Compression by Contracting Straight-Line Programs

Moses Ganardi
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

In grammar-based compression a string is represented by a context-free grammar, also called a *straight-line program (SLP)*, that generates only that string. We refine a recent balancing result stating that one can transform an SLP of size g in linear time into an equivalent SLP of size $\mathcal{O}(g)$ so that the height of the unique derivation tree is $\mathcal{O}(\log N)$ where N is the length of the represented string (FOCS 2019). We introduce a new class of balanced SLPs, called *contracting* SLPs, where for every rule $A \to \beta_1 \dots \beta_k$ the string length of every variable β_i on the right-hand side is smaller by a constant factor than the string length of A. In particular, the derivation tree of a contracting SLP has the property that every subtree has logarithmic height in its leaf size. We show that a given SLP of size g can be transformed in linear time into an equivalent contracting SLP of size $\mathcal{O}(g)$ with rules of constant length. This result is complemented by a lower bound, proving that converting SLPs into so called α-balanced SLPs or AVL-grammars can incur an increase by a factor of $\Omega(\log N)$.

We present an application to the navigation problem in compressed unranked trees, represented by *forest straight-line programs (FSLPs)*. A linear space data structure by Reh and Sieber (2020) supports navigation steps such as going to the parent, left/right sibling, or to the first/last child in constant time. We extend their solution by the operation of moving to the i-th child in time $\mathcal{O}(\log d)$ where d is the degree of the current node.

Contracting SLPs are also applied to the finger search problem over SLP-compressed strings where one wants to access positions near to a pre-specified *finger* position, ideally in $\mathcal{O}(\log d)$ time where d is the distance between the accessed position and the finger. We give a linear space solution for the dynamic variant where one can set the finger in $\mathcal{O}(\log N)$ time, and then access symbols or move the finger in time $\mathcal{O}(\log d + \log^{(t)} N)$ for any constant t where $\log^{(t)} N$ is the t-fold logarithm of N. This improves a previous solution by Bille, Christiansen, Cording, and Gørtz (2018) with access/move time $\mathcal{O}(\log d + \log \log N)$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases grammar-based compression, balancing, finger search

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.45

Related Version *Full Version*: `https://arxiv.org/abs/2107.00446` [9]

Acknowledgements The author thanks Paweł Gawrychowski, Artur Jeż, Philipp Reh, and Louisa Seelbach Benkner for helpful discussions. The author is also indebted to the anonymous referees whose comments improved the presentation of this work.

1 Introduction

In grammar-based compression a long string is represented by a context-free grammar, also called a *straight-line program (SLP)*, that generates only that string. Straight-line programs can achieve exponential compression, e.g. a string of length 2^n can be produced by the grammar with the rules $A_n \to A_{n-1}A_{n-1}, \dots, A_0 \to a$. While it is NP-hard to compute a smallest SLP for a given string [5] there are efficient grammar-based compressors of both practical and theoretical interest such as the LZ78/LZW-algorithms [25, 24], Sequitur [19], and Re-Pair [16]. There is a close connection between grammar-based compression and the LZ77 algorithm, which parses a string into z phrases (without self-references): On the one hand z is always a lower bound on the size of the smallest SLP for the string [5]. On the other hand one can always construct an SLP of size $\mathcal{O}(z \log N)$ where N is the string length [5, 22] (see also [13] for LZ77 with self-referential phrases). Furthermore, the hierarchical structure of straight-line programs makes them amenable to algorithms that work directly on the compressed representation, without decompressing the string first. We refer to [17] for a survey on the broad literature on algorithms on grammar-compressed data.

Balanced grammars. For some algorithmic applications it is useful if the SLP at hand satisfies certain balancedness conditions. In the following we always denote by N the length of the represented string. A recent result states that one can transform an SLP of size g in linear time into an equivalent SLP of size $\mathcal{O}(g)$ so that the height of the unique derivation tree is $\mathcal{O}(\log N)$ [10]. This yields a clean $\mathcal{O}(g)$ space data structure which supports random access to any position i in the string in time $\mathcal{O}(\log N)$, by descending in the derivation tree from the root to the i-th leaf. The original solution for the random access problem by Bille, Landau, Raman, Sadakane, Satti, and Weimann relied on a sophisticated weighted ancestor data structure [3]. Its advantage over the balancing approach from [10] is that it supports random access to the string defined by any given variable A in time $\mathcal{O}(\log |A|)$.

Although the derivation tree of an SLP may have logarithmic height its subtrees may still be very unbalanced. Arguably, the strongest balancedness notions are α-balanced SLPs [5] and AVL-grammars [22]. An SLP in Chomsky normal form is *α-balanced* if for every rule $A \to BC$ the ratios $|B|/|A|$ and $|C|/|A|$ lie between α and $1-\alpha$. An *AVL-grammar* is an SLP in Chomsky normal form whose derivation tree is an AVL-tree, i.e. for every rule $A \to BC$ the subtree heights below B and C differ at most by one. In fact, the aforementioned transformations from LZ77 into SLPs produce an α-balanced SLP, with $\alpha \leq 1 - \frac{1}{2}\sqrt{2}$ [5], and an AVL-grammar [22]. Using the same proof techniques one can also transform an SLP of size g into an α-balanced SLP or an AVL-grammar of size $\mathcal{O}(g \log N)$ [5, 22].

Let us list a few algorithmic results on α-balanced SLPs and AVL-grammars. The *compressed pattern matching problem* can be solved in linear time if the text is given by an α-balanced SLP and the pattern is given explicitly [13]. Gagie, Gawrychowski, Kärkkäinen, Nekrich, and Puglisi [7] presented a solution for the *bookmarking problem* in α-balanced SLPs or AVL-grammars of size g. Given b positions in the string, called bookmarks, we can decompress any substring of length ℓ that covers a bookmark in time $\mathcal{O}(\ell)$ and space $\mathcal{O}(g + b \log^* N)$. Based on this bookmarking data structure they present *self-indexes* for LZ77 and SLPs [7, 8], which support extracting substrings and finding all occurrences of a given pattern. Abboud, Backurs, Bringmann, and Künnemann studied the Hamming distance problem and the subsequence problem on SLP-compressed strings [1]. As a first step their algorithms convert the input SLPs into AVL-grammars, and solve both problems in time $\tilde{\mathcal{O}}(g^{1.410} \cdot N^{0.593})$, improving on the decompress-and-solve $\mathcal{O}(N)$ time algorithms.

