Slowing Down Top Trees for Better Worst-Case Compression

Bartłomiej Dudek
Institute of Computer Science, University of Wrocław, Poland
bartlomiej.dudek@cs.uni.wroc.pl

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Abstract

We consider the top tree compression scheme introduced by Bille et al. [ICALP 2013] and construct an infinite family of trees on n nodes labeled from an alphabet of size σ, for which the size of the top DAG is $\Theta(\frac{n}{\log_\sigma n}\log\log_\sigma n)$. Our construction matches a previously known upper bound and exhibits a weakness of this scheme, as the information-theoretic lower bound is $\Omega(\frac{n}{\log_\sigma n})$. This settles an open problem stated by Lohrey et al. [arXiv 2017], who designed a more involved version achieving the lower bound. We show that this can be also guaranteed by a very minor modification of the original scheme: informally, one only needs to ensure that different parts of the tree are not compressed too quickly. Arguably, our version is more uniform, and in particular, the compression procedure is oblivious to the value of σ.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Data compression

Keywords and phrases top trees, compression, tree grammars

Digital Object Identifier 10.4230/LIPIcs.CPM.2018.16

Funding Work supported under National Science Centre, Poland, project number 2014/15/B/ST6/00615.

1 Introduction

Labeled trees are fundamental data structures in computer science. Generalizing strings, they can be used to compactly represent hierarchical dependencies between objects and have multiple applications. In many of them, such as XML files, we need to operate on very large trees that are in some sense repetitive. Therefore, it is desirable to design compression schemes for trees that are able to exploit this. Known tree compression methods include DAG compression that uses subtree repeats and represents a tree as a Directed Acyclic Graph [3, 7, 13], compression with tree grammars that focuses on the more general tree patterns and represents a tree by a tree grammar [4, 8, 11, 12], and finally succinct data structures [6, 10].

In this paper we analyze tree compression with top trees introduced by Bille et al. [2]. It is able to take advantage of internal repeats in a tree while supporting various navigational queries directly on the compressed representation in logarithmic time. At a high level, the idea is to hierarchically partition the tree into *clusters* containing at most two boundary nodes that are shared between different clusters. A representation of this hierarchical partition is called the top tree. Then, the top DAG is obtained by identifying isomorphic subtrees of the top tree. Bille et al. [2] proved that the size of the top DAG is always $O(n/\log_\sigma^{0.19} n)$ for a tree on n nodes labeled with labels from Σ where $\sigma = \max\{2, |\Sigma|\}$. Furthermore, they showed that top DAG compression is always at most logarithmically worse than the classical DAG compression (and Bille et al. [1] constructed a family of trees for which this logarithmic upper bound is tight). Later, Hübschle-Schneider and Raman [9] improved the bound on the size of the top DAG to $O(\frac{n}{\log_\sigma n}\log\log_\sigma n)$ using a more involved reasoning based on the heavy path decomposition. This should be compared with the information-theoretic lower bound of $\Omega(\frac{n}{\log_\sigma n})$.

A natural question is to close the gap between the information-theoretic lower bound of $\Omega(\frac{n}{\log_\sigma n})$ and the upper bound of $O(\frac{n}{\log_\sigma n}\log\log_\sigma n)$. We show that the latter is tight for the top tree construction algorithm of Bille et al. [2].

▶ **Theorem 1.** *There exists an infinite family of trees on n nodes labeled from an alphabet Σ for which the size of the top DAG is $\Omega(\frac{n}{\log_\sigma n}\log\log_\sigma n)$ where $\sigma = \max\{2, |\Sigma|\}$.*

This answers an open question explicitly mentioned by Lohrey et al. [14], who developed a different algorithm for constructing a top tree which guarantees that the size of the top DAG matches the information-theoretic lower bound. A crucial ingredient of their algorithm is a partition of the tree T into $O(n/k)$ clusters of size at most k, where $k = \Theta(\log_\sigma n)$. As a byproduct, they obtain a top tree of depth $O(\log n)$ for each cluster. Then they consider a tree T' obtained by collapsing every cluster of T and run the algorithm of Bille et al. [2] on T'. Finally, the edges of T' are replaced by the top trees of their corresponding clusters of T constructed in the first phase of the algorithm to obtain the top tree of the whole T. While this method guarantees that the number of distinct clusters is $O(\frac{n}{\log_\sigma n})$, its disadvantage is that the resulting procedure is non-uniform, and in particular needs to be aware of the value of σ and n.

