over Σ to describe trees: if $t_1, \ldots, t_n$ $(n \geq 0)$ are trees and $f \in \Sigma$ then $f(t_1, \ldots, t_n)$ is also a tree. The set of all Σ-labeled ordered trees t with $\rho(t) \leq r$ is denoted as $\mathcal{T}_r(\Sigma)$. Moreover, let $\mathcal{T}_\infty(\Sigma) = \bigcup_{r \geq 1} \mathcal{T}_r(\Sigma)$. For $r \in \mathbb{N} \cup \{\infty\}$ and $k \in \mathbb{N}$ let $\mathcal{T}_{r,k} = \mathcal{T}_r([k])$. In

this notation, $\mathcal{T}_{\infty,1}$ is the set of all unlabelled trees (trees, where every node is labelled with the same symbol 1). For a tree t there is no essential difference between the size measures $\|t\|$ and $|t|$ (we have $\|t\| = |t| - 1$).

A dag $d = (V, \gamma, \lambda, v_0)$ can be unfolded into a tree $\tau(d)$. To define this tree, we associate with every node $v \in V$ the tree $\tau(v)$ inductively as follows: if $\lambda(v) = f$ and $\gamma(v) = v_1 v_2 \cdots v_n$ then $\tau(v) = f(\tau(v_1), \tau(v_2), \ldots, \tau(v_n))$. Finally, let $\tau(d) = \tau(v_0)$. For a tree t we define its *minimal dag* $\mathrm{dag}(t)$ as the smallest dag d with respect to $|d|$ such that $\tau(d) = t$. This is also the smallest dag d with respect to $\|d\|$ such that $\tau(d) = t$. The minimal dag $\mathrm{dag}(t)$ is unique up to isomorphism. It can be obtained from t by merging nodes u and v with $\lambda(u) = \lambda(v)$ and $\gamma(u) = \gamma(v)$ as long as possible. Also note that $|\mathrm{dag}(t)|$ is exactly the number of different subtrees of t. It is known that $\mathrm{dag}(t)$ can be computed in linear time [9].

There exist obvious unordered counterparts to the above definitions. A Σ-*labeled unordered dag* can be defined as a tuple $d = (V, \gamma, \lambda, v_0)$, where V, λ and v_0 have the same properties as for an ordered dag, and $\gamma : V \to \mathbb{N}^V$ assigns to each node v a V-indexed tuple of natural numbers, which can be seen as a multiset over V. Let us write $\gamma(u, v)$ instead of $(\gamma(u))(v)$, which is the number of multi-edges from u to v. It is required that the edge relation $E_d := \{(u, v) \in V \times V \mid \gamma(u, v) > 0\}$ is acyclic and that $(v_0, v) \in E_d^*$ for all $v \in V$. Unordered trees are then defined in the obvious way. As for ordered dags we define the two size measures $\|d\| = \sum_{u,v \in V} \gamma(u, v)$ and $|d| = |V|$. The unfolding of an unordered dag d is an unordered tree that we again denote with $\tau(d)$. This allows us to define the minimal dag $\mathrm{dag}(t)$ of the unordered tree t, which is an unordered dag. We make the following convention for the rest of the paper:

▶ Convention 1. Whenever we solely use the term dag (resp., tree), we always refer to the ordered version. If we want to speak about the unordered versions, we use the term unordered dag (resp., unordered tree).

For an ordered dag $d = (V, \gamma, \lambda, v_0)$, we define the *corresponding unordered dag* $d^u = (V, \gamma^u, \lambda, v_0)$, where $\gamma^u(v, w) = |\gamma(v)|_w$ is the number of occurrences of node w in the list $\gamma(v)$. For a tree t we define its *unordered minimal dag* $\mathrm{dag}^u(t)$ of t as the minimal dag of the corresponding unordered tree t^u. In the following we omit the adjective "minimal" when we speak of the minimal (unordered) dag of a tree. Figure 1 shows on the right the unordered dag of the tree on the left.

4 Sizes of Dags versus Unordered Dags

We clearly have $|\mathrm{dag}^u(t)| \leq |\mathrm{dag}(t)|$ and $\|\mathrm{dag}^u(t)\| \leq \|\mathrm{dag}(t)\|$. In this section we study the question, how much smaller the unordered dag for a given tree can be compared to its ordered dag. Formally, we study the growth of the following two worst case ratios, where $n, r, k \in \mathbb{N}$ and $r, k \leq n$:

$$\alpha_N(n, r, k) = \max\left\{ \frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \,\middle|\, t \in \mathcal{T}_{r,k}, |t| \leq n \right\},$$
$$\alpha_E(n, r, k) = \max\left\{ \frac{\|\mathrm{dag}(t)\|}{\|\mathrm{dag}^u(t)\|} \,\middle|\, t \in \mathcal{T}_{r,k}, |t| \leq n \right\}.$$

Let $x \in \{N, E\}$. Note that $\alpha_x(n, 1, k) = 1$ since for a tree $t \in \mathcal{T}_{1,k}$ (which is a linear chain) the ordered as well as the unordered dag is equal to t. Hence, we only consider the ratio $\alpha_x(n, r, k)$ for $r \geq 2$. Note that $\alpha_x(n, r, k) \leq \alpha_x(n, r', k')$ for all $r, r', k, k' \leq n$ with $r \leq r'$ and $k \leq k'$, since this implies $\mathcal{T}_{r,k} \subseteq \mathcal{T}_{r',k'}$. In the following we mainly concentrate on the extreme cases $\alpha_x(n, 2, 1)$ and $\alpha_x(n, n, n)$. We use the abbreviation $\alpha_x(n) = \alpha_x(n, n, n)$.

