SIZE-OPTIMAL TOP DAG COMPRESSION

MARKUS LOHREY, CARL PHILIPP REH, AND KURT SIEBER

Abstract. It is shown that for a given ordered node-labelled tree of size n and with s different node labels, one can construct in linear time a top dag of height $\mathcal{O}(\log n)$ and size bounded by $\mathcal{O}(n/\log_\sigma n)$ and $\mathcal{O}(d \cdot \log n)$, where $\sigma = \max\{2, s\}$ and d is the size of the minimal dag. The size bound $\mathcal{O}(n/\log_\sigma n)$ is optimal and improves on previous bounds.

1. Introduction

Top dags were introduced by Bille et al. [1] as a formalism for the compression of unranked node-labelled ordered trees. Roughly speaking, the top dag for such a tree t is the dag representation of an expression that evaluates to t, where the expression builds t from edges using two merge operations (horizontal and vertical merge). In [1], a linear time algorithm is presented that constructs from a tree of size n with node labels from the set Σ a top dag of size $\mathcal{O}(n/\log_\sigma^{0.19} n)$, where $\sigma = \max\{2, |\Sigma|\}$ (note that this definition of σ avoids the base 1 in the logarithm) and the size of a top dag is the number of its edges. Later, in [7] this bound was improved to $\mathcal{O}(n \log\log n/\log_\sigma n)$ (for the same algorithm as in [1]).

If $|\mathsf{tdag}(t)|$ denotes the size of a smallest top dag for the tree t, then a simple counting argument shows that $\Omega(n/\log_\sigma n)$ is the information-theoretic lower bound for $\max\{|\mathsf{tdag}(t)| \mid t \in \mathcal{T}_{n,\sigma}\}$, where $\mathcal{T}_{n,\sigma}$ denotes the set of all trees of size n with σ node labels.[1] We present a new linear-time top dag construction that achieves this bound. In addition, our construction has two properties that are also true for the original construction of Bille et al. [1]: (i) the size of the constructed top dag is bounded by $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$, where $\mathrm{dag}(t)$ is the minimal dag of t and (ii) the height of the top dag is bounded by $\mathcal{O}(\log n)$. Concerning (i) it was shown in [2] that the factor $\log n$ is unavoidable. The logarithmic bound on the height of the top dag in (ii) is important to obtain the logarithmic time bounds for the querying operations (e.g., computing the label, parent node, first child, right sibling, depth, height, nearest common ancestor, etc. of nodes given by their preorder numbers) in [1].

Our construction is based on a modification of the algorithm BU-Shrink (bottom-up shrink) from [5], which constructs in linear time a TSLP of size $\mathcal{O}(n/\log_\sigma n)$ for a given binary tree. In fact, we construct the top dag in two phases: in the first phase we apply the modification of BU-Shrink, whereas in the second phase we apply the top dag construction of Bille et al.

[1]This can be shown by noting that (i) a top dag of size m can be encoded by a bit string of length $\mathcal{O}(m \log m)$ and (ii) the number of trees of size n with σ node labels is $\Theta(4^n \sigma^n / n^{3/2})$, which follows from the asymptotics for the Catalan numbers.

Related work. After the arXiv-version of this paper had appeared, an alternative construction of top dags of size $\mathcal{O}(n/\log_\sigma n)$ was presented in [4]. In that paper, it is also shown that the $\mathcal{O}(n \log\log n/\log_\sigma n)$ bound for the top dag construction from [1, 7] cannot be improved.

In [6] top dags are compared with two other formalisms for the grammar-based compression of unranked trees: (i) forest straight-line programs (which generalize tree straight-line programs that are used for the compression of ranked trees, see the survey [8]) and (ii) tree straight-line programs for first-child next sibling encodings of unranked trees. It is shown in [6] that (i) and (ii) are equally succint and that top dags can be transformed into both formalisms with a constant blow-up, but the transformation from (i) or (ii) to top dags requires a blow-up of size $\mathcal{O}(\sigma)$.