We show that a slight modification of the algorithm of Bille et al. [2] is, in fact, enough to guarantee that the number of distinct clusters, and so also the size of the top DAG, matches the information-theoretic lower bound. The key insight actually comes from the proof of Theorem 1, where we construct a tree with the property that some of its parts are compressed much faster than the others, resulting in a larger number of different clusters. The original algorithm proceeds in iterations, and in every iteration tries to merge adjacent clusters as long as they meet some additional conditions. Surprisingly, it turns out that the information-theoretic lower bound can be achieved by slowing down this process to avoid some parts of the tree being compressed much faster than the others. Informally, we show that it is enough to require that in the t^{th} iteration adjacent clusters are merged only if their size is at most α^t, for some constant $\alpha > 1$. The modified algorithm preserves nice properties of the original method such as the $O(\log n)$ depth of the obtained top tree.

A detailed description of the original algorithm of Bille et al. [2] can be found in Section 2. In Section 3 we prove Theorem 1 and in Section 4 describe the modification.

2 Preliminaries

In this section, we briefly restate the top tree construction algorithm of Bille et al. [2]. To construct trees that can be used to show the lower bound and present our modification of the original algorithm we need to work with exactly the same definitions. Consequently, the following description closely follows the condensed presentation from Bille et al. [1] and can be omitted if the reader is already familiar with the approach.

Let T be a (rooted) tree on n nodes. The children of every node are ordered from left to right, and every node has a label from an alphabet Σ. $T(v)$ denotes the subtree of v, including v itself, and $F(v)$ is the forest of subtrees of all children $v_1, v_2, \ldots, v_k$ of v, that is,

$F(v) = T(v_1) \cup T(v_2) \cup \ldots \cup T(v_k)$. For $1 \le s \le r \le k$ we define $T(v, v_s, v_r)$ to be the tree consisting of v and a contiguous range of its children starting from the s^{th} and ending at the r^{th}, that is, $T(v, v_s, v_r) = \{v\} \cup T(v_s) \cup T(v_{s+1}) \cup \ldots \cup T(v_r)$.

We define two types of *clusters*. A cluster with only a top boundary node v is of the form $T(v, v_s, v_r)$. A cluster with a top boundary node v and a bottom boundary node u is of the form $T(v, v_s, v_r) \setminus F(u)$ for a node $u \in T(v, v_s, v_r) \setminus \{v\}$.

If edge-disjoint clusters A and B have exactly one common boundary node and $C = A \cup B$ is a cluster, then A and B can be *merged* into C. Then one of the top boundary nodes of A and B becomes the top boundary node of C and there are various ways of choosing the bottom boundary node of C. See Figure 2 in [2] for the details of all five possible ways of merging two clusters.

A *top tree* $\mathcal{T}$ of T is an ordered and labeled binary tree describing a hierarchical decomposition of T into clusters.

- The nodes of $\mathcal{T}$ correspond to the clusters of T.
- The root of $\mathcal{T}$ corresponds to the whole T.
- The leaves of $\mathcal{T}$ correspond to the edges of T. The label of each leaf is the pair of labels of the endpoints of its corresponding edge (u, v) in T. The two labels are ordered so that the label of the parent appears before the label of the child.
- Each internal node of $\mathcal{T}$ corresponds to the merged cluster of the clusters corresponding to its two children. The label of each internal node is the type of merge it represents (out of the five merging options). The children are ordered so that the left child is the child cluster visited first in a preorder traversal of T.