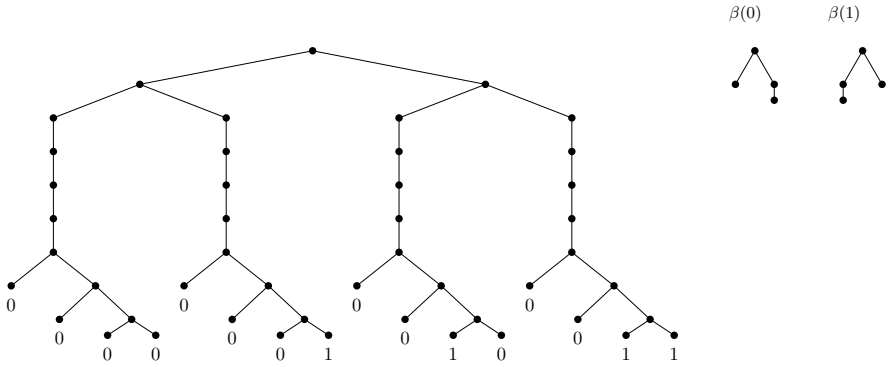

Figure 2 A possible choice for the tree t_{16} from Theorem 1 with $n = 16$, $r = h = 4$ and $k = 2$. For the 0- (resp., 1-labelled) nodes one has to substitute the tree $\beta(0)$ (resp., $\beta(1)$) on the right.

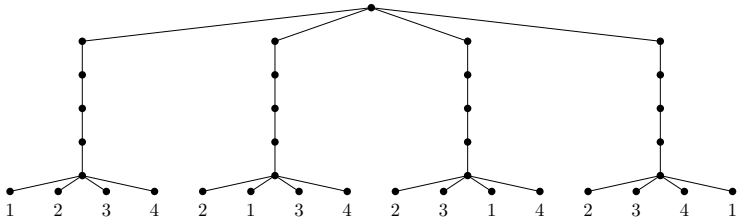

Figure 3 A possible choice for the tree t_{16} from Theorem 2 with $n = 16$, $x = \frac{1}{2}\log\log n = 1$, and $r = k = 4$.

4.1 Lower Bounds for α_E and α_N

In this section, we prove two lower bounds. In the first part we derive a lower bound of $\Omega(n/\log n)$ for $\alpha_E(n, 2, 1)$. For this, we construct a family of binary trees, where the dag achieves almost no compression, while the unordered dag achieves exponential compression ratios. Later, we show that this bound is tight by providing a matching upper bound for $\alpha_E(n)$. Note that $\alpha_N(n, 2, 1) \in \Theta(\alpha_E(n, 2, 1))$, since for a binary tree we have $|\mathrm{dag}(t)| - 1 \leq \|\mathrm{dag}(t)\| \leq 2 \cdot |\mathrm{dag}(t)|$ and analogously for $\mathrm{dag}^u(t)$. In the following theorem and its proof, we only consider trees from $\mathcal{T}_{2,1}$ (binary unlabelled trees). We denote such trees by well-parenthesized strings over (and). Formally, $() \in \mathcal{T}_{2,1}$ and if $t_1, t_2 \in \mathcal{T}_{2,1}$ then also $(t_1), (t_1t_2) \in \mathcal{T}_{2,1}$. By $B_h \in \mathcal{T}_{2,1}$ we denote the complete unlabelled binary tree with 2^h leaves and height h. This tree has size $2^{h+1} - 1$ and its dag has $2h$ multi-edges. For a tree t with k leaves and trees $t_1, \ldots, t_k$ we write $t[t_1, \ldots, t_k]$ to denote the tree obtained from t by replacing the i-th leaf (in pre-order) by t_i. For $k \geq 1$ let $c_k = (^k)^k$ denote a chain of k nodes. We encode non-empty bit strings by trees from $\mathcal{T}_{2,1}$ using the function β that is inductively defined as follows: $\beta(0) = (()(()))$, $\beta(1) = ((())())$, and $\beta(ds) = (\beta(d)\beta(s))$ for $d \in \{0, 1\}$ and $s \in \{0, 1\}^+$. The trees $\beta(0)$ and $\beta(1)$ are shown in Figure 2 on the right. Note that if $s_1, s_2 \in \{0, 1\}^+$ and $|s_1| = |s_2|$ then the unordered trees $\beta(s_1)^u$ and $\beta(s_2)^u$ are isomorphic. The construction in the proof of the following theorem is similar to a construction from [11].

▶ **Theorem 1.** *For every $n \geq 2$ there exists a tree $t_n \in \mathcal{T}_{2,1}$ with*

- $|t_n| \in \Theta(n)$
- $\|\mathrm{dag}(t_n)\| \in \Theta(n)$
- $\|\mathrm{dag}^u(t_n)\| \in \Theta(\log n)$

Hence, we have $\alpha_E(n, 2, 1) \in \Omega(n/\log n)$ and $\alpha_N(n, 2, 1) \in \Omega(n/\log n)$.

Proof. Let $n \in \mathbb{N}$ and $h = \lceil \log n \rceil$. Let $r = 2^k \in \Theta(n/\log n)$ be the smallest power of two that is at least n/h. Let $u_1, \ldots, u_r$ be r distinct bit strings of length h (note that $r \leq n \leq 2^h$). Consider the trees $s_1 = \beta(u_1), \ldots, s_r = \beta(u_r)$. We add to s_i a chain of length h and obtain the tree $s'_i = c_h[s_i]$ $(1 \leq i \leq r)$. Finally, set $t_n = B_k[s'_1, \ldots, s'_r]$. A possible choice for the tree t_{16} is shown in Figure 2.

Let us first bound the size of t_n. For B_k we have $|B_k| \in \Theta(r) = \Theta(n/\log n)$. The total size of all r copies of the chain c_h is $\Theta(r \cdot h) = \Theta(n)$. Finally, every s_i has size $\Theta(h)$; so their sizes sum up to $\Theta(r \cdot h) = \Theta(n)$. Altogether, we get $|t_n| \in \Theta(n)$.

To bound $\|\mathrm{dag}(t_n)\|$, note that the trees $s_1, \ldots, s_r$ are pairwise different (as ordered trees). This implies that in the dag of t_n, the r copies of the chains c_h are still present. Therefore, $\mathrm{dag}(t_n)$ has at least $r \cdot h \in \Theta(n)$ many nodes (and, of course, it has at most n nodes). Since every node of t_n has rank at most 2, we get $\|\mathrm{dag}(t_n)\| \in \Theta(n)$.