Main results. The starting point of this paper is the observation that the size increase by a $\mathcal{O}(\log N)$ factor in the transformation from SLPs to α-balanced SLPs or AVL-grammars is unavoidable (Theorem 4). This lower bound holds whenever in the derivation tree any path from a variable A to a leaf has length $\Theta(\log |A|)$. This motivates the search for balancedness notions of SLPs that can be established without increasing the size by more than a constant factor and that provide good algorithmic properties. We introduce a new class of balanced SLPs, called *contracting straight-line programs*, in which every variable β_i occurring on the right-hand side of a rule $A \to \beta_1 \dots \beta_k$ satisfies $|\beta_i| \leq |A|/2$. The derivation tree of an contracting SLP has the property that every subtree has logarithmic height in its leaf size, i.e. in the number of descendant leaves. We explicitly admit rules with right-hand sides of length greater than two, however, the length will always be bounded by a constant in this paper. We say that an SLP $\mathcal{G}$ defines a string s if some variable in $\mathcal{G}$ derives s (and s only). The main theorem of this paper refines the balancing theorem from [10] as follows:

▶ **Theorem 1.** *Given an SLP $\mathcal{G}$ of size g, one can compute in linear time a contracting SLP of size $\mathcal{O}(g)$ with constant-length right-hand sides which defines all strings that $\mathcal{G}$ defines.*

As an immediate corollary we obtain a simple $\mathcal{O}(g)$ size data structure supporting access to the i-th symbol of a variable A in time $\mathcal{O}(\log |A|)$ instead of $\mathcal{O}(\log N)$. This is useful whenever multiple strings $s_1, \dots, s_m$ are compressed using a single SLP since we can support random access to any string s_i in time $\mathcal{O}(\log |s_i|)$. We present an example application to unranked trees represented by *forest straight-line programs (FSLPs)*. FSLPs are a natural generalization of SLPs that can compress trees both horizontally and vertically, and share the good algorithmic applicability of their string counterparts [11]. Reh and Sieber presented a linear space data structure on FSLP-compressed trees that allows to perform various navigation steps in constant time [21]. We extend their data structure by the operation of moving to the i-th child in time $\mathcal{O}(\log d)$ where d is the degree of the current node.

▶ **Theorem 2.** *Given an FSLP $\mathcal{G}$ of size g, one can compute an data structure in $\mathcal{O}(g)$ time and space supporting the following operations in constant time: Move to the root of the first/last tree of a given variable, move to the first/last child, to the left/right sibling or to the parent of the current node, return the symbol of the current node. One can also move to the i-th child of the current node in time $\mathcal{O}(\log d)$ where d is the degree of the current node.*

A second application concerns the *finger search problem* on grammar-compressed strings. A finger search data structure supports fast updates and searches to elements that have small rank distance from the *fingers*, which are pointers to elements in the data structure. The survey [4] provides a good overview on dynamic finger search trees. In the setting of finger search on a string s, Bille, Christiansen, Cording, and Gørtz [2] considered three operations: $\mathsf{access}(i)$ returns symbol $s[i]$, $\mathsf{setfinger}(i)$ sets the finger at position i of s, and $\mathsf{movefinger}(i)$ moves the finger to position i in s. Given an SLP of size g for a string of length N, they presented an $\mathcal{O}(g)$ size data structure which supports $\mathsf{setfinger}(i)$ in $\mathcal{O}(\log N)$ time, and $\mathsf{access}(i)$ and $\mathsf{movefinger}(i)$ in $\mathcal{O}(\log d + \log\log N)$ time where d is the distance from the current finger position [2]. If we assume that the SLP is α-balanced or an AVL-grammar, there is a linear space solution supporting $\mathsf{access}(i)$ and $\mathsf{movefinger}(i)$ in $\mathcal{O}(\log d)$ time (Theorem 17). For general SLPs we present a finger search structure with improved time bounds:

▶ **Theorem 3.** *Let $t \geq 1$. Given an SLP of size g for a string of length N, one can support $\mathsf{setfinger}(i)$ in $\mathcal{O}(\log N)$ time, and $\mathsf{access}(i)$ and $\mathsf{movefinger}(i)$ in $\mathcal{O}(\log d + \log^{(t)} N)$ time, where d is the distance between i and the current finger position, after $\mathcal{O}(tg)$ preprocessing time and space.*

Here $\log^{(t)} N$ is the t-fold logarithm of N, i.e. $\log^{(0)} N = N$ and $\log^{(t+1)} N = \log\log^{(t)} N$. Choosing any constant t we obtain a linear space solution for dynamic finger search, supporting $\mathsf{access}(i)$ and $\mathsf{movefinger}(i)$ in $\mathcal{O}(\log d + \log^{(t)} N)$ time. Alternatively, we obtain a clean $\mathcal{O}(\log d)$ time solution if we admit a $\mathcal{O}(g \log^* N)$ space data structure. Theorem 3 also works for multiple fingers where every finger uses additional $\mathcal{O}(\log N)$ space.

Let us remark that Theorem 1 holds in the *pointer machine model* [23], whereas for Theorem 2 and Theorem 3 we assume the *word RAM model* with the standard arithmetic and bitwise operations on w-bit words, where $w \geq \log N$. The assumption on the word length is standard in the area of grammar-based compression, see [3, 2].

Overview of the proofs. The proof of Theorem 1 follows the ideas from [3] and [10]. The obstacle for $\mathcal{O}(\log N)$ time random access or $\mathcal{O}(\log N)$ height are occurrences of *heavy* variables on right-hand sides of rules $A \to \beta_1 \dots \beta_k$, i.e. variables β_i whose length exceeds $|A|/2$. These occurrences can be summarized in the *heavy forest*, which is a subgraph of the directed acyclic graph associated with the SLP. The random access problem can be reduced to *weighted ancestor queries* (see Section 5) on every heavy tree whose edges are weighted by the lengths of the variables that branch off from the heavy tree. Using a "biased" weighted ancestor data structure one can descend in the derivation tree in $\mathcal{O}(\log N)$ time, spending amortized constant time on each heavy tree [3]. Our main contribution is a solution of the weighted ancestor problem in the form of an SLP: Given a tree T of size n where the edges are labeled by weighted symbols, we construct a contracting SLP of size $\mathcal{O}(n)$ defining all prefixes in T, i.e. labels of paths from the root to some node. The special case of defining all prefixes of a weighted string by a weight-balanced SLP of linear size (i.e. T is a path) was solved in [10]; however, the constructed SLP only satisfies a weaker balancedness condition.