2. Preliminaries

Let Σ be a finite alphabet. By $\mathcal{T}$ we denote the set of *ordered labelled trees* with labels from Σ. Here, "ordered" means that the children of every node are linearly ordered. Fix a tree $t \in \mathcal{T}$. We use the following notations: r_t is the root of t, $V(t)$ is the set of nodes of t and $E(t)$ is the set of edges of t. If $v \in V(t)$ then we use $\lambda_t(v)$ to denote the label of v in t, and we use $t(v)$ to denote the subtree of t that is rooted in v. The number of children of v is called the *degree* of v, $\deg(v)$ for short. Our trees are unranked in the sense that the node label does not determine the degree of the node. A *cluster of rank* 0 is just a tree t. A *cluster of rank* 1 is a tree t together with a distinguished leaf node ℓ_t that we call the *bottom boundary node* of t. In both cases, the root r_t is called the *top boundary node*. Let C_i be the set of all clusters of rank $i \in \{0, 1\}$ and let $C = C_0 \cup C_1$. With $\mathrm{rank}(t)$ we denote the rank of the cluster t. An atomic cluster consists of a single edge, i.e., it is a tree with two nodes.

We define two partial binary merge operations $\oplus, \ominus : C \times C \to C$:

(1) $s \oplus t$ (the vertical merge of s and t) is only defined if $s \in C_1$ and $\lambda_s(\ell_s) = \lambda_t(r_t)$. We obtain $s \oplus t$ by taking the disjoint union of s and t and then merging $\ell_s \in V(s)$ with $r_t \in V(t)$ (note that this is possible since the labels coincide). The rank of $s \oplus t$ is $\mathrm{rank}(t)$ and if $t \in C_1$, then the bottom boundary node of $s \oplus t$ is ℓ_t.

(2) $s \ominus t$ (the horizontal merge of s and t) is only defined if $\mathrm{rank}(s) + \mathrm{rank}(t) \leqslant 1$ and $\lambda_s(r_s) = \lambda_t(r_t)$. We obtain $s \ominus t$ by taking the disjoint union of s and t and then merging $r_s \in V(s)$ with $r_t \in V(t)$ (note that this is possible since the labels coincide). If the children of r_s (resp., r_t) are ordered as $u_1, \ldots, u_k$ (resp., $v_1, \ldots, v_l$) then the order of the children of the root of $s \ominus t$ is $u_1, \ldots, u_k, v_1, \ldots, v_l$. The rank of $s \ominus t$ is $\mathrm{rank}(s) + \mathrm{rank}(t)$. In case $s \in C_1$ (resp., $t \in C_1$), the bottom boundary node of $s \ominus t$ is ℓ_s (resp., ℓ_t).

The (minimal) directed acyclic graph (dag) for a tree t is obtained by keeping for every subtree t' of t only one occurrence of that subtree and replacing every edge that goes to a node in which a copy of t' is rooted by an edge to the unique chosen occurrence of t'. We denote this dag as $\mathrm{dag}(t)$. Note that the number of nodes in $\mathrm{dag}(t)$ is the number of different subtrees that occur in t. We define the size $|\mathrm{dag}(t)|$ as the number of edges of $\mathrm{dag}(t)$.

We can now define *top trees* and *top dags*. A top tree is a binary node-labelled ordered tree, where every internal node is labelled with one of the two operations $\oplus, \ominus$ and every leaf is labelled with an atomic cluster plus a bit for the rank of the

cluster. The latter information can be represented by a triple (a, b, i) with $a, b \in \Sigma$ and $i \in \{0, 1\}$. Moreover, for a top tree T it is required that we can evaluate T to a tree $t \in \mathcal{T}$ by recursively applying the merge operations at its inner nodes. In other words, if an internal node v of T is labelled with the operation $\odot \in \{\ominus, \oplus\}$ and its left (resp., right) child evaluates to the cluster s_1 (resp., s_2), then $s_1 \odot s_2$ must be defined according to the above definitions (1) and (2). Also note that the root of T is required to evaluate to a tree t, i.e., a cluster of rank 0. We then say that T is a top tree for t and $\mathrm{dag}(T)$ is a top dag for t. Note that the number of nodes and the number of edges of a top dag differ at most by the factor 2 (since every internal node of a top dag has two outgoing edges). Hence, if we only want to prove an asymptotic upper bound for the size of a top dag, we can either count its edges or its nodes.