Finally, for the unordered dag note that the trees $s_1, \ldots, s_r$ are pairwise isomorphic when considered as unordered trees. Therefore, the copies of the chains c_h are collapsed into a single chain in $\mathrm{dag}^u(t_n)$ and also the top B_k-part is collapsed into $\Theta(\log r) = \Theta(\log n)$ many nodes. We get $\|\mathrm{dag}^u(t_n)\| \in \Theta(\log n)$ as well as $|\mathrm{dag}^u(t_n)| \in \Theta(\log n)$. ◀

In the next section, we will prove $\alpha_E(n) \in O(n/\log n)$, which yields the same upper bound for $\alpha_E(n, 2, 1)$. Moreover, also the lower bound of $\Omega(n/\log n)$ for $\alpha_N(n, 2, 1)$ turns out be sharp (see Corollary 7 for $k = 1$). On the other hand, for $\alpha_N(n) = \alpha_N(n, n, n)$ we can improve the lower bound to $\Omega(n \log\log n/\log n)$:

▶ **Theorem 2.** *Fix a constant $\delta > 1$. For every $n \geq 1$ large enough (depending on δ) there exists a tree $t_n \in \mathcal{T}_{r,k}$ with the following properties:*

- $k = \lceil \delta \cdot \log n/\log\log n \rceil$ *and* $r \in \Theta(n \cdot \log\log n/\log n)$.
- $|t_n| \in \Theta(n)$
- $|\mathrm{dag}(t_n)| \in \Theta(n)$
- $|\mathrm{dag}^u(t_n)| \in \Theta(\log n/\log\log n)$

Hence, we have $\alpha_N(n) \in \Omega(n \cdot \log\log n/\log n)$.

Proof. Fix $n \geq 1$ and let $x = \frac{1}{\delta}\log\log n$, $k = \lceil (\log n)/x \rceil = \lceil \delta \cdot \log n/\log\log n \rceil$, and $r = \lfloor n/k \rfloor \in \Theta(n \cdot \log\log n/\log n)$. Let us first show that with this choice we have $k! \geq r$. With Stirling's formula (or, more precisely, the inequality $z! \geq \sqrt{2\pi z} \cdot (z/e)^z$) we get

$$k! \geq (k/e)^k \geq (\log n/ex)^{(\log n)/x} = 2^{(\log\log n - \log(ex))(\log n)/x} = n^{(\log\log n - \log(ex))/x}.$$

Since moreover $n \geq n/k \geq r$, it suffices to show

$$n^{(\log\log(n) - \log(ex))/x} \geq n,$$

i.e., $\log\log n - \log(ex) \geq x = \frac{1}{\delta}\log\log n$, or, equivalently $(1 - \frac{1}{\delta})\log\log n \geq \log(ex) = \log(e/\delta) + \log\log\log n$, which holds for n large enough. This shows that, indeed, $k! \geq r$.

We now construct the tree $t_n \in \mathcal{T}_{r,k}$ as follows: take r many pairwise different trees $s_1, \ldots, s_r$ consisting of a root node with k many children, which are leaves. The sequence of labels of these k leaves forms a permutation of $[k]$. Since $k! \geq r$, these r pairwise different trees exist. From s_i we next construct s'_i by adding a chain of length k on top of s_i. Finally, the tree t_n is obtained by taking a new root node, whose children are the roots of the trees $s'_1, \ldots, s'_r$. A possible choice for the tree t_{16} is shown in Figure 3. Note that for n large enough we have $k \leq r$ since $k \in \Theta(\log n/\log\log n)$ and $r \in \Theta(n \cdot \log\log n/\log n)$. Hence, $t_n \in \mathcal{T}_{r,k}$.

We get $|t_n| = 1 + 2rk \in \Theta(n)$. For the node size of the dag, we obtain $|\mathrm{dag}(t_n)| = 1 + rk + k \in \Theta(n)$. Finally, for the node size of the unordered dag, note that the unordered trees corresponding to $s'_1, \ldots, s'_r$ are all isomorphic. Hence, we obtain that $|\mathrm{dag}^u(t_n)| = 1 + 2k \in \Theta(k) = \Theta(\log n / \log\log n)$, which proves the statement. ◀

4.2 Upper Bound for α_E

In this section we prove an upper bound for $\alpha_E(n)$ via a lower bound of $\Omega(\log n)$ for the function

$$\mu(n) := \min\{\|\mathrm{dag}^u(t)\| \mid t \in \mathcal{T}_{n,n}, |t| \leq n\}.$$

Thus, the unordered dag of a tree of size n has at least logarithmic size in n. Note that for binary trees (or trees of constant rank) this is obvious since the height of such a tree is at least $\log n$, which implies that also the minimal unordered dag has at least $\log n$ many edges. Also note that

$$\mu(n) = \min\{\|\mathrm{dag}(t)\| \mid t \in \mathcal{T}_{n,n}, |t| \leq n\}.$$

The reason is that for every tree t there is a tree t' with $|t| = |t'|$ and $\mathrm{dag}^u(t) = (\mathrm{dag}(t'))^u$ and thus $\|\mathrm{dag}^u(t)\| = \|\mathrm{dag}(t')\|$. Moreover, it holds that

$$\mu(n) = \min\{\|\mathrm{dag}(t)\| \mid t \in \mathcal{T}_{n,1}, |t| \leq n\},$$

i.e., it suffices to consider unlabelled trees. This is because adding labels to a tree can make the minimal dag only larger. Therefore we do not consider labels in the following and consider dags as triples (V, γ, v_0) without a labelling function λ.

Let $d = (V, \gamma, v_0)$ be such a dag. For a node $v \in V$ define $\mathrm{depth}(v)$ as the length of a *longest* path from the root v_0 to v. Thus, $\mathrm{depth}(v_0) = 0$. Note that $h(d) = \max\{\mathrm{depth}(v) \mid v \in V\}$. For $1 \leq i \leq h(d)+1$ let $V_i(d) = \{v \in V \mid \mathrm{depth}(v) = i-1\}$ be the set of nodes at depth $i-1$. Finally, let $\rho_i(d) = \sum_{v \in V_i(d)} |\gamma(v)|$ for $1 \leq i \leq h(d)$. This is the total number of multi-edges that start in a node at depth $i-1$. Every such multi-edge goes to a node at depth $j \geq i$. We write V_i and ρ_i for $V_i(d)$ and $\rho_i(d)$, respectively, if d is clear from the context.