To solve finger search efficiently, Bille, Christiansen, Cording, and Gørtz first consider the *fringe access problem* [2]: Given a variable A and a position $1 \le i \le |A|$, access symbol $A[i]$, ideally in time $\mathcal{O}(\log d)$ where $d = \min\{i, |A| - i + 1\}$. For this purpose the SLP is partitioned into *leftmost* and *rightmost trees*, which produce strings of length N, $N^{1/2}$, $N^{1/4}$, $N^{1/8}$, etc. The leftmost/rightmost trees can be traversed in $\mathcal{O}(\log \log N)$ time using a $\mathcal{O}(\log \log N)$ time weighted ancestor data structure by Farach-Colton and Muthukrishnan [6]. Applying this approach to contracting SLPs one can solve fringe access in time $\mathcal{O}(\log d + \log \log |A|)$ since the trees have $\mathcal{O}(\log N)$ height, for which one can answer weighted ancestor queries in constant time using a predecessor data structure by Pătraşcu-Thorup [20]. Using additional weighted ancestor structures, we can reduce the term $\log \log |A|$ to $\log^{(t)} N$.

2 Straight-line programs

A *context-free grammar* $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ consists of a finite set $\mathcal{V}$ of *variables*, an alphabet Σ of *terminal symbols*, where $\mathcal{V} \cap \Sigma = \emptyset$, a finite set $\mathcal{R}$ of *rules* $A \to u$ where $A \in \mathcal{V}$ and $u \in (\mathcal{V} \cup \Sigma)^*$ is a *right-hand side*, and a *start variable* $S \in \mathcal{V}$. The set of *symbols* is $\mathcal{V} \cup \Sigma$. We call $\mathcal{G}$ a *straight-line program (SLP)* if every variable occurs exactly once on the left-hand side of a rule and there exists a linear order $<$ on $\mathcal{V}$ such that $A < B$ whenever B occurs on the right-hand side of a rule $A \to u$. This ensures that every variable A derives a unique string $[\![A]\!] \in \Sigma^*$. We also write $|A|$ for $|[\![A]\!]|$. A string $s \in \Sigma^*$ is *defined* by $\mathcal{G}$ if $[\![A]\!] = s$ for some $A \in \mathcal{V}$. The *size* of $\mathcal{G}$ is the total length of all right-hand sides in $\mathcal{G}$. We denote by $\mathsf{height}(A)$ the height of the derivation tree rooted in A. The height of $\mathcal{G}$ is $\mathsf{height}(S)$. We define the directed acyclic graph $\mathsf{dag}(\mathcal{G}) = (\mathcal{V} \cup \Sigma, E)$ where E is a multiset of edges, containing for every rule $A \to \beta_1 \dots \beta_k$ in $\mathcal{R}$ the edges $(A, \beta_1), \dots, (A, \beta_k)$. An SLP $\mathcal{G}$ can be transformed in linear time into an SLP $\mathcal{G}'$ in *Chomsky normal form* which defines all strings that $\mathcal{G}$ defines, i.e. each rule is of the form $A \to BC$ or $A \to a$ where $A, B, C \in \mathcal{V}$ and $a \in \Sigma$.

An SLP is *α-balanced*, for some constant $0 < \alpha \le 1/2$, if it is in Chomsky normal form and for all rules $A \to BC$ both $|B|/|A|$ and $|C|/|A|$ lie between α and $1 - \alpha$. An *AVL-grammar* is an SLP in Chomsky normal form where for all rules $A \to BC$ we have $|\mathsf{height}(B) - \mathsf{height}(C)| \le 1$. An SLP in Chomsky normal form is *(α, β)-path balanced*, for some constants $0 < \alpha \le \beta$, if for every variable A the length of every root-to-leaf path in the derivation tree is between $\alpha \log |A|$ and $\beta \log |A|$. Observe that every α-balanced SLP is $(1/\log(\alpha^{-1}), 1/\log((1-\alpha)^{-1}))$-path balanced and AVL-grammars are $(0.5, 2)$-path balanced. The latter follows from the fact that the height decreases at most by 2 when going from an

AVL-tree to an immediate subtree. There are algorithms that compute for given a string w an α-balanced SLP [5] and an AVL-grammar [22] of size $\mathcal{O}(g \log N)$ where g is the size of a smallest SLP for w. We show that these bounds are optimal even for path balanced SLPs: There are strings for which the smallest path balanced SLPs have size $\Omega(g \log N)$.

▶ **Theorem 4.** *There exists a family of strings $(s_n)_{n \geq 1}$ over $\{a, b\}$ such that $|s_n| = \Omega(2^n)$, s_n has an SLP of size $\mathcal{O}(n)$ and every (α, β)-path balanced SLP has size $\Omega(n^2)$.*

Proof. First we use an unbounded alphabet. Let $s_n = b_1 a^{2^n} b_2 a^{2^n} \dots b_{n-1} a^{2^n} b_n$, which has an SLP of size $\mathcal{O}(n)$ with the rules $S \to b_1 A_n b_2 A_n \dots b_n$, $A_0 \to a$ and $A_i \to A_{i-1} A_{i-1}$ for all $1 \leq i \leq n$. Consider an (α, β)-path balanced SLP $\mathcal{G}$ for s_n. We will show that $\mathsf{dag}(\mathcal{G})$ has $\Omega(n^2)$ edges. Let $1 \leq i \leq n$ and consider the unique path in $\mathsf{dag}(\mathcal{G})$ from the starting variable to b_i. Let π_i be the suffix path starting in the lowest node A_i such that $[\![A_i]\!]$ contains some symbol b_j with $i \neq j$. Therefore $|A_i| \geq 2^n$. Since $\mathcal{G}$ is (α, β)-path balanced π_i has length $\geq \alpha n$. Since all paths π_i are edge-disjoint it follows that $\mathcal{G}$ has size $\Omega(n^2)$.