The top tree $\mathcal{T}$ is constructed bottom-up in iterations, starting with the edges of T as the leaves of $\mathcal{T}$. During the whole process, we maintain an auxiliary ordered tree $\widetilde{T}$, initially set to T. The edges of $\widetilde{T}$ correspond to the nodes of $\mathcal{T}$, which in turn correspond to the clusters of T. The internal nodes of $\widetilde{T}$ correspond to the boundary nodes of these clusters and the leaves of $\widetilde{T}$ correspond to a subset of the leaves of T.

On a high level, the iterations are designed in such a way that each of them merges a constant fraction of edges of $\widetilde{T}$. This is proved in Lemma 1 of [2], and we describe a slightly stronger property in Lemma 2. This guarantees that the height of the resulting top tree is $O(\log n)$. Each iteration consists of two steps:

Horizontal merges. For each node $v \in \widetilde{T}$ with $k \ge 2$ children $v_1, \ldots, v_k$, for $i = 1$ to $\lfloor \frac{k}{2} \rfloor$, merge the edges (v, v_{2i-1}) and (v, v_{2i}) if v_{2i-1} or v_{2i} is a leaf. If k is odd and v_k is a leaf and both v_{k-2} and v_{k-1} are non-leaves then also merge (v, v_{k-1}) and (v, v_k).

Vertical merges. For each maximal path $v_1, \ldots, v_p$ of nodes in $\widetilde{T}$ such that v_{i+1} is the parent of v_i and $v_2, \ldots, v_{p-1}$ have a single child: If p is even merge the following pairs of edges $\{(v_1, v_2), (v_2, v_3)\}, \ldots, \{(v_{p-3}, v_{p-2}), (v_{p-2}, v_{p-1})\}$. If p is odd merge the following pairs of edges $\{(v_1, v_2), (v_2, v_3)\}, \ldots, \{(v_{p-4}, v_{p-3}), (v_{p-3}, v_{p-2})\}$, and if (v_{p-1}, v_p) was not merged in the previous step then also merge $\{(v_{p-2}, v_{p-1}), (v_{p-1}, v_p)\}$.

See an example of a single iteration in Figure 1. Finally, the compressed representation of T is the so-called top DAG $\mathcal{TD}$, which is the minimal DAG representation of $\mathcal{T}$ obtained by identifying identical subtrees of $\mathcal{T}$. As every iteration shrinks $\widetilde{T}$ by a constant factor, $\mathcal{T}$ can be computed in $O(n)$ time, and then $\mathcal{TD}$ can be computed in $O(|\mathcal{T}|)$ time [5]. Thus, the entire compression takes $O(n)$ time.

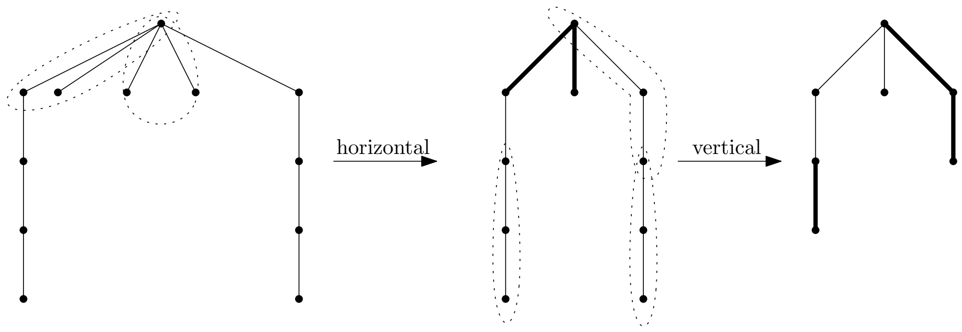

Figure 1 Tree $\widetilde{T}$ after two steps of a single iteration. Dotted lines denote the merged edges (clusters) and thick edges denote the results of merging. Note that one edge does not participate in the vertical merge due to having been obtained as a result of a horizontal merge.

