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 $\sigma$, 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 $\sigma$.

1 Introduction

Tree compression with top trees introduced by Bille et al. [2] 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 $O(\frac{n}{\log_{\sigma} n})$ for a tree on $n$ nodes labeled from an alphabet of size $\sigma$. 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 [4] 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 1.1. There exists an infinite family of trees on $n$ nodes labeled from an alphabet of size $\sigma$ for which size of the top DAG is $\Omega(\frac{n}{\log_{\sigma} n} \log \log_{\sigma} n)$.

This answers an open question explicitly mentioned by Lohrey et al. [5], 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, a top tree of the
tree $T'$ obtained by collapsing every cluster of $T$ is constructed by applying the algorithm of Bille et al. [2]. Finally, all top trees are patched together to obtain a top tree of $T$. While this method guarantees that the number of distinct clusters is $\Omega\left(\frac{n}{\log n}\right)$, its disadvantage is that the resulting procedure is non-uniform, and in particular needs to be aware of the value of $\sigma$ 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 1.1, 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 $i$th iteration adjacent clusters are merged only if their size is at most $\alpha^i$, 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 1.1 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]. The naming convention is mostly preserved.

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 $\Sigma$. $T(v)$ 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 $1 \leq s \leq r \leq k$ 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 $\mathcal{T}$.
- 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 its two children. The label of each internal node is the type of merge it represents (out of the five merging options).
Figure 1: Result of a single iteration. Dotted lines denote the merged edges (clusters) and thick edges denote 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.

The children are ordered so that the left child is the child cluster visited first in a preorder traversal of $T$.

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

On a high level, the iterations are designed in such a way that every time a constant fraction of edges of $	ilde{T}$ are merged. This is proved in Lemma 1 of [2], and we describe a slightly more general property in Lemma 4.1. 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 $v \in \tilde{T}$ with $k \geq 2$ children $v_1, \ldots, v_k$, for $i = 1$ to $\left\lfloor \frac{k}{2} \right\rfloor$, 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 $\tilde{T}$ 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 $\{(v_1, v_2), (v_2, v_3)\}, \ldots, \{(v_{p-2}, v_{p-1})\}$. If $p$ is odd merge the following pairs of edges $\{(v_1, v_2), (v_2, v_3)\}, \ldots, \{(v_{p-3}, v_{p-2})\}$, 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 one iteration in Figure 1. Finally, the compressed representation of $T$ is the so-called top DAG $\mathcal{T}_D$, which is the minimal DAG representation of $T$ obtained by identifying identical subtrees of $T$. As every iteration shrinks $\tilde{T}$ by a constant factor, $T$ can be computed in $O(n)$ time, and then $\mathcal{T}_D$ can be computed in $O(|T|)$ time [3]. Thus, the entire compression takes $O(n)$ time.

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

In this section, we prove Theorem 1.1 and show that the $O(n \log \log n)$ bound from [4] on the number of distinct clusters created by the algorithm described in Section 2 is tight.
For every $k \in \mathbb{N}$ we will construct a tree $T_k$ on $n = \sigma^{8k}$ nodes for which the corresponding top DAG is of size $\Theta\left(\frac{n}{\log \sigma} \log \log \sigma n\right)$. Let $t = 8^k = \log \sigma n$. 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 3$k$ iterations becomes a single cluster. Similarly, let $S_k$ be the full ternary tree of height $k$ with $3^k$ leaves, so $\frac{3^k - 1}{2} = O(3^k) = O(t^{0.53})$ nodes in total. Observe that after 3 iterations $S_k$ becomes $S_{k-1}$, and so after 3$k$ 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 3$k$ iterations $G_k$ consists of $2^k - 1$ clusters $CS$ corresponding to $S_k$ and one cluster $CP$ 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. In every gadget $G_k$ we assign the labels of the nodes of $P_k$ so that the cluster $CP$ obtained after the first 3$k$ iterations corresponds to a distinct subtree of the top tree. Consequently, so does the cluster obtained from $CP$ in each of the subsequent $k$ iterations.

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 $\Sigma$. 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 $C^{(i)}_P$ and the clusters obtained from it during the $k$ iterations corresponds to a distinct subtree of the
top tree. Thus, overall the top DAG contains $\Omega(n/t \cdot k) = \Omega(n/t \cdot \log t) = \Omega(n/\log \sigma \cdot n \cdot \log \log \sigma)$ nodes, which concludes the proof of Theorem 1.1.