For a binary alphabet define the separator $T_i = ba^{2i-2}ba^{2i-1}b$ for $1 \leq i \leq n$ and define $s_n = T_1 a^{2^n} T_2 \dots T_{n-1} a^{2^n} T_n$ of length $\Omega(2^n)$. The string s_n also has an SLP of size $\mathcal{O}(n)$. Consider an (α, β)-path balanced SLP $\mathcal{G}$ for s_n. Let $1 \leq i \leq n$ and consider the unique path ρ_i in $\mathsf{dag}(\mathcal{G})$ from the starting variable to the symbol b in the middle of the separator $T_i = ba^{2i-2}ba^{2i-1}b$. Let B_i be the lowest node on ρ_i such that $[\![B_i]\!]$ contains either $ba^{2i-2}b$ or $ba^{2i-1}b$. Since the successor of B_i on ρ_i produces a string strictly shorter than $|T_i| \leq 4n$, the suffix path of ρ_i starting in B_i has length at most $1 + \beta \log(4n) = \mathcal{O}(\log n)$. Let A_i be the lowest ancestor of B_i on ρ_i such that $[\![A_i]\!]$ contains a symbol from a separator T_j for $i \neq j$. Therefore $|A_i| \geq 2^n$ and hence the suffix path of ρ_i starting in A_i has length at least $\alpha \log(2^n) = \alpha n = \Omega(n)$. Thus, the path π_i from A_i to B_i has length $\Omega(n) - \mathcal{O}(\log n) = \Omega(n)$. All paths π_i are edge-disjoint since for any edge (X, Y) in π_i, $[\![Y]\!]$ is of the form $a^\ell ba^{2i-2}ba^r$ or $a^\ell ba^{2i-1}ba^r$. This implies that $\mathcal{G}$ has size $\Omega(n^2)$. ◀

We define contracting SLPs over a weighted alphabet, i.e. an alphabet Γ equipped with a function $\|\cdot\|\colon \Gamma \to \mathbb{N} \setminus \{0\}$, which is extended additively to Γ^*. The standard weight function is the length function $|\cdot|$. A symbol β occurring in a weighted string s is *heavy in s* if $\|\beta\| > \|s\|/2$; otherwise it is *light in s*. Consider an SLP $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ over a weighted alphabet Σ. We define $\|A\| = \|[\![A]\!]\|$ for $A \in \mathcal{V}$. A symbol $\beta \in \mathcal{V} \cup \Sigma$ is a *heavy child* of $A \in \mathcal{V}$ if β is heavy on the right-hand side of the rule $A \to u$. We also call β a *heavy* symbol. A rule $A \to u$ is *contracting* if u contains no heavy variables, i.e. every variable B occurring in u satisfies $\|B\| \leq \|A\|/2$. Let us emphasize that heavy *terminal* symbols from Σ are permitted in contracting rules. If all rules in $\mathcal{G}$ are contracting we call $\mathcal{G}$ contracting. If B occurs heavily in a rule $A \to uBv$ and the rule $B \to x$ is contracting we can *expand* the occurrence of B and obtain a contracting rule $A \to uxv$.

3 Transformation into contracting SLPs

A *labeled tree* $T = (V, E, \gamma)$ is a rooted tree where each edge $e \in E$ is labeled by a string $\gamma(e)$ over a weighted alphabet Γ. A *prefix* in T is the labeling of a path starting from the root. The first step towards proving Theorem 1 is a reduction to the following problem: Given a labeled tree T, construct a contracting SLP over Γ of size $\mathcal{O}(|T|)$, defining all prefixes in T.

Decomposition into heavy trees. Consider an SLP $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$. If a rule $A \to \beta_1 \dots \beta_k$ contains a unique heavy symbol β_i then $\beta_1 \dots \beta_{i-1}$ is the *light prefix* of A and $\beta_{i+1} \dots \beta_k$ is the *light suffix* of A. The *heavy forest* $H = (\mathcal{V} \cup \Sigma, E_H)$ contains all edges (A, β) where

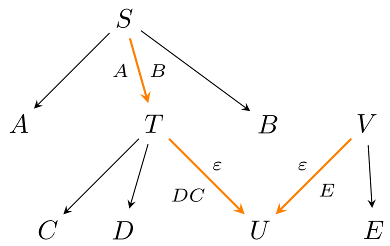

Figure 1 An excerpt from the dag representation of an SLP. The variables S, T, U, V form a heavy tree with root U. The value of S can be split into the prefix ACD, the root U of the heavy tree, and the suffix B. Observe that the left labeling of the path from U to S is DCA, which is the reverse of the prefix ACD.

$\beta \in \mathcal{V} \cup \Sigma$ is a heavy child of $A \in \mathcal{V}$, which is a subgraph of $\mathsf{dag}(\mathcal{G})$. Notice that the edges in H point towards the roots, i.e. if $(\alpha, \beta) \in E$ then α is a child of β in H. We define two labeling functions: The *left label* $\lambda(e)$ of an edge $e = (A, \beta)$ is the *reversed* light prefix of A and the *right label* $\rho(e)$ of e is the light suffix of A. The connected components of (H, λ) and (H, ρ) are called the *left labeled* and *right labeled heavy trees*, which can be computed in linear time from $\mathcal{G}$. If β is the root of a heavy tree containing a variable A we can factorize $[\![A]\!]$ into the reversed left labeling from A to its root β in H, the value of β, and the right labeling of the path from β to A. In that way one can redefine every variable using SLPs which define all prefixes in the left labeled and the right labeled heavy trees.

▶ **Proposition 5.** *Given an SLP $\mathcal{G}$ and contracting SLPs $\mathcal{H}_\mathsf{L}$ and $\mathcal{H}_\mathsf{R}$ defining all prefixes of all left labeled and right labeled heavy trees of $\mathcal{G}$. Let g be the total number of variables in the SLPs and r be the maximal length of a right-hand side. One can compute in linear time a contracting SLP $\mathcal{G}'$ which defines all strings that $\mathcal{G}$ defines, has $\mathcal{O}(g)$ variables and right-hand sides of length $\mathcal{O}(r)$.*

The goal of this section is to prove the following result.