▶ **Lemma 3.** *Let $d = (V, \gamma, v_0)$ be a dag of height $h = h(d)$. The number of leaves of the unfolding $\tau(d)$ is bounded by $\prod_{i=1}^{h} \rho_i$.*

Proof. Consider the dag $d' = (\{1, \ldots, h+1\}, \gamma', 1)$ with $\gamma'(i) = (i+1)^{\rho_i}$ (the string with ρ_i many occurrences of $i+1$) for $1 \leq i \leq h$ and $\gamma'(h+1) = \varepsilon$. It is a chain of $h+1$ nodes with ρ_i many multi-edges from node i to node $i+1$. The unfolding of d' has $\prod_{i=1}^{h} \rho_i$ many leaves. It therefore suffices to transform d into d' and show that this transformation does not reduce the number of leaves of the unfolding.

First of all, we can merge in d all nodes v with $\gamma(v) = \varepsilon$ to a single node. This does not change the unfolding, the height of the dag, and the number of multi-edges starting at depth i. Hence, V_{h+1} consists of the unique sink node of d; let us call this node s. Next, we construct from d the dag $d_1 = (V, \gamma_1, v_0)$, where γ_1 is defined as follows: we set $\gamma_1(s) = \varepsilon$. Now, let $v \in V_i$ with $1 \leq i \leq h$ and $\gamma(v) = v_1 v_2 \cdots v_r$. We set $\gamma_1(v) = v'_1 v'_2 \cdots v'_r$, where the nodes v'_j are defined as follows: if $v_j \in V_{i+1}$ then set $v'_j = v_j$. Otherwise, i.e., if $v_j \in V_k$ with $k > i+1$, then let v'_j be a node in V_{i+1} such that there exists a path from v'_j to v_j. Note that such a node v'_j exists, since every node in V_k ($k > 1$) has a predecessor in V_{k-1}. Note that the unfolding $\tau(v_j)$ is a subtree of the unfolding $\tau(v'_j)$. Therefore, $\tau(d_1)$ has indeed at least as many leaves as $\tau(d)$.

The dag d_1 has still height h and $\rho_i(d_1) = \rho_i(d)$ for $1 \leq i \leq h$. But in contrast to d, all multi-edges in d_1 go from a node in V_i to a node in V_{i+1} for some $1 \leq i \leq h$. Moreover, every node in V_i ($1 \leq i \leq h$) has at least one successor node (in V_{i+1}). If we now merge all nodes in V_i to a single node, we obtain (up to isomorphism) the dag d'. Clearly, this merging increases the number of paths from the root v_0 to the sink s. But the number of such paths is exactly the number of leaves in the unfolding. This shows the lemma. ◀

▶ **Theorem 4.** *We have* $\mu(n) \geq \frac{e}{2} \cdot \ln(n/2)$.

Proof. Let t be an arbitrary (unlabelled tree) of size n. We first transform t into a new tree t' by adding exactly one additional child node to every non-leaf of t. These new children are leaves in t'. Now t' has the property that every non-leaf node has at least two children. Note that $n \leq |t'| \leq 2n$. Moreover, for the dag we also have $\|\mathrm{dag}(t)\| \leq \|\mathrm{dag}(t')\| \leq 2 \cdot \|\mathrm{dag}(t)\|$. Intuitively, $\mathrm{dag}(t')$ is obtained from $\mathrm{dag}(t)$ by adding for every internal node v an additional multi-edge to the unique sink node of $\mathrm{dag}(t)$.

Let ℓ be the number of leaves of t'. Since every non-leaf node of t' has at least two children, we have $\ell \geq |t'|/2 \geq n/2$. Moreover, let h be the height of t' and let $\rho_i = \rho_i(\mathrm{dag}(t'))$. From Lemma 3 we obtain

$$\ell \leq \prod_{i=1}^{h} \rho_i.$$

On the other hand, we have

$$\|\mathrm{dag}(t')\| = \sum_{i=1}^{h} \rho_i.$$

The well-known inequality between the arithmetic and geometric mean states that for all $x_1, \ldots, x_m \in \mathbb{R}$,

$$\frac{1}{m} \cdot \sum_{i=1}^{m} x_i \geq \left(\prod_{i=1}^{m} x_i \right)^{1/m}.$$

Applying this to the numbers ρ_i ($1 \leq i \leq h$), we get

$$\|\mathrm{dag}(t')\| = \sum_{i=1}^{h} \rho_i \geq h \cdot \left(\prod_{i=1}^{h} \rho_i \right)^{1/h} \geq h \cdot \ell^{1/h}.$$

To further bound the term $h \cdot \ell^{1/h}$, we consider it as a function of h: let $f(x) = x \cdot \ell^{1/x}$. Its derivative is

$$f'(x) = \ell^{1/x} \left(1 - \frac{\ln(\ell)}{x} \right).$$

Therefore $f(x)$ has a minimum at $x = \ln \ell$ in the interval $(0, \infty)$, from which it follows that

$$h \cdot \ell^{1/h} \geq \ell^{1/\ln(\ell)} \cdot \ln \ell = e \cdot \ln \ell.$$

With $\ell \geq n/2$ we finally get

$$\|\mathrm{dag}(t)\| \geq \frac{1}{2} \cdot \|\mathrm{dag}(t')\| \geq \frac{e}{2} \cdot \ln \ell \geq \frac{e}{2} \cdot \ln(n/2)$$ ◀

For every tree t of size n we have $\|\mathrm{dag}(t)\| \leq n$. Moreover, by Theorem 4 it holds that $\|\mathrm{dag}^u(t)\| \geq \frac{e}{2} \cdot \ln(n/2)$. Hence, we obtain

$$\frac{\|\mathrm{dag}(t)\|}{\|\mathrm{dag}^u(t)\|} \leq \frac{2n}{e \cdot \ln(n/2)} \in \Theta(n/\log n),$$

which is stated in the next corollary.