Let $t \in \mathcal{T}$ be a tree. A *subcluster* of t of rank one is an induced subgraph of t that is obtained as follows: Take a node $u \in V(t)$ with the ordered sequence of children $u_1, \ldots, u_d$ and let $1 \leqslant i \leqslant j \leqslant d$. Let $v \in V$ be a node that belongs to one of the subtrees $t(u_i), \ldots, t(u_j)$. Then the tree is induced by the nodes in $\{u, v\} \cup (\bigcup_{s=i}^{j} t(u_s) \setminus t(v))$. The node u (resp., v) is the top (resp., bottom) boundary node of the cluster. A subcluster of t of rank zero is obtained in the same way, except that we take the tree induced by the nodes in $\{u\} \cup \bigcup_{s=i}^{j} t(u_s)$. Its top boundary node is u. Note that every edge of t is a subcluster of t. We identify a subcluster of t with the set of edges of t belonging to the subcluster. If T is a top tree for t then it follows easily by induction that every subtree of T evaluates to an isomorphic copy of a subcluster of t.

To prove our main result in the next section, we need the following lemma, which is shown in [1].

Lemma 2.1. *There is a linear time algorithm that computes from a given tree t with $n \geqslant 1$ edges a top tree T of height $\mathcal{O}(\log n)$ and size $\mathcal{O}(n)$. The corresponding top dag $\mathrm{dag}(T)$ has size $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$.*

We also need the following simple lemma:

Lemma 2.2. *Let $t \in \mathcal{T}$ with $m \geqslant 1$ edges and let $U = \{v \in V(t) \setminus \{r_t\} \mid \deg(v) \leqslant 1\}$. Then we have $|U| > m/2$.*

Proof. Let $U' = \{v \in V(t) \mid \deg(v) \geqslant 2\}$ and let $m' \leqslant m$ be the total number of outgoing edges of nodes $v \in U'$. Then $m' \geqslant 2 \cdot |U'|$ and thus $|U'| \leqslant m'/2$.

We distinguish two cases: If $r_t \in U'$ then we have $U = V(t) \setminus U'$ which implies $|U| = m + 1 - |U'| \geqslant m + 1 - m'/2 > m/2$. If $r_t \notin U'$ then $m' < m$ (since the outgoing edge of r_t is not counted in m') and $U = V(t) \setminus (U' \cup \{r_t\})$. This implies $|U| = m + 1 - (|U'| + 1) = m - |U'| \geqslant m - m'/2 > m/2$. □

Our exposition of top trees and top dags differs slightly from Bille et al. [1]. Bille et al. only define the above notion of a subcluster inside a concrete tree t, and then construct a concrete top tree for t by partitioning t into certain subclusters. Our approach is slightly different. We define top trees as expression trees T over the two merge operations $\ominus$ and $\oplus$. Such an expression tree T can then be evaluated to a tree t, and we say that T is a top tree of t. We believe that this definition adds flexibility: a top dag compression algorithm is any algorithm that constructs a top tree T of a given input tree t. The output is then the dag representation of this top tree. Moreover, the definition of a top tree as an expression tree over certain

merge operations allows to compare top dags better with related grammar-based formalisms for tree compression that only differ in the merge operations (e.g. forest straight-line programs [6]). To define a particular grammar-based compression formalism for trees, one has to fix a finite set of operations for constructing trees. Every tree should be constructible from certain constants (for top dags these constants are atomic clusters) and the operations. Then a grammar-based compressor constructs from a tree t the dag-representation of an expression tree that evaluates to t. Top dags, forest straight-line programs and ordinary dags can be seen as concrete instantiations of this idea. This is also our approach from [6].

3. Optimal worst-case compression

We can now state and prove the main result of this paper:

Theorem 3.1. *Let $\sigma = \max\{|\Sigma|, 2\}$. There is a linear time algorithm that computes from a given tree $t \in \mathcal{T}$ with $n \geqslant 1$ edges a top dag of height $\mathcal{O}(\log n)$, whose size is bounded by $\mathcal{O}(n/\log_\sigma n)$ and $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$.*

Proof. We first prove the theorem without the bound $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$ on the size of the constructed top dag. In a second step, we explain how to modify the algorithm to obtain the $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$ bound.