▶ **Theorem 6.** *Given a labeled tree T with n edges and labels of length $\le \ell$, one can compute in linear time a contracting SLP with $\mathcal{O}(n)$ variables and right-hand sides of length $\mathcal{O}(\ell)$ defining all prefixes in T.*

Together with Proposition 5 it implies Theorem 1. We can always assume that every edge in T is labeled by a single symbol: Edges labeled by ε can clearly be contracted. Edge labels u of length > 1 are replaced by a new symbol X_u of weight $\|u\|$, which can be replaced by u again in the constructed SLP. We will also assume that all symbols in T are distinct.

Prefixes of weighted strings. We start with the case where the tree is a path, i.e. we need to define all prefixes of a weighted string s using $\mathcal{O}(|s|)$ contracting rules. The following theorem refines [10, Lemma III.1] where only the path length from a prefix variable S_i to a symbol a_j in the derivation tree was bounded by $\mathcal{O}(1 + \log \frac{\|S_i\|}{\|a_j\|})$.

▶ **Theorem 7.** *Given a weighted string s of length n one can compute in linear time a contracting SLP with $\mathcal{O}(n)$ variables with right-hand sides of length at most 10 that defines all nonempty prefixes of s.*

Let us illustrate the difficulty of defining all prefixes with contracting rules. Consider the weighted string $s = a_1 \dots a_n$ where symbol a_i has weight 2^{n-i}. Since in every factor $a_i \dots a_j$ the left-most symbol a_i is heavy, every rule for $a_i \dots a_j$ must split off the first symbol. If for

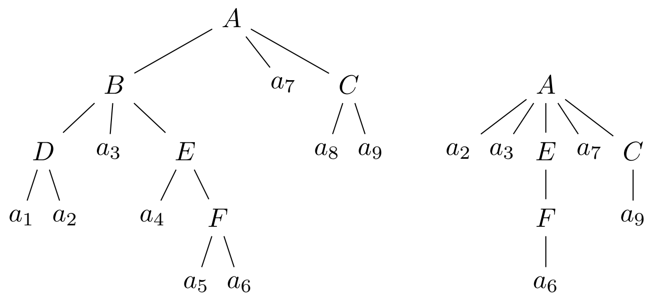

Figure 2 The derivation tree D of a base SLP and the modified tree D_0 containing all symbols which are not a left-most child in D.

every prefix we would only repeatedly split off the first symbol we would create $\Omega(n^2)$ many variables. This shows that there is no better solution with right-hand sides of length ≤ 2. However, using longer rules we can simultaneously reduce both the weight (in a contracting fashion) and the length.

First we recursively construct a contracting "base" SLP $\mathcal{B} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ for the weighted string $s = a_1 \dots a_n$. It will have the additional property of being *left-heavy*, i.e. for every rule $A \to \beta_1 \dots \beta_k$ and all $2 \leq i \leq k$ with $\beta_i \in \mathcal{V}$ we have $\|\beta_1 \dots \beta_{i-1}\| \geq \|\beta_i\|$. Let us emphasize that the condition does not apply when β_i is a terminal symbol. The case $n = 1$ is clear. If $n > 1$ we factorize $s = ua_iv$ such that $u, v \in \Sigma^*$ have weight at most $\|s\|/2$. Next factorize $v = v_1v_2$ such that $|v_1|$ and $|v_2|$ differ at most by one. We add the rule $S \to Ua_iV_1V_2$ to the SLP, possibly omitting variables if some of the strings u, v_1, v_2 are empty. Finally, we recursively define the variables U, V_1 and V_2. The SLP $\mathcal{B}$ is clearly contracting, has at most n variables, since every variable can be identified with the unique symbol $a \in \Sigma$ on its right-hand side, and its right-hand sides have length at most 4. Notice that the rule $S \to Ua_iV_1V_2$ is left-heavy since $\|ua_i\| > \|s\|/2 \geq \|v_1\| + \|v_2\|$.

▶ **Lemma 8.** *The base SLP $\mathcal{B}$ can be computed in linear time from s.*

Consider the derivation tree D of $\mathcal{B}$ whose node set is $\mathcal{S} = \mathcal{V} \cup \Sigma$. Let $\preceq_D$ and $\prec_D$ be the ancestor and the proper ancestor relation on $\mathcal{S}$. For all $\alpha \preceq_D \beta$ we define $\mathsf{left}(\alpha, \beta) = u$ where $\alpha \Rightarrow^*_{\mathcal{B}} u\beta v$ is the unique derivation with $u, v \in \Sigma^*$. In the derivation tree $\mathsf{left}(\alpha, \beta)$ is the string that branches off to the left on the path from α to β. Notice that every proper nonempty prefix of s can be written as $\mathsf{left}(S, a_i) = a_1 \dots a_{i-1}$. For $\alpha \in \mathcal{S} \setminus \{S\}$ occurring in the unique rule $\alpha' \to u\alpha v$ we define the *left sibling string* $\mathsf{lsib}(\alpha) = u$. It satisfies $\mathsf{lsib}(\alpha) \Rightarrow^*_{\mathcal{B}} \mathsf{left}(\alpha', \alpha)$. Notice that we can have $\mathsf{left}(\alpha, \beta) = \mathsf{left}(\alpha', \beta')$ for different pairs (α, β), (α', β'). For a unique description we define the set of nodes $\mathcal{S}_0 \subseteq \mathcal{S}$ which are not a left-most child in D, i.e. symbols α such that $\alpha = S$ or $\mathsf{lsib}(\alpha) \neq \varepsilon$. In particular S belongs to $\mathcal{S}_0$. Observe that $\mathsf{left}(\alpha, \beta) = \mathsf{left}(\alpha', \beta')$ where α' and β' are the lowest ancestors of α and β, respectively, that belong to $\mathcal{S}_0$. In particular, every proper nonempty prefix of s is of the form $\mathsf{left}(\alpha, \beta)$ for some $\alpha, \beta \in \mathcal{S}_0$. Let D_0 be the unique unordered tree with node set $\mathcal{S}_0$ whose ancestor relation is the ancestor relation of D restricted to $\mathcal{S}_0$. Figure 2 shows an example of a tree D with the modified tree D_0.