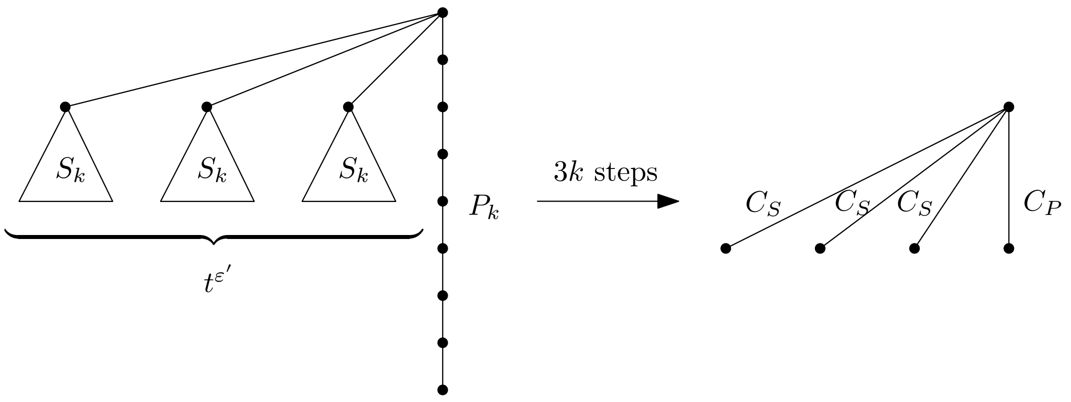

Figure 2 Gadget G_k consists of $2^k - 1 = O(t^{\varepsilon'})$ trees S_k and one path P_k. After $3k$ iterations it gets compressed to a tree with 2^k nodes connected to the root.

3 A lower bound for the approach of Bille et al.

In this section, we prove Theorem 1 and show that the $O(\frac{n}{\log_\sigma n} \log\log_\sigma n)$ bound from [9] on the number of distinct clusters created by the algorithm described in Section 2 is tight. We first consider labeled trees for which $|\Sigma| > 1$ and $\sigma = |\Sigma|$. Then we show how to modify our construction and apply it to unlabeled trees.

For every $k \in \mathbb{N}$ we will construct a tree T_k with $n = \Theta(\sigma^{8^k})$ nodes for which the corresponding top DAG is of size $\Theta(\frac{n}{\log_\sigma n} \log\log_\sigma n)$. Let $t = 8^k = \Theta(\log_\sigma n)$. In the beginning, we describe a gadget G_k that is the main building block of T_k. It consists of $O(t)$ nodes: a path of t nodes and $2^k - 1 = O(t^{\varepsilon'})$ full ternary trees of size $O(t^\varepsilon)$ connected to the root, where $\varepsilon + \varepsilon' < 1$. See Figure 2. The main intuition behind the construction is that full ternary trees are significantly smaller than the path, but they need the same number of iterations to get compressed.

More precisely, let P_k be the path of length $8^k = t$. Clearly, after 3 iterations it gets compressed to P_{k-1}, and so after $3k$ iterations becomes a single cluster. Similarly, let S_k be the full ternary tree of height k with 3^k leaves, so $\frac{3^{k+1}-1}{2} = O(3^k) = O(t^{0.53})$ nodes in total. Observe that after 3 iterations S_k becomes S_{k-1}, and so after $3k$ iterations becomes a single cluster. To sum up, the gadget G_k consists of path P_k of t nodes and $2^k - 1 = O(t^{1/3})$ trees of size $O(t^{0.53})$, so in total $O(t)$ nodes. After $3k$ iterations G_k consists of $2^k - 1$ clusters C_S