▶ **Corollary 5.** *It holds that* $\alpha_E(n) \in O(n/\log n)$.

4.3 Upper Bound for α_N

In this section, we derive an upper bound on the node size of the minimal dag.

▶ **Theorem 6.** *For every tree* $t \in \mathcal{T}_{n,k}$ *of size* n *and height* h*, it holds that*[2]

$$|\mathrm{dag}(t)| \in O\left(\frac{n \cdot h \cdot \log(k+1)}{\log n}\right).$$

Proof. Let $t \in \mathcal{T}_{n,k}$ be a tree of size n and height h. Note that $|\mathrm{dag}(t)|$ is the number of different subtrees of t. Let t' be the tree that is obtained from t by removing all maximal subtrees of size at most

$$m := \frac{1}{2} \cdot \log_{4k} n = \frac{\log n}{2 \cdot \log 4k}.$$

Let F be the forest consisting of all these removed subtrees. Then the number of different subtrees of t (i.e., $|\mathrm{dag}(t)|$) is bounded by $|t'|$ plus the number of different subtrees in F. But the latter is bounded by the number of trees $s \in \mathcal{T}_{\infty,k}$ with $|s| \leq m$, which by [11, Lemma 2] is at most $\frac{4}{3}(4k)^m = \frac{4}{3}n^{1/2}$.

Let us now bound $|t'|$. Consider a leaf v of t'. Then, the subtree of t rooted in v must have size larger than m; otherwise v would not belong to t'. Therefore, t' has at most n/m many leaves. Clearly, if every internal node in t' would have at least two children, then we could conclude that t' has at most $2n/m$ many nodes. But t' may contain nodes with a single child. Let us call such nodes *unary*. Moreover, let ℓ be the length of a longest path in t' in which all nodes except the last one are unary. Then, we get $|t'| \leq 2(\ell+1)n/m \leq 2(h+1)n/m$. In total, we get

$$\begin{aligned} |\mathrm{dag}(t)| &\leq \frac{2(h+1)n}{m} + \frac{4}{3} \cdot n^{1/2} \\ &= \frac{4 \cdot (h+1) \cdot n \cdot \log 4k}{\log n} + \frac{4}{3} \cdot n^{1/2} \in O\left(\frac{n \cdot h \cdot \log(k+1)}{\log n}\right). \end{aligned}$$

◀

▶ **Corollary 7.** *It holds that* $\alpha_N(n,n,k) \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right)$ *and* $\alpha_N(n) \in O\left(\frac{n \cdot \log\log n}{\log n}\right)$.

Proof. Let us first show $\alpha_N(n,n,k) \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right)$. Let t be a tree of size n and height h with labels from $[k]$. By Theorem 6 we have

$$|\mathrm{dag}(t)| \in O\left(\frac{n \cdot h \cdot \log(k+1)}{\log n}\right).$$

[2] We write $\log(k+1)$ instead of $\log k$ in order to avoid $\log k = 0$ for $k = 1$.

On the other hand, we clearly have $|\mathrm{dag}^u(t)| \geq h$. Therefore, we get

$$\frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right).$$

Let us now prove $\alpha_N(n) \in O\left(\frac{n \cdot \log\log n}{\log n}\right)$. Consider an arbitrary tree t of size n with labels from $[n]$. If more than $\log n$ labels occur in t, then we clearly have $|\mathrm{dag}^u(t)| > \log n$. Since $|\mathrm{dag}(t)| \leq n$ we get (for n large enough)

$$\frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \leq \frac{n}{\log n} \leq \frac{n \cdot \log\log n}{\log n}.$$

On the other hand, if at most $\log n$ many different labels occur in t then the bound $\alpha_N(n, n, k) \in O\left(\frac{n \cdot \log(k+1)}{\log n}\right)$ implies

$$\frac{|\mathrm{dag}(t)|}{|\mathrm{dag}^u(t)|} \in O\left(\frac{n \cdot \log\log n}{\log n}\right).$$

This proves the bound $\alpha_N(n) \in O\left(\frac{n \cdot \log\log n}{\log n}\right)$. ◀

4.4 Summary of the Results for α_N and α_E

The following result summarizes our theoretical bounds for the functions $\alpha_N(n, 2, 1)$, $\alpha_N(n)$, $\alpha_E(n, 2, 1)$, and $\alpha_E(n)$:

▶ **Corollary 8.** *It holds that:*

$$\alpha_N(n, 2, 1) = \Theta\left(\frac{n}{\log n}\right), \qquad \alpha_N(n) = \Theta\left(\frac{n \cdot \log\log n}{\log n}\right),$$
$$\alpha_E(n, 2, 1) = \Theta\left(\frac{n}{\log n}\right), \qquad \alpha_E(n) = \Theta\left(\frac{n}{\log n}\right).$$

5 Experimental Results

In this section we experimentally evaluate the impact of unorderedness with regards to compression of XML trees. We compute for an XML tree t its *canonical tree* $\mathsf{canon}(t)$. The tree $\mathsf{canon}(f(t_1, t_2, \ldots, t_n))$ is obtained by sorting the trees $\mathsf{canon}(t_1), \ldots, \mathsf{canon}(t_n)$ according to their size, and in case of equal sizes, according to the lexicographical order of their XML traversal strings. Clearly, for all trees s, t we have $s^u = t^u$ if and only if $\mathsf{canon}(s) = \mathsf{canon}(t)$. The minimal unordered dag can be obtained from the canonical tree by computing its minimal dag. Clearly, this is a very expensive way of obtaining the minimal unordered dag: it can be obtained in linear time by a variant of the algorithm of Aho, Hopcroft and Ullman [2], see also [19]. We have implemented that procedure and can confirm its efficiency: it runs at least as fast as our (ordered) dag programs based on hashing. The reason for computing the canonical tree is to provide a simple way of gauging how other compressors benefit from unorderedness (by imposing a canonical order). It should be understood that our experiment only gives a rough indication of the benefit of unorderedness for compressors other than the dag. We expect that a more careful adaptation of those compressors to unordered trees will provide stronger compression.