Take a tree $t \in \mathcal{T}$ with $n \geqslant 1$ edges and let $\sigma = \max\{|\Sigma|, 2\}$. We build from t a sequence of trees $t_0, t_1, \ldots t_m$, where every edge $(u, v) \in E(t_i)$ (u is the parent node of v) is labelled with a subcluster $c^i_{u,v}$ of t. If v is a leaf of t_i, then $c^i_{u,v}$ is a subcluster of rank 0 with top boundary node u, otherwise $c^i_{u,v}$ is a subcluster of rank 1 with top boundary node u and bottom boundary node v. The number of edges in the subcluster $c^i_{u,v}$ is also called the weight $\gamma^i_{u,v}$ of the edge (u, v).

Our algorithm does not have to store the subclusters explicitly but only their weights. Moreover, the algorithm builds for every edge $(u, v) \in E(t_i)$ a top tree $T^i_{u,v}$. The invariant of the algorithm is that $T^i_{u,v}$ evaluates to (an isomorphic copy of) the subcluster $c^i_{u,v}$. The top trees $T^i_{u,v}$ are stored as pointer structures, but below we write them for better readability as expressions using the operators $\oplus$ and $\ominus$.

The initial tree t_0 is the tree t, where $c^0_{u,v} = \{(u, v)\}$ for every edge $(u, v) \in E(t_0)$. Thus, $c^0_{u,v}$ is a subcluster of rank 0 if v is a leaf, and of rank 1 otherwise. We set $\gamma^0_{u,v} = 1$ and $T^0_{u,v} = (\lambda_t(u), \lambda_t(v), i)$, where i is the rank of the subcluster $c^0_{u,v}$.

Let us now fix a number $k \leqslant n$ that will be made precise later. Our algorithm proceeds as follows: Let t_i be the current tree. We proceed by a case distinction. Ties between the following three cases can be broken in an arbitrary way. The updating of the subclusters $c^i_{u,v}$ is only shown to give a better intuition for the algorithm; it is not part of the algorithm.

Case 1. There exist edges $(u, v), (v, w) \in E(t_i)$ of weight at most k such that w is the unique child of v. We obtain t_{i+1} from t_i by (*i*) removing the node v, and (*ii*) replacing the edges $(u, v), (v, w)$ by the edge (u, w). Moreover, we set

$$\begin{aligned} c^{i+1}_{u,w} &:= c^i_{u,v} \cup c^i_{v,w}, \\ T^{i+1}_{u,w} &:= T^i_{u,v} \oplus T^i_{v,w}, \\ \gamma^{i+1}_{u,w} &:= \gamma^i_{u,v} + \gamma^i_{v,w}. \end{aligned}$$

For all edges $(x,y) \in E(t_i) \backslash \{(u,v),(v,w)\}$ we set

$$c^{i+1}_{x,y} := c^i_{x,y}, \quad T^{i+1}_{x,y} := T^i_{x,y} \quad \text{and} \quad \gamma^{i+1}_{x,y} := \gamma^i_{x,y}. \tag{1}$$

Case 2. There exist edges $(u,v),(u,w) \in E(t_i)$ of weight at most k such that v is a leaf and the left sibling of w. Then t_{i+1} is obtained from t_i by removing the edge (u,v). Moreover, we set

$$\begin{aligned} c^{i+1}_{u,w} &:= c^i_{u,v} \cup c^i_{u,w}, \\ T^{i+1}_{u,w} &:= T^i_{u,v} \ominus T^i_{u,w}, \\ \gamma^{i+1}_{u,w} &:= \gamma^i_{u,v} + \gamma^i_{u,w}. \end{aligned}$$

For all edges $(x,y) \in E(t_i) \backslash \{(u,v),(u,w)\}$ we make the updates from (1).

Case 3. There exist edges $(u,v),(u,w) \in E(t_i)$ of weight at most k such that v is a leaf and the right sibling of w. We make the same updates as in Case 2 except that $T^{i+1}_{u,w}$ is set to $T^i_{u,w} \ominus T^i_{u,v}$.

If none of the above three cases holds, then the algorithm stops. Let $t' = t_m$ be the final tree that we computed. Note that every edge (u,v) of t' has weight at most $2k$. We now bound the number of edges of t':

Claim: The number of edges of t' is less than $\frac{8n}{k}$.