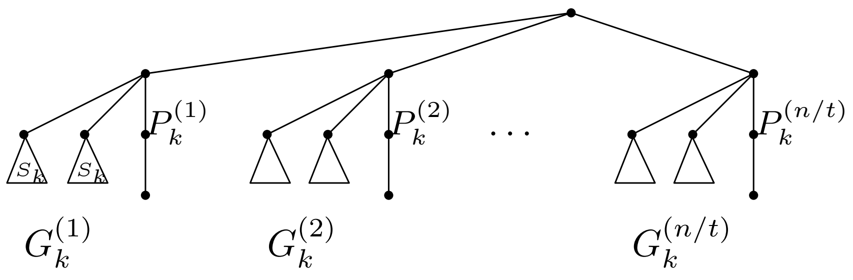

Figure 3 T_k consists of $\Theta(n/t)$ gadgets $G_k^{(i)}$, where the i^{th} of them contains a unique path $P_k^{(i)}$.

corresponding to S_k and one cluster C_P corresponding to P_k, as shown in Figure 2. In each of the subsequent k iterations, the remaining clusters are merged in pairs.

Recall that the top DAG contains a node for every distinct subtree of the top tree, and every node of the top tree corresponds to a cluster obtained during the compression process. Our next step will be to create many almost identical gadgets connected to a common root. In order to ensure that they are distinct, we assign labels on the nodes on paths P_k so that no two paths are equal. Then the cluster C_P obtained after the first $3k$ iterations corresponds to a distinct subtree of the top tree. Consequently, so does the cluster obtained from C_P in each of the subsequent k iterations. Note that during all the subsequent iterations only horizontal merges are performed and each of them halves the number of clusters.

Finally, the tree T_k consists of $\Theta(n/t)$ gadgets connected to a common root as in Figure 3. The i^{th} gadget $G_k^{(i)}$ is a copy of G_k with the labels of $P_k^{(i)}$ chosen as to spell out the i^{th} (in the lexicographical order) word of length t over Σ. For all the remaining nodes (nodes of trees S_k, roots of gadgets and the common root of T_k) it is enough to choose the same label, e.g. the smallest in Σ. Note that $\sigma^t > n/t$, so there are more possible words of length t than the number of gadgets that we want to create. Then each $C_P^{(i)}$ and the clusters obtained from it during the subsequent k iterations correspond to distinct subtrees of the top tree. Thus, overall the top DAG contains $\Omega(n/t \cdot k) = \Omega(n/t \cdot \log t) = \Omega(n/\log_\sigma n \cdot \log\log_\sigma n)$ nodes, which concludes the proof of Theorem 1 for labeled trees with non-unary alphabet.

Unlabeled trees. Now we modify the above construction so that it works for unary alphabets. Recall that we set $t = \log n$ and $k = \log_8 t$. We cannot use the earlier approach directly, as we cannot distinguish the gadgets $G_k^{(i)}$ by modifying labels on the path $P_k^{(i)}$. To address this we extend the gadgets $G_k^{(i)}$ with distinct unlabeled binary trees in such a way that after $3k$ steps the new gadgets get compressed to the same trees as before (shown in Figure 2) that is the i^{th} gadget is compressed to $t^{\varepsilon'}$ clusters C_S and a cluster $C_P'^{(i)}$. Again C_S represents the cluster of full ternary tree S_k and clusters $C_P'^{(i)}$ correspond to distinct subtrees.

More precisely, now the i^{th} gadget $G_k'^{(i)}$ consists of $t^{\varepsilon'}$ trees S_k connected to the root, a path P_k' of length $4 \cdot 8^{k-1} + 1$ and the i^{th} binary tree $T_k^{(i)}$ on $t = \log n$ nodes (we consider an arbitrary ordering on all such trees). Intuitively, the construction of path P_k' guarantees that no matter how fast the tree $T_k^{(i)}$ gets compressed, during the first $3k$ steps it does not interact with subtrees S_k. Without the path P_k', it might happen that the single cluster obtained from $T_k^{(i)}$ participates in a horizontal step with the (partially compressed) rightmost tree S_k within the first $3k$ steps. Next, the sizes of each component of $G_k'^{(i)}$ are chosen in such a way that again $G_k'^{(i)}$ consists of $O(t)$ nodes and after $3k$ steps the obtained tree is exactly the same as in the case of non-unary alphabets. See Figure 4.