We will introduce variables $L_{\alpha,\beta}$ for the strings $\mathsf{left}(\alpha, \beta)$. The variable $L_{\alpha,\beta}$ can be defined using $L_{\alpha',\beta'}$ where α' is a child of α in D_0 and β' is the parent of β in D_0. To achieve the $\mathcal{O}(n)$ bound we will restrict to variables $L_{\alpha,\beta}$ that are used in the derivation of a prefix variable, namely $\mathcal{L} = \{L_{\alpha,\beta} \mid \alpha, \beta \in \mathcal{S}_0,\ \alpha \prec \beta,\ \mathsf{level}(\alpha) \leq \mathsf{height}(\beta)\}$. Here $\mathsf{level}(\alpha)$ refers to distance from α to the root in D_0, and $\mathsf{height}(\beta)$ is the height of the subtree of D_0 below β.

▶ **Lemma 9.** *We can compute in linear time a contracting SLP $\mathcal{G} = (\mathcal{V} \cup \mathcal{L}, \Sigma, \mathcal{R} \cup \mathcal{Q}, S)$ with right-hand sides of constant length such that $[\![L_{\alpha,\beta}]\!] = \mathsf{left}(\alpha, \beta)$ for all $L_{\alpha,\beta} \in \mathcal{L}$.*

We have seen that $\mathcal{G}$ defines all nonempty prefixes (S derives s and every proper nonempty prefix is defined by some variable L_{S,a_i}). To finish the proof of Theorem 7 remains to show that $\mathcal{G}$ has $\mathcal{O}(n)$ variables. The SLP $\mathcal{G}$ consists of n variables from the base SLP $\mathcal{B}$ and the variables in $\mathcal{L}$. A variable $L_{\alpha,\beta} \in \mathcal{L}$ is determined by β and the level of α, which is an integer between 0 and the height of β in D_0. Hence it suffices to show that $\sum_{\beta \in \mathcal{S}_0} \mathsf{height}(\beta)$ is $\mathcal{O}(n)$. This follows from the fact that every node in D_0 has logarithmic height in its leaf size.

Prefixes in trees. For Theorem 6 we will construct an SLP $\mathcal{G}$ for the prefixes in T, which will not be contracting in general. Still, its heavy forest is a disjoint union of *caterpillar trees*, i.e. trees where every node has at most one child which is not a leaf. Put differently, every heavy tree of $\mathcal{G}$ consists of a central path $\alpha_1, \dots, \alpha_m$ such that every α_i occurs at most once heavily in a rule $A \to u$ where A is heavy, namely $A = \alpha_{i-1}$. We first extend Theorem 7 to caterpillar trees and then apply Proposition 5 to $\mathcal{G}$, concluding the proof of Theorem 6.

▶ **Proposition 10.** *Given a labeled caterpillar tree T with n edges and labels of length $\le \ell$, one can compute a contracting SLP $\mathcal{G}$ defining all nonempty prefixes in T such that $\mathcal{G}$ has $\mathcal{O}(n)$ variables and right-hand sides of length $\mathcal{O}(\ell)$.*

▶ **Proposition 11.** *Given a labeled tree T with n edges we can compute an SLP $\mathcal{G}$ defining all nonempty prefixes in T such that*
(a) *$\mathcal{G}$ has $4n$ variables and right-hand sides of length ≤ 6,*
(b) *the subgraph of $\mathsf{dag}(\mathcal{G})$ induced by the set of heavy symbols is a disjoint union of paths.*

Proof sketch. We proceed by induction on n. Let us assume a tree $T = (V, E, \omega)$ with $n \ge 2$ edges. We partition E into maximal *unary paths* $\pi = (v_0, \dots, v_k)$, where $k \ge 1$, and $v_1, \dots, v_{k-1}$ have degree one. For every such a path π we create an SLP $\mathcal{G}_\pi$ with the rules $P_{v_0,v_1} \to \omega(v_0, v_1)$ and $P_{v_0,v_i} \to P_{v_0,v_{i-1}}\omega(v_{i-1}, v_i)$ for $2 \le i \le k$. We contract every such a maximal unary path into a single edge (v_0, v_k) labeled by the variable P_{v_0,v_k} and remove all leaves. This new tree T' has at most $n/2$ many edges. Let $V' \subseteq V$ be the node set of T'.

For $v \in V'$ let $d(v)$ be the weight of the path from the root to v in T' and define $\mathsf{rk}(v) = \inf\{k \in \mathbb{Z} \mid d(v) \le 2^k\}$. Let $\hat{v}$ be the highest ancestor of v in T' with $\mathsf{rk}(v) = \mathsf{rk}(\hat{v})$, called the *peak node* of v. We partition T' into subtrees consisting of nodes with the same peak node, and apply the construction recursively on each part. Let $\mathcal{G}'$ be the union of all obtained SLPs with at most $4 \cdot n/2 \le 2n$ variables. For every node $v \in V'$ which is not a peak node $\mathcal{G}'$ contains a variable $B_{\hat{v},v}$ where $[\![B_{\hat{v},v}]\!]$ is the path labeling from $\hat{v}$ to v in T'.

Let $\mathcal{G}$ be the union of $\mathcal{G}'$ and all SLPs $\mathcal{G}_\pi$, which has at most $n + 2n = 3n$ variables. For every $x \in V$ which is not the root of T we add a variable A_x such that $[\![A_x]\!]$ is the labeling of the path from the root to x in T. This yields $4n$ variables, as claimed. Let v be the lowest ancestor of x in T contained in V'. If v is the root then we add the rule

$$A_x \to P_{v,x}. \tag{1}$$

Now assume that v is not the root and hence $\hat{v}$ is also not the root, since the children of the root are peak nodes. Let u be the parent of $\hat{v}$ in T'. If u is the root of T' we add the rule

$$A_x \to P_{u,\hat{v}}\, B_{\hat{v},v}\, P_{v,x}. \tag{2}$$

Otherwise, u and $\hat{u}$ are not the root. Let s be the parent node of $\hat{u}$ in T' and add the rule

$$A_x \to A_s\, P_{s,\hat{u}}\, B_{\hat{u},u}\, P_{u,\hat{v}}\, B_{\hat{v},v}\, P_{v,x}. \qquad (3)$$