Proof of the claim: Let n' be the number of edges of t'. Note that $n' \geqslant 1$ and that t' has $n'+1$ nodes. If $n' = 1$ we are done, since $\frac{8n}{k} \geqslant 8$. So, assume that $n' \geqslant 2$. Let U be the set of all nodes in t' of degree at most one except for the root node $r_{t'}$. By Lemma 2.1 we have $|U| > n'/2$. For every node $u \in U$, let $p(u)$ be its parent node. We now assign to certain edges of t' (possibly several) markings by doing the following for every $u \in U$: If the weight of the edge $(p(u),u)$ is larger than k then we assign to $(p(u),u)$ a marking. Now assume that the weight of $(p(u),u)$ is at most k. If u has degree one and v is the unique child of u, then the weight of (u,v) must be larger than k (otherwise, we would merge the edges $(p(u),u)$ and (u,v)), and we assign a marking to (u,v). The remaining case is that u is a leaf. Since t' has at least two edges, at least one of the following three edges must exist:

- $(p(u),v)$, where v is the left sibling of u,
- $(p(u),v)$, where v is the right sibling of u,
- $(v,p(u))$, where $p(u)$ has degree one.

Moreover, each of these edges (if it exists) must have weight more than k. We choose such an edge and assign a marking to it. The following then holds:

- Markings are only assigned to edges of weight more than k.
- Every edge of t' can get at most 4 markings (see Figure 1 for an edge which really gets 4 markings).
- In total, t' contains $|U| > n'/2$ many markings.

Since the sum of all edge weights of t' is n, we obtain

$$n \geqslant \frac{|U| \cdot k}{4} > \frac{n' \cdot k}{8}$$

Thus, we have $n' < \frac{8n}{k}$. This shows the claim.

We now build a top tree T for t as follows: Construct a top tree T' for t' of height $\mathcal{O}(\log n)$ using Lemma 2.2. Consider a leaf e of T'. It corresponds to an edge $(u,v) \in E(t')$. In the process of folding the cluster $c^m_{u,v}$ into the edge (u,v) we have

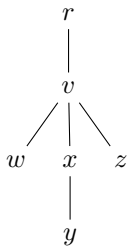


FIGURE 1. If (v, x) has weight $> k$ then no further merges are allowed by our algorithm. If all other edges have weight $\leqslant k$ then (v, x) gets 4 markings because $U = \{w, x, y, z\}$ and each node $u \in U$ causes a marking on (v, x).

constructed the top tree $T_e := T^m_{u,v}$ that evaluates to the cluster $c^m_{u,v}$. Therefore, we obtain a top tree T for t by replacing every leaf e of T' by the top tree T_e. To bound the minimal dag of T we have to count the number of different subtrees of T. This number can be upper bounded by the number of nodes in T' (which is in $\mathcal{O}(n/k)$) plus the number of different top trees of size at most $2k$. The latter number can be bounded as follows: A top tree for a tree from $\mathcal{T}$ is a binary tree with $2(|\Sigma|^2 + 1)$ many node labels ($2|\Sigma|^2$ many different atomic clusters together with the bit for their rank and two labels for the two merge operations). The number of binary trees with m nodes is bounded by 4^m. Hence, we can bound the number of different top trees of size at most $2k$ by $2k \cdot r^k$ with $r = 64(|\Sigma|^2 + 1)^2$. Note that $\log r \in \Theta(\log \sigma)$, where $\sigma = \max\{|\Sigma|, 2\}$. Take $k = \frac{1}{2} \log_r n \in \Theta(\log_\sigma n)$. Then we obtain the following upper bound on the number of non-isomorphic subtrees of T:

$$\mathcal{O}\left(\frac{n}{\log_\sigma n}\right) + \log_r n \cdot \sqrt{n} = \mathcal{O}\left(\frac{n}{\log_\sigma n}\right)$$

Moreover, the height of T is in $\mathcal{O}(\log n)$ since T' and all T_e have height $\mathcal{O}(\log n)$. For the T_e this follows from the fact that every T_e has size at most $2k \in \mathcal{O}(\log n)$.

It remains to argue that our algorithm can be implemented in linear time. The arguments are more or less the same as for the analysis of BU-Shrink in [5]: The algorithm maintains for every node of t_i its degree, and for every edge (u, v) its weight $\gamma^i_{u,v}$. Additionally, the algorithm maintains a queue that contains pointers to all edges (u, v) of t_i having weight at most k and such that v has degree one. Then every merging step can be done in constant time, and there are at most n merging steps. Finally, the minimal dag of T can be computed in linear time by [3].