We only report number of edges, so "size" in this section always refers to number of edges.

5.1 Tree Compressors

We compare seven known tree compressors, which are considered in [5]:

1. minimal dag,
2. minimal binary dag,
3. minimal reverse binary dag,
4. minimal hybrid dag,
5. minimal reverse hybrid dag,
6. DS, and
7. TreeRePair.

We choose these compressors, since they all produce a graph-based representation of the input tree. This makes the output sizes of the compressors comparable.

It should be clear that the minimal dag of the canonical tree is isomorphic to the unordered dag of the original tree. Thus, the size of the minimal dag of a canonical tree is always smaller than or equal to the size of the minimal dag of the original tree.

The *minimal binary dag (bdag)* of an unranked tree t is the minimal dag of the "first-child/next-sibling encoding" (for short, fcns encoding) of t. The fcns encoding s is common for XML: it has the same nodes as t, a node v is the left child in s of a node u if and only if v is the first child of u in t, and, a node v is the right child in s of a node u if and only if v is the next sibling of u in t. This encoding is considered in Paragraph 2.3.2 of Knuth's first book [12]. The *minimal reverse binary dag (rbdag)* is the minimal dag of the "first-child/previous-sibling encoding" (fcps), defined in the obvious way. Binary dags and reverse binary dags share end- and begin-sequences, respectively, of subtrees. This implies that both the bdag and rbdag of a canonical tree can be *larger* than the corresponding dags of the original tree. As an example consider the following tree

$$t = f(g(c, d, b, a, h), g(c, d, b), g(b, d, c, d, b)).$$

This tree has 16 edges. Its minimal binary dag has only 14 edges, because the end-sequence of subtrees "c, d, b" occurs twice and can be shared. Similarly, the minimal reverse binary dag has size 14 (because "c, d, b" appears twice). In contrast, the canonical tree of t

$$\mathsf{canon}(t) = f(g(b, c, d), g(a, b, c, d, h), g(b, b, c, d, d))$$

has a bdag and rbdag of 16 edges. Interestingly, such scenarios where bdag and rbdag become larger for the canonical tree appear frequently in practice.

The *hybrid dag (hdag)* (and *reverse hybrid dag (rhdag)*) were introduced in [5] as data structures that are guaranteed to be smaller than or equal in size to both the dag and the bdag (rbdag) of an unranked tree. The hdag (resp., rhdag) is obtained from a dag by applying the fcns-encoding (resp., fcps-encoding) to the rules of the dag (where the dag is viewed as a regular tree grammar), and then computing the minimal dag of the resulting forest of encoded rules; see [5] for a precise definition. Similarly as with bdag and rbdag, the hdag and rhdag can be *larger* for the canonical tree than for the original one.

The acronym *DS* stands for "dag and string compression". The idea is to compute a minimal dag and to then apply a string compressor to the above mentioned rules of the dag. As in [5], we use RePair [13] as our string compressor. Finally, *TR* refers to the grammar-based tree compressor TreeRePair of [15]. The sizes are numbers of edges of the compressed structures, see [5] for details.

Table 1 Document characteristics, Edges = average number of edges in a tree, aD = average depth of a node, mD = maximum depth of a node in any tree, aR = average rank of a node, mR = maximum rank of a tree.

Corpus	Documents	Edges	aD	mD	aR	mR
I	21	$3.1 \cdot 10^6$	6.6	36	5.7	$3.9 \cdot 10^6$
II	1131	79465	7.9	65	6.0	2925

5.2 Corpora of XML Documents

We use two different Corpora of XML documents. These corpora were also used in [5]. For each document we consider the unranked tree of its element nodes, i.e., we ignore attribute and text values. *Corpus I* consists of XML documents from the web which are often used in XML compression research. Many of the files of this corpus can be downloaded from the XMLCompBench site[3] (see [5] for details). *Corpus II* is a subset of files from the *University of Amsterdam XML Web Collection*[4]. We have verified by hand that, according to the tag names, all of the documents in Corpus I appear to be order independent. By sampling Corpus II we also did not find order dependent documents.

The characteristics of the Corpora are quite different: Corpus I consists of few and very large files while Corpus II has many small files. Some characteristics are shown in Table 1. As can be seen, the average size of documents from Corpus I is about 40 times larger than that of Corpus II, and the rank (=maximum length of sibling lists) of documents from Corpus I is about 1300 times larger; this indicates that most of the documents from Corpus I are indeed very long lists of (small) subtrees.

5.3 Experimental Setup

The implementations for dag, bdag, rbdag, hdag, and DS are the same ones as used in [5]. Note that DS uses Gonzalo Navarro's implementation of RePair for strings[5]. For TreeRePair, called "TR" in what follows, we use Roy Mennicke's implementation[6]; we do not change any parameters and run it plain from the command line (thus, the maxRank parameter of TR is at its default value of 4). We do not report running times (they are provided in [5]). The canonizer was implemented from scratch in java using integer and string sorting as provided by java (this runs quite slow and can take several hours for some of the documents).

5.4 Compression of Canonical versus Original Tree

The results of applying the different compressors to the documents of Corpus I are shown in Table 2. The first line shows how the compression ratio on the canonical tree changes with respect to the compression ratio for the original tree (a percentage of more than 100% means that the compression ratio is better on the canonical tree). The second row shows the sizes of the compressed canonical trees (in number of edges). For instance, the compression ratio of the hdag of the canonical tree of document "sprot39.dat" is 67% of the ratio for the original tree. On the other hand, DS compressor over the canonical tree of document

[3] `http://xmlcompbench.sourceforge.net`
[4] `http://data.politicalmashup.nl/xmlweb`
[5] `http://www.dcc.uchile.cl/~gnavarro/software/`
[6] `http://code.google.com/p/treerepair`