One can prove correctness by induction on $\mathsf{rk}(v)$. It remains to prove property (b) of the statement. One can observe that the A- and B-variables are light in the rules (2) and (3). Consider a maximal unary path $\pi = (v_0, \dots, v_k)$. The variable P_{v_0,v_i} for $1 \le i \le k-1$ only occurs in the rule of $P_{v_0,v_{i-1}}$. The variable P_{v_0,v_k} can occur on the right-hand sides of (1), (2) and (3), but the corresponding left-hand side A_x is not heavy. By induction hypothesis P_{v_0,v_k} is the heavy child of at most one heavy variable in $\mathcal{G}'$. This concludes the proof. ◀

4 Navigation in FSLP-compressed trees

As a simple application we extend the navigation data structure on FSLP-compressed trees [21] by the operation which moves to the i-th child in time $\mathcal{O}(\log d)$ where d is the degree of the current node. This is established by applying Theorem 1 to the substructure of the FSLP that compresses forests horizontally.

SLP navigation. The navigation data structure on FSLPs is based on a navigation data structure on (string) SLPs from [18], which extends the data structure from [12] from one-way to two-way navigation. The data structure represents a position $1 \le i \le |A|$ in a variable A by a data structure $\sigma(A, i)$, that we will call *pointer*, which is a compact representation of the path in the derivation tree from A to the leaf corresponding to position i.

▶ **Theorem 12** ([18]). *A given SLP $\mathcal{S}$ can be preprocessed in $\mathcal{O}(|\mathcal{S}|)$ time and space so that the following operations are supported in constant time:*

- *Given a variable A, compute $\sigma(A, 1)$ or $\sigma(A, |A|)$.*
- *Given $\sigma(A, i)$, compute $\sigma(A, i-1)$ or $\sigma(A, i+1)$, or return $\perp$ if the position is invalid.*
- *Given $\sigma(A, i)$, return the symbol at position i in A.*

Furthermore, a single pointer $\sigma(A, i)$ uses $\mathcal{O}(\mathsf{height}(A))$ space and can be computed in time $\mathcal{O}(\mathsf{height}(A))$ for a given pair (A, i).

Forest straight-line programs. In this section we use the natural term representation for forests. Let Σ be an alphabet of node labels. The set of *forests* is defined inductively as follows: The concatenation of $n \ge 0$ forests is a forest (this includes the empty forest ε), and, if $a \in \Sigma$ and t is a forest, then $a(t)$ is a forest. A *context* is a forest over $\Sigma \cup \{x\}$ where x occurs exactly once and this occurrence is at a leaf node. If f is a context and g is a forest or a context then $f\langle g \rangle$ is obtained by replacing the unique occurrence of x in f by g. A *forest straight-line program (FSLP)* $\mathcal{G} = (\mathcal{V}_0, \mathcal{V}_1, \Sigma, \mathcal{R}, S)$ consists of finite sets of forest variables $\mathcal{V}_0$ and context variables $\mathcal{V}_1$, the alphabet Σ, a finite set of rules $\mathcal{R}$, and a start variable $S \in \mathcal{V}_0$. The rules contain arbitrary applications of horizontal concatenation and substitutions of forest and context variables. We restrict ourselves to rules in a certain *normal form*, which can be established in linear time with a constant factor size increase [11]. The normal form assumes a partition $\mathcal{V}_0 = \mathcal{V}_0^\top \cup \mathcal{V}_0^\perp$ where $\mathcal{V}_0^\perp$-variables produce trees whereas $\mathcal{V}_0^\top$-variables

$$
\begin{array}{ll}
A \to a(B) & X \to Y\langle Y\rangle \\
B \to CC & Y \to b\langle DxD\rangle \\
C \to X\langle D\rangle & E \to \varepsilon \\
D \to c(E) &
\end{array}
$$

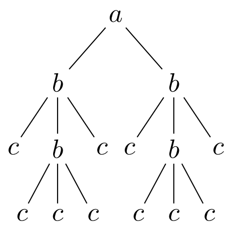

Figure 3 An example FSLP with the variables $\mathcal{V}_0^\perp = \{A, C, D\}$, $\mathcal{V}_0^\top = \{B, E\}$ and $\mathcal{V}_1 = \{X, Y\}$. The tree defined by A is displayed on the right.

produce forests with arbitrarily many trees. The rules in $\mathcal{R}$ have one of the following forms:

$A \to \varepsilon$	where $A \in \mathcal{V}_0^\top$,
$A \to BC$	where $A \in \mathcal{V}_0^\top$ and $B, C \in \mathcal{V}_0$,
$A \to a(B)$	where $A \in \mathcal{V}_0^\perp$, $a \in \Sigma$, and $B \in \mathcal{V}_0$,
$A \to X\langle B\rangle$	where $A, B \in \mathcal{V}_0^\perp$ and $X \in \mathcal{V}_1$,
$X \to Y\langle Z\rangle$	where $X, Y, Z \in \mathcal{V}_1$
$X \to a(LxR)$	where $X \in \mathcal{V}_1$, $a \in \Sigma$, and $L, R \in \mathcal{V}_0$,

Every variable $A \in \mathcal{V}_0$ derives a forest $[\![A]\!]$ and every variable $X \in \mathcal{V}_1$ derives a context $[\![X]\!]$, see [11] for formal definitions. An example FSLP for a tree is shown in Figure 3.