We now explain the modification of the above algorithm such that the constructed top dag has size $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$. Note that the above algorithm has in general several choices for the edges that are merged in each step of the first phase. It is not clear to us, whether every possible choice leads to a top dag of size $\mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$. Intuitively, to obtain this bound, one should construct isomorphic top trees for isomorphic subtrees of t (at least to some extent). The idea that ensures this property is to perform the first phase of the above algorithm (where the tree t' is constructed) on $\mathrm{dag}(t)$ instead of t itself. Thus, the algorithm starts with the construction of $d := \mathrm{dag}(t)$ from t, which is possible in linear time [3]. We now build from d a sequence of dags $d_0, d_1, \ldots d_m$, where analogously to the above

construction every edge (u, v) of d_i is labelled with a weight $\gamma^i_{u,v}$ and a top tree $T^i_{u,v}$. Since we are working on the dag, we cannot assign a unique subcluster $c^i_{u,v}$ of t to the dag edge (u, v). In fact, every edge of d_i represents a set of isomorphic subclusters that are shared in the dag. The top tree $T^i_{u,v}$ evaluates to an isomorphic copy of these subclusters.

The initial dag d_0 is the dag d where every edge $(u, v) \in E(d_0)$ is labelled with the top tree $T^0_{u,v}$ that only consists of the leaf $(\lambda_d(u), \lambda_d(v), i)$ (λ_d assigns to every node of the dag its label from Σ) where $i = 0$ if v is a leaf of the dag and $i = 1$ otherwise. We take the same threshold value $k \leqslant n$ as before. Let d_i be the current dag. Also the case distinction is the same as before:

Case 1. There exist edges $(u, v), (v, w) \in E(d_i)$ of weight at most k such that w is the unique child of v. We obtain d_{i+1} from d_i by replacing the edge (u, v) by the edge (u, w). If the node v has no incoming edge after this modification, we can remove v and the edge (v, w) (although this is not necessary for the further arguments). The weights $\gamma^i_{x,y}$ and the top trees $T^i_{x,y}$ are updated in exactly the same way as in the previous case 1.

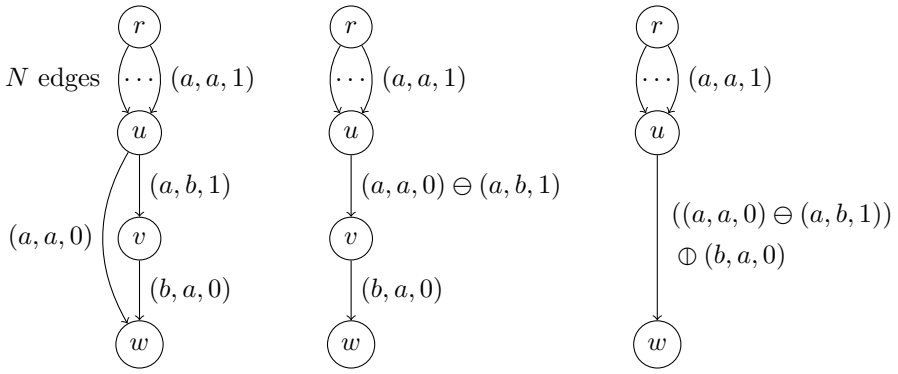
Case 2. There exist edges $(u, v), (u, w) \in E(d_i)$ of weight at most k such that v is a leaf of d_i (i.e., has no outgoing edge) and it is the left sibling of w in the list of all children of u. Then d_{i+1} is obtained from d_i by removing the edge (u, v). If v has no more incoming edges after this modification, then we can also remove v. The weights $\gamma^i_{x,y}$ and the top trees $T^i_{x,y}$ are updated in exactly the same way as in the previous case 2.

Case 3. There exist edges $(u, v), (u, w) \in E(d_i)$ of weight at most k such that v is a leaf and it is the right sibling of w in the list of all children of u. Then d_{i+1} is obtained from d_i by removing the edge (u, v). If v has no more incoming edges after this modification, then we can remove v. The weights $\gamma^i_{x,y}$ and the top trees $T^i_{x,y}$ are updated in exactly the same way as in the previous case 3.