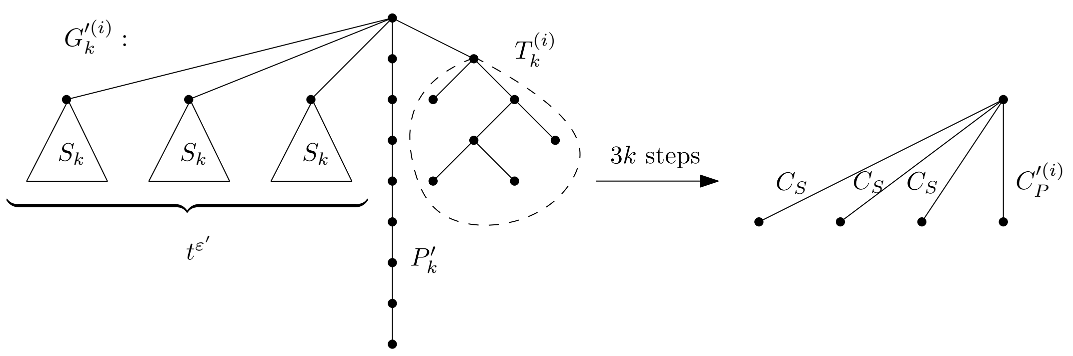

Figure 4 The modified gadget $G_k'^{(i)}$ for unlabeled trees.

Note that path P_k' gets compressed to path P_{k-1}' in 3 steps. Furthermore, the first edge of P_k' does not take part in any vertical merge unless the path consists of only two edges, that is in the $(3k)^{\text{th}}$ step. Observe that trees $T_k^{(i)}$ are compressed with different speeds depending on their shape and at some moment they become a single cluster that will be merged in the next horizontal step. As pointed earlier, the first edge of P_k' does not take part in vertical merges before the $(3k)^{\text{th}}$ step, so eventually it can participate in a horizontal merge with the cluster of $T_k^{(i)}$ without affecting the compression of the remaining edges of P_k'. As all the merges inside $T_k^{(i)}$ are independent from the rest of the tree, Lemma 1 of [2] guarantees that after every step of the compression the tree $T_k^{(i)}$ shrinks at least by a factor of $8/7$. Thus $T_k^{(i)}$ becomes a single cluster in at most $\log_{8/7}\log n \ < 9\log\log n = 3k$ steps, and so after $3k$ steps the gadget $G_k'^{(i)}$ gets compressed to the tree described in Figure 4.

Finally, in order to further apply the reasoning from the case of labeled trees, it remains to show that there are $\Omega(n/t)$ distinct binary trees on t nodes. From the folklore properties of Catalan numbers, there are $\frac{1}{t+1}\binom{2t}{t}$ distinct binary trees on t nodes. Applying the bound $\binom{n}{k} \geq (\frac{n}{k})^k$ we obtain that there are at least $\frac{2^t}{t+1} = \Omega(n/t)$ distinct binary trees $T_k^{(i)}$, which is sufficient for our construction. It concludes the case of unlabeled trees and thus ends the proof of Theorem 1.