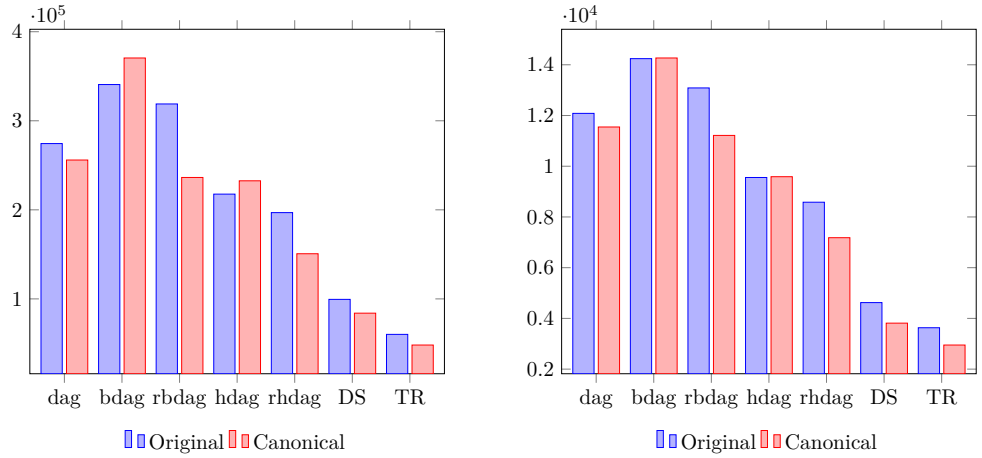

Figure 4 Comparison of average sizes of Corpus I (left) and Corpus II (right).

"EnWikTionary" has a compression that is 191-times better than the ratio for the original tree. For each document we indicate in bold the unique best increase of compression, and underline the smallest size.

Note that the dag of the canonical tree can never be larger than the dag of the original tree. Intuitively, every original repeating subtree (that gets shared in the dag) is also repeating in the canonical tree. Thus, there cannot be percentages below 100 in the column for the dag. In every other column the percent number can potentially be below 100. This is because these compressors take into account sibling sequences and hence are effected by the change of sibling orders due to canonization. In fact, this happens for the file "EXI-factbook": here *all* compression ratios (except that for the dag) become worse for the canonical tree. It means the the ordering of the canonization removes repetitions that are meaningful for the compressor. It is interesting to see that for this outlier, the strongest overall compressor TR (with respect to size) is affected the most: the compression goes down to 81% of the original; this is also the only file where this ever happens for TR. Another outlier that comes from the EXI group is EXI-weblog, where no compression ratio changes; this document is in an order that is isomorphic to that of the canonical tree.

The majority (almost one half) of documents have the strongest increase for the DS compressor. In particular, all the EnWik documents belong to this group. It is interesting to observe that for all the EnWik documents *only* DS and TR give (*massive*) compression, while for all the dag variants the compression ratio does not change. This means that after canonization there are (i) no repeating subtrees and (ii) no repeating prefixes or suffixes of sibling lists that were different before canonization. Note also that for this group,

DS achieves the smallest size values for each document. It means that there are no complex tree patterns that are repeating, and hence would be compressed by TR but not by DS; all repetition seems to be purely on the level of sibling lists. In contrast to that, observe the treebank file which features (by far) the most complex tree structure of all the documents: here the size of TR is almost twice smaller than that of DS. Curiously, the rhdag has the highest increase for this document. There is another interesting group of documents, namely those where the rbdag has the largest increase. It means that after canonization there are a lot of repeating prefixes of sibling sequences; thus, optional elements which typically appear at the end of sibling lists (the reverse dags profit from that) have, after canonization, remained to appear at the end. Apparently, this is less often the case for

Table 2 Difference (in %) of canonical versus original tree compression, and size of canonical compression output (largest in bold and smallest underlined, respectively).

document	dag	bdag	rbdag	hdag	rhdag	DS	TR
1998statistics	118%	352%	**373%**	306%	333%	239%	211%
	1164	682	632	422	373	200	238
catalog-01	146%	82%	177%	84%	**182%**	146%	117%
	5856	8514	5830	5302	3295	2994	3390
catalog-02	114%	98%	111%	98%	113%	**473%**	331%
	28496	53647	50858	27912	25823	5761	8072
dictionary-01	107%	102%	**225%**	104%	187%	130%	136%
	54575	75827	33386	45214	24960	24634	16434
dictionary-02	116%	116%	**254%**	117%	207%	138%	145%
	469915	588665	257228	353365	197197	194324	115932
EnWikiNew	100%	100%	100%	100%	100%	**2712%**	2282%
	35075	70018	70025	35057	35054	341	422
EnWikiQuote	100%	100%	100%	100%	100%	**2370%**	2091%
	23904	47692	47699	23888	23887	267	316
EnWikiVersity	100%	100%	100%	100%	100%	**2672%**	2287%
	43693	87258	87263	43676	43673	264	326
EnWikTionary	100%	100%	100%	100%	100%	**19108%**	15839%
	726221	1452273	1452279	726197	726191	428	531
EXI-Array	100%	100%	100%	100%	100%	**425%**	375%
	95584	128009	128011	95562	95563	213	267
EXI-factbook	100%	100%	91%	96%	96%	93%	81%
	4477	5090	3227	3766	2225	1937	1708
EXI-Invoice	100%	100%	100%	100%	100%	98%	**102%**
	1073	2073	2067	1071	1066	98	106
EXI-Telecomp	100%	100%	100%	100%	100%	99%	**102%**
	9933	19807	19808	9932	9931	111	137
EXI-weblog	100%	100%	100%	100%	100%	100%	100%
	8504	16997	16997	8504	8504	44	58
JST_gene.chr1	100%	99%	100%	99%	100%	**430%**	396%
	9176	14718	14103	7840	7206	917	1062
JST_snp.chr1	100%	98%	101%	97%	101%	**382%**	347%
	23509	41444	37425	22805	19111	2571	2980
medline	165%	150%	**240%**	141%	222%	145%	141%
	395754	493136	158984	326638	113932	122270	88109
NCBI_gene.chr1	100%	93%	110%	91%	116%	**148%**	137%
	16038	15504	9839	11606	5912	4237	3764
NCBI_snp.chr1	100%	100%	100%	100%	100%	100%	100%
	404704	809394	809394	404704	404704	61	83
sprot39.dat	102%	60%	**269%**	67%	237%	102%	102%
	1724689	2394532	586523	1484814	376067	328469	257376
treebank	101%	99%	106%	99%	**107%**	104%	103%
	1292198	1455300	1171666	1246195	1039933	1073301	510683

the reverse hybrid dag, i.e., after building the dag there is less profit from canonization. An interesting document that has always been challenging with respect to compression [6] is medline: with 165% it has the largest increase within the dag column. This means that many permutations of the same subtree sequences exist. This could be because these bibliography entries have been entered manually by different persons, each having their own preferences of ordering sibling lists. Observe also that every single compressor has an increase of at least 140% for the medline document. Similar to this is the 1998statistics document: here the dag only increases by 118%, but all others increase by 210% or more. Thus, there are not many subtrees with precisely the same subtrees (possibly in different orders), but, there is a large number of repetitions of subsequences of sibling lists, in particular of prefix subsequences (viz. the highest increases of rbdag and bdag).