If none of the above three cases holds, then the algorithm stops. Let $d' = d_m$ be the final dag that we computed. We unfold d' to a tree t'. This tree t' is one of the potential outcomes of the above tree version of the algorithm. The rest of the construction is the same as before: We apply Lemma 2.2 to construct a top tree T' for t', then we combine T' with the top trees $T^m_{u,v}$ to obtain a top tree T for t, and finally we return $\mathrm{dag}(T)$ as the result of our algorithm.

The size bound $\mathcal{O}(n/\log_\sigma n)$ and the height bound $\mathcal{O}(\log n)$ for $\mathrm{dag}(T)$ follow from our previous arguments. It remains to show that $|\mathrm{dag}(T)| \in \mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$. Note that $|\mathrm{dag}(T)|$ is bounded by $|\mathrm{dag}(T')|$ plus the total size of all top trees $T^m_{u,v}$. Lemma 2.2 ensures that $|\mathrm{dag}(T')|$ is bounded by $\mathcal{O}(|\mathrm{dag}(t')| \cdot \log |t'|) \leqslant \mathcal{O}(|d'| \cdot \log n)$ where the latter inequality holds because d' is a dag for t'. The total size of all top trees $T^m_{u,v}$ is also bounded by $\mathcal{O}(|d'| \cdot \log n)$ since each $T^m_{u,v}$ has size at most $2k \in \mathcal{O}(\log_\sigma n) \leqslant \mathcal{O}(\log n)$. Hence we have $|\mathrm{dag}(T)| \in \mathcal{O}(|d'| \cdot \log n)$. Finally note that $|d'| = |d_m| \leqslant |d_0| = |\mathrm{dag}(t)|$ since each step from d_i to d_{i+1} of the above construction does not increase the number of nodes and edges. Thus we obtain $|\mathrm{dag}(T)| \in \mathcal{O}(|\mathrm{dag}(t)| \cdot \log n)$. □

Example 3.2. Let $\Sigma = \{a, b\}$. In the following picture we show a dag d_0 (on the left) with 4 nodes and $N + 3$ edges and a possible run of the merging algorithm up to d_2 (on the right):



In the first step we merge the two atomic clusters $(a, a, 0)$ and $(a, b, 1)$ using $\ominus$. This is done by removing the edge (u, w) since w is a leaf and it is the left sibling of v among all children of u. Then we merge the clusters $(a, a, 0) \ominus (a, b, 1)$ and $(b, a, 0)$ using $\oplus$. This is done by replacing the edge (u, v) with the edge (u, w). The edge (v, w) is removed because it has no further incoming edges.

Note that these two merging steps are only allowed in our algorithm if the threshold value k is at least 2, i.e., if $|t_0| \geqslant 4 \cdot (64(|\Sigma|^2 + 1)^2)^2 = 10240000$ for the original tree t_0 with $d_0 = \mathrm{dag}(t_0)$. Since $|t_0| = 4N + 1 > 4N$ we can choose $N = 64(|\Sigma|^2 + 1)^2)^2$ to obtain a threshold value k with $2 < k < 3$. In this case no further merging step is allowed on d_2 because (u, w) has weight 3 and merging steps without the edge (u, w) are anyway impossible. Hence we unfold d_2 to a tree t_2 of size $2N + 1$ and proceed with the algorithm of Bille et al. [1] on t_2.

References

[1] P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. *Inf. Comput.*, 243:166–177, 2015.

[2] P. Bille, F. Fernstrøm, and I. L. Gørtz. Tight bounds for top tree compression. In *Proc. SPIRE 2017*, volume 10508 of LNCS, 97–102. Springer, 2017.

[3] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

[4] B. Dudek and P. Gawrychowski. Slowing down top trees for better worst-case bounds. In *Proc. CPM 2018*, volume 105 of *LIPIcs*, pages 16:1–16:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

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

[6] A. Gascón, M. Lohrey, S. Maneth, C. P. Reh, and K. Sieber. Grammar-based compression of unranked trees. In *Proc. CSR 2018*, volume 10846 of LNCS, 118–131, Springer 2018.

[7] L. Hübschle-Schneider and R. Raman. Tree compression with top trees revisited. In *Proc. SEA 2015*, volume 9125 of LNCS, 15–27. Springer, 2015.

[8] M. Lohrey. Grammar-based tree compression. In *Proc. DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2015.

E-mail address: {lohrey,reh,sieber}@eti.uni-siegen.de