4 An optimal tree compression algorithm

Let α be a constant greater than 1 and consider the following modification of algorithm [2]. Our algorithm works in the same way for both labeled and unlabeled trees. As mentioned in the introduction, intuitively we would like to proceed exactly as the original algorithm, except that in the t^{th} iteration we do not perform a merge if one of the participating clusters is of size larger than α^t. However, this would require a slight modification of the original charging argument showing that that after every iteration the tree $\widetilde{T}$ shrinks by a constant factor. To avoid adapting the whole proof of [2] to our new approach, we proceed slightly differently. In each iteration we first generate and list all the merges that would have been performed in both steps of a single iteration of the original algorithm. Then we apply only the merges in which both clusters have size at most α^t.

We run the algorithm until the tree $\widetilde{T}$ becomes a single edge. Clearly, there are $O(\log n)$ iterations, because after $\log_\alpha n$ iterations the algorithm is no longer constrained and can behave not worse than the original one. Thus the depth of the obtained DAG is $O(\log n)$ as before. In the following lemma we show that even if there are some clusters that cannot be

Algorithm 1 A modified top tree construction algorithm of Bille et al. [2] for a tree T.

```
1: $\widetilde{T} := T$
2: initialize leaves of $\mathcal{T}$ with edges of $T$
3: for $t = 1, \ldots, \Theta(\log n)$, as long as $\widetilde{T}$ is not a single edge do
4:     list all the merges that would have been made by one iteration of the original algorithm
5:     filter out the merges that use a cluster of size bigger than $\alpha^t$
6:     modify $\widetilde{T}$ and $\mathcal{T}$ by applying the remaining merges
7: construct DAG $\mathcal{TD}$ of $\mathcal{T}$                    ▷ $\mathcal{TD}$ is the top DAG of $T$
```

merged in one step, the tree still shrinks by roughly a constant factor.

▶ **Lemma 2.** *Suppose that there are $m = p + q$ clusters in $\widetilde{T}$ after $t - 1$ iterations of Algorithm 1, where q is the number of clusters of size larger than α^t. Then, after t iterations there are at most $7/8m + q$ clusters.*

Proof. The proof is a generalization of Lemma 1 from [2]. There are $m + 1$ nodes in $\widetilde{T}$, so at least $m/2 + 1$ of them have degree smaller than 2. Consider $m/2$ edges from these nodes to their parents and denote this set as M. Then, from a charging argument (see the details in [2]) we obtain that at least half of the edges in M would have been merged in a single iteration of the original algorithm. Denote these edges by M', where $|M'| \geq m/4$ and observe that at least $|M'|/2 \geq m/8$ pairs of edges can be merged.

Now, q clusters (edges) are too large to participate in a merge and in the worst case each of them would have participated in a different merge of the original algorithm. Thus, Algorithm 1 performs at least $m/8 - q$ merges and after a single iteration the number of clusters decreases to at most $m - (m/8 - q) = 7/8m + q$. ◀

Our goal will be to prove the following theorem.

▶ **Theorem 3.** *Let T be a tree on n nodes labeled from an alphabet of size σ. Then the size of the corresponding top DAG obtained by Algorithm 1 with $\alpha = 10/9$ is $O(\frac{n}{\log_\sigma n})$.*

In the following we assume that $\alpha = 10/9$, but do not substitute it to avoid clutter.

▶ **Lemma 4.** *After t iterations of Algorithm 1 there are $O(n/\alpha^{t+1})$ clusters in $\widetilde{T}$.*

Proof. We prove by induction on t that after t iterations $\widetilde{T}$ contains at most cn/α^{t+1} clusters, where $c = 113$. The case $t = 0$ is immediate. Let $t > 0$. From the induction hypothesis, after $t - 1$ iterations there are at most cn/α^t clusters, p of them having size at most α^t (call them small) and q of them having size larger than α^t that cannot be yet merged in the t^{th} iteration (call them big). We know that $p \leq cn/\alpha^t$ and, as the big clusters are pairwise disjoint, $q \leq n/\alpha^t$.