4 An optimal tree compression algorithm

Let $\alpha$ be a constant greater than 1 and consider the following modification of algorithm \cite{2}. 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 $\alpha^t$. However, this would require a slight modification of the original charging argument. To avoid this, in each iteration we first generate all merges that would have been performed in both steps of the original algorithm. Then we apply only the merges in which both clusters have size at most $\alpha^t$.

Algorithm 1 A modified top tree construction algorithm of Bille et al. \cite{2}.

1: for $t = 1, \ldots, O(\log n)$ do
2: simulate one iteration of the original algorithm
3: apply only merges with both clusters of size at most $\alpha^t$
4: construct the top DAG $\mathcal{T}\mathcal{D}$ of the obtained tree $\mathcal{T}$
5: return $\mathcal{T}\mathcal{D}$

Clearly, the depth of the obtained DAG is $O(\log n)$ as before, because after $\log \alpha n$ iterations the algorithm is no longer constrained and can behave not worse than the original one. In the following lemma we show that even if there are some clusters that cannot be merged in one step, the tree still shrinks by roughly a constant factor.

Lemma 4.1. Suppose that there are $m = p + q$ clusters in the beginning of the $t$th iteration of Algorithm 1, where $q$ is the number of clusters of size larger than $\alpha^t$. Then, after the $t$th iteration there are at most $7/8m + q$ clusters.

Proof. The proof is a generalization of the Lemma 1 from \cite{2}. There are $m + 1$ nodes in $\tilde{T}$, 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 \cite{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 clusters can be merged.

Now, $q$ clusters are too large to participate in a merge. Thus, in the worst case, we can perform at least $m/8 - q$ merges. Thus, after a single iteration the number of clusters decreases to at most $m - (m/8 - q) = 7/8m + q$. \hfill \square

Our goal will be to prove the following theorem.

Theorem 4.2. Let $T$ be a tree on $n$ nodes labeled from an alphabet of size $\sigma$. Then the size of the corresponding top DAG obtained by Algorithm 1 with $\alpha = 10/9$ is $O(\sqrt[n]{\log \sigma n})$.

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

Lemma 4.3. After the $t$th iteration of Algorithm 1 there are $O(n/\alpha^{t+1})$ clusters in $\tilde{T}$.

Proof. We prove by induction on $t$ that after the $t$th iterations $\tilde{T}$ contains at most $cn/\alpha^{t+1}$ clusters, where $c = 113$. The basis of the induction is immediate. Consider the $t$th iteration. From the induction hypothesis, after the $(t - 1)$th iteration there are at most $cn/\alpha^t$ clusters, $p$
of them having size at most $\alpha^t$ (call them small) and $q$ of them having size larger than $\alpha^t$ 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 disjoint, $q \leq n/\alpha^t$.

We need to show that the total number of clusters after the $t^{th}$ iteration is at most $cn/\alpha^{t+1}$.

There are two cases to consider:

- $q \leq \frac{1}{100}p$: We apply Lemma 4.1 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 4.2.** Clusters are represented with binary trees labeled either with pairs of labels from the original alphabet or one of the $5\sigma^2$ labels representing the type of merging, so in total there are $\sigma^2 + 5$ 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 $\sum_{x=1}^{x}(4(\sigma^2 + 5))^i \leq \sum_{x=1}^{x}(24\sigma^2)^i \leq (24\sigma^2)^{x+1}$ distinct labeled trees of size at most $x$. Even if some of them appear many times in $\tilde{T}$, they will be represented only once in $\mathcal{T}D$.

Consider the situation at the beginning of the $t^{th}$ iteration of the algorithm. Then, from Lemma 4.3 there are at most $O(n/\alpha^t)$ clusters in $\tilde{T}$. Setting $t$ such that $\alpha^t+1 = 3/4 \log_{24\sigma^2} n$ we obtain that up to this point at most $n^{3/4}$ distinct clusters of size at most $\alpha^t$ have been created. As identical subtrees of $\mathcal{T}$ are identified by the same node in the top DAG, all these clusters are represented by $n^{3/4}$ nodes in $\mathcal{T}D$. Next, the remaining $O(n/\alpha^t)$ clusters can introduce at most that many new nodes in the DAG.

Finally, size of the DAG obtained by the Algorithm 1 on a tree $T$ of size $n$ is bounded by $n^{3/4} + O(n/\alpha^t) = O(n/\log_{24\sigma^2} n)$.

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

[4] Lorenz Hübischle-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.

[5] Markus Lohrey, Carl Philipp Reh, and Kurt Sieber. Optimal top dag compression. CoRR, abs/1712.05822, 2017. arXiv:1712.05822.