In summary, Figure 4 shows the average sizes for the different compressors for Corpus I and Corpus II, respectively. For Corpus I, all compressors, except bdag (91%) and hdag (93%), show an improvement of the compression ratio. DS (118%) and TR (124%), which already give very high compression ratios, also have high increases. The biggest increases, however, are seen for rbdag (134%) and rhdag (130%). For Corpus II, we see that there is almost no difference in the case of bdag and hdag. Again, DS (121%) and TR (123%) improve on their already high compression ratios, while rbdag (117%) and rhdag (120%) achieve improvements as well.

Finally, we also tried a different canonizer: It assigns to every subtree t a number $i(t)$ such that for every pair of subtrees t_1, t_2 it holds that $i(t_1) = i(t_2)$ if and only if the unordered trees of t_1 and t_2 are equal. The children $t_1, \ldots, t_n$ of a subtree t are then sorted with respect to $i(t_1), \ldots, i(t_n)$. While this algorithm runs a lot faster than sorting the whole subtrees, the compression ratios only change very slightly.

6 Conclusions

In this paper we showed that the minimal *unordered dag* of a tree can be exponentially smaller than the minimal (ordered) dag of that tree. Furthermore, we proved that this difference is exponential at most, thus providing matching upper and lower bounds for the ratio of the ordered and unordered dag size of a tree. These results hold for both size measures: number of nodes and the number of multi-edges. It would be interesting in the future to investigate also the number of "collapsed" edges, where multi-edges between the same pair of nodes are collapsed to a single edge. For the unordered dag, this size measure makes sense, since one only needs to store the number of multi-edges between two nodes and these numbers can be stored succinctly in binary notation. Another interesting theoretical research problem is to compute the average size of the unordered dag, where the average is taken with respect to the uniform distribution on all trees of size n. The corresponding average for ordered dags was analysed in [10] and the asymptotic growth rate $\Theta(n/\sqrt{\log n})$ was derived using tools from analytic combinatorics, see also [5]. We conjecture that the average unordered dag size has the same asymptotic growth. Related to this, one might study the average values of the ratios α_N and α_E for which we only derived worst case bounds.

In the second part of this paper, we have experimentally evaluated the difference of sibling orders for different compressors. Within the compressors that are based on dags, the biggest impact of ignoring order is observed for the reverse binary dag (thus, the minimal dag of the first-child/*previous-sibling*-encoded tree); the biggest difference is a better compression in the unordered setting by a factor of 3.7. Within the RePair-variants DS and TreeRePair, DS features the larger difference, with a factor of 191 for one document. Such a massive

improvement due to ignoring document order could be useful in practice: it could mean that the entire tree structure of a document that is Terabytes large, can be conveniently stored (and queried) in main memory, while the data values would be stored on secondary memory.

It would be interesting to consider trees with ordered and unordered nodes. For XML documents this can be done via XML Schema Definition. JSON very naturally provides this possibility: its two primitives are lists (= ordered) and sets of key-value pairs (= unordered). We expect that canonization of unordered nodes will improve compression. In the future, we would like to find also a method that combines features of the compressors presented here, and is guaranteed to compress equally well or better than each of the the compressors.

References

1 S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. *Theory of Computing Systems*, 57(4):927–966, 2015.

2 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

3 A. Boiret, V. Hugot, J. Niehren, and R. Treinen. Logics for unordered trees with data constraints on siblings. In *Proceedings of LATA 2015*, volume 8977 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2015.

4 I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theory of Computing Systems*, 57(2):337–376, 2015.

5 M. Bousquet-Mélou, M. Lohrey, S. Maneth, and E. Noeth. XML compression via directed acyclic graphs. *Theory of Computing Systems*, 57(4):1322–1371, 2015.

6 P. Buneman. Private Communication, 2005.

7 P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proceedings of VLDB 2003*, pages 141–152. Morgan Kaufmann, 2003.

8 S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Proceedings of KGC 1997*, volume 1289 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 1997.

9 P. J. Downey, R. Sethi, and R. Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, 1980.

10 P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Proceedings of ICALP 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990.

11 M. Ganardi, D. Hucke, A. Jeż, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. Technical report, arXiv.org, 2014. `http://arxiv.org/abs/1407.4286`.

12 D. E. Knuth. *The Art of Computer Programming, Vol. I: Fundamental Algorithms*. Addison-Wesley, 1968.

13 N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proceedings of DCC 1999*, pages 296–305. IEEE Computer Society, 1999.

14 H. Liefke and D. Suciu. XMILL: an efficient compressor for XML data. In *Proceedings of ACM SIGMOD Conference 2000*, pages 153–164. ACM, 2000.

15 M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.

16 F. Neven and T. Schwentick. XML schemas without order. Unpublished manuscript, 1999.

17 J. M. Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. MAA Problem Books Series. Cambridge University Press, 2004.

18 S. Sundaram and S. Kumar Madria. A change detection system for unordered XML data using a relational model. *Data & Knowledge Engineering*, 72:257–284, 2012.

19 Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.
20 S. Zhang, Z. Du, and J. Tsong-Li Wang. New techniques for mining frequent patterns in unordered trees. *IEEE Transactions on Cybernetics*, 45(6):1113–1125, 2015.