We need to show that the total number of clusters after t iterations is at most cn/α^{t+1}. There are two cases to consider:

- $q \leq \frac{1}{100}p$: We apply Lemma 2 and conclude that the total number of clusters after the t^{th} iteration is at most $7/8(p + q) + q < 9/10p \leq cn/\alpha^{t+1}$.
- $p < 100q$: In the worst case no pair of clusters was merged and the total number of clusters after the t^{th} iteration is $p + q < 101q < 101n/\alpha^t \leq 113n/\alpha^{t+1} = cn/\alpha^{t+1}$. ◀

Proof of Theorem 3. Clusters are represented with binary trees labeled either with pairs of labels from the original alphabet or one of the 5 labels representing the type of merging, so in total there are $|\Sigma|^2 + 5 \leq \sigma^2 + 5$ possible labels of nodes in $\mathcal{T}$. From the properties of

Catalan numbers, it follows that the number of different binary trees of size x is bounded by 4^x. Thus there are at most $\sum_{i=1}^{x}(4(\sigma^2+5))^i \leq \sum_{i=1}^{x}(12\sigma^2)^i \leq (12\sigma^2)^{x+1}$ distinct labeled trees of size at most x, since $\sigma \geq 2$. Even if some of them appear many times in $\widetilde{T}$, they will be represented only once in $\mathcal{TD}$.

Consider the situation after $t-1$ iterations of the algorithm. Then, from Lemma 4 there are at most $O(n/\alpha^t)$ clusters in $\widetilde{T}$. Setting t to be the maximal integer number such that $\alpha^t + 1 \leq 3/4 \log_{12\sigma^2} n$ we obtain that there are at most $n^{3/4}$ distinct subtrees of $\mathcal{T}$ of size at most α^t. As identical subtrees of $\mathcal{T}$ are identified by the same node in the top DAG, all clusters created during the first $t-1$ iterations of the algorithm are represented by at most $n^{3/4}$ nodes in $\mathcal{TD}$. Next, the remaining $O(n/\alpha^t)$ clusters can introduce at most that many new nodes in the DAG.

Finally, the size of the DAG obtained by Algorithm 1 on a tree T of size n is bounded by $n^{3/4} + O(n/\alpha^t) = O(n/\log_{12\sigma^2} n)$, which is $O(n/\log_\sigma n)$ as $\sigma \geq 2$. ◀

References

1 Philip Bille, Finn Fernstrøm, and Inge Li Gørtz. Tight bounds for top tree compression. In *SPIRE*, volume 10508 of *Lecture Notes in Computer Science*, pages 97–102. Springer, 2017.

2 Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Inf. Comput.*, 243:166–177, 2015.

3 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In *VLDB*, pages 141–152. Morgan Kaufmann, 2003.

4 Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4-5):456–474, 2008.

5 Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

6 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009.

7 Markus Frick, Martin Grohe, and Christoph Koch. Query evaluation on compressed trees (extended abstract). In *LICS*, page 188. IEEE Computer Society, 2003.

8 Paweł Gawrychowski and Artur Jeż. LZ77 factorisation of trees. In *FSTTCS*, volume 65 of *LIPIcs*, pages 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

9 Lorenz Hübschle-Schneider and Rajeev Raman. Tree compression with top trees revisited. In *SEA*, volume 9125 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2015.

10 Guy Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554. IEEE Computer Society, 1989.

11 Artur Jeż and Markus Lohrey. Approximation of smallest linear tree grammar. *Inf. Comput.*, 251:215–251, 2016.

12 Markus Lohrey and Sebastian Maneth. The complexity of tree automata and xpath on grammar-compressed trees. *Theor. Comput. Sci.*, 363(2):196–210, 2006.

13 Markus Lohrey, Sebastian Maneth, and Eric Noeth. XML compression via dags. In *ICDT*, pages 69–80. ACM, 2013.

14 Markus Lohrey, Carl Philipp Reh, and Kurt Sieber. Optimal top dag compression. *CoRR*, abs/1712.05822, 2017. `arXiv:1712.05822`.