The normal form allows us to define two string SLPs (without start variables) that capture the horizontal and the vertical compression in $\mathcal{G}$. The *rib SLP* $\mathcal{G}_\boxminus = (\mathcal{V}_0, \Sigma_\boxminus, \mathcal{R}_\boxminus)$ over the alphabet $\Sigma_\boxminus = \{\underline{A} \mid A \in \mathcal{V}_0^\perp\}$ contains all rules of the form $A \to \varepsilon$ or $A \to BC$ from $\mathcal{R}$ where $A \in \mathcal{V}_0^\top$, and the rule $A \to \underline{A}$ for all $A \in \mathcal{V}_0^\perp$. We write $[\![A]\!]_\boxminus = \underline{A_1} \dots \underline{A_n}$ for the string derived by A in $\mathcal{G}_\boxminus$, which satisfies $[\![A]\!] = [\![A_1]\!] \dots [\![A_n]\!]$. In the example of Figure 3 we have $[\![B]\!]_\boxminus = \underline{C}\,\underline{C}$. The *spine SLP* $\mathcal{G}_\boxbar = (\mathcal{V}_0^\perp \cup \mathcal{V}_1, \Sigma_\boxbar, \mathcal{R}_\boxbar)$ is defined over the alphabet

$$\Sigma_\boxbar = \{a(B) \mid (A \to a(B)) \in \mathcal{R}\} \cup \{a(LxR) \mid (X \to a(LxR)) \in \mathcal{R}\}.$$

The set $\mathcal{R}_\boxbar$ contains all rules $A \to a(B)$ and $X \to a(LxR)$ from $\mathcal{R}$. It also contains the rule $A \to X$ for all $(A \to X\langle B\rangle) \in \mathcal{R}$ where $A \in \mathcal{V}_0^\perp$, and $X \to YZ$ for all $(X \to Y\langle Z\rangle) \in \mathcal{R}$. We write $[\![V]\!]_\boxbar$ for the string derived by V in $\mathcal{G}_\boxbar$. If $X \in \mathcal{V}_1$ and $[\![X]\!]_\boxbar = a_1(L_1xR_1) \dots a_n(L_nxR_n)$ then $[\![X]\!]$ is the vertical composition of all contexts $a_i([\![L_i]\!]x[\![R_i]\!])$. In the example of Figure 3 we have $[\![C]\!]_\boxminus = b\langle DxD\rangle\, b\langle DxD\rangle$.

FSLP navigation. Now we define the data structure from [21]. It represents a node v in a tree produced by a variable $A \in \mathcal{V}_0$ by a pointer $\tau(A, v)$, which is basically a sequence of navigation pointers in the SLPs $\mathcal{G}_\boxminus$ and $\mathcal{G}_\boxbar$ describing the path from the root of $[\![A]\!]$ to v. Intuitively, the pointer $\tau(A, v)$ can be described as follows. First we select the subtree of $[\![A]\!]$ which contains v, by navigating in $\mathcal{G}_\boxminus$ to a symbol $\underline{B_0}$ where $B_0 \in \mathcal{V}_0^\perp$. The tree $[\![B_0]\!]$ is defined by a sequence of insertion rules $B_0 \to X_1\langle B_1\rangle$, $B_1 \to X_2\langle B_2\rangle, \dots, B_{k-1} \to X_k\langle B_k\rangle$, where possibly $k = 0$, and a final rule $B_k \to a(C)$. We navigate in $\mathcal{G}_\boxbar$ in the variable B_0 from left to right. The string $[\![B_0]\!]_\boxbar$ specifies the contexts $a_j(L_jxR_j)$ which together form the context $[\![X_1]\!]$. If we encounter a context $a_j(L_jxR_j)$ which contains v, there are two cases. If v is the a_j-labeled root then we are done. If v is contained in either L_j or R_j then we record the direction (L or R) and continue recursively from the variable L_j or R_j. If v is not contained in the context X_1 then we reach the end of $[\![B_0]\!]_\boxbar$, and continue searching from B_1, etc. If v is contained in B_k then it is either its root or it is contained in C. In the former case, we are done; in the latter case we record the direction M and continue recursively from C.

To define $\tau(A, v)$ formally, let us write $\sigma_{\boxminus}(A, i)$ and $\sigma_{\boxbar}(A, i)$ for the pointers to the i-th position of a variable $A \in \mathcal{V}_0$ in $\mathcal{G}_{\boxminus}$ and $\mathcal{G}_{\boxbar}$, respectively. We represent every node v in every variable $A \in \mathcal{V}_0$ by a *horizontal pointer* $\tau_{\boxminus}(A, v)$. Furthermore, we represent every node v in every variable $A \in \mathcal{V}_0^{\perp}$, deriving a tree, by a *vertical pointer* $\tau_{\boxbar}(A, v)$. The pointers are defined recursively as follows:

1. Let $A \in \mathcal{V}_0$. If $\llbracket A \rrbracket_{\boxminus} = \underline{A_1} \dots \underline{A_n}$ and v is contained in $\llbracket A_i \rrbracket$ then set $\tau_{\boxminus}(A, v) := \sigma_{\boxminus}(A, i)\, \tau_{\boxbar}(A_i, v)$.
2. Let $A \in \mathcal{V}_0^{\perp}$ with a rule $A \to a(B)$. If v is the root of $\llbracket A \rrbracket$ set $\tau_{\boxbar}(A, v) := \sigma_{\boxbar}(A, 1)$, and otherwise $\tau_{\boxbar}(A, v) := \sigma_{\boxbar}(A, 1)\, \mathsf{M}\, \tau_{\boxminus}(B, v)$.
3. Let $A \in \mathcal{V}_0^{\perp}$ with a rule $A \to X\langle B \rangle$ and $\llbracket A \rrbracket_{\boxbar} = \llbracket X \rrbracket_{\boxbar} = a_1(L_1 x R_1) \dots a_n(L_n x R_n)$. If v is contained in $f_i = a_i(\llbracket L_i \rrbracket x \llbracket R_i \rrbracket)$ set $\tau_{\boxbar}(A, v)$ to be $\sigma_{\boxbar}(A, i)$, $\sigma_{\boxbar}(A, i)\, \mathsf{L}\, \tau_{\boxminus}(L_i, v)$ or $\sigma_{\boxbar}(A, i)\, \mathsf{R}\, \tau_{\boxminus}(R_i, v)$, depending whether v is the root of f_i or is contained in $\llbracket L_i \rrbracket$ or $\llbracket R_i \rrbracket$. If v is contained in $\llbracket B \rrbracket$ set $\tau_{\boxbar}(A, v) := \sigma_{\boxbar}(A, n)\, \tau_{\boxbar}(B, v)$.

For the navigation we only use the horizontal pointers and write $\tau(A, v)$ instead of $\tau_{\boxminus}(A, v)$.

▶ **Theorem 13** ([21]). *A given FSLP $\mathcal{G}$ can be preprocessed in $\mathcal{O}(|\mathcal{G}|)$ time and space so that the following operations are supported in constant time:*

- *Given a variable A, compute $\tau(A, v)$ where v is the root of the first/last tree in $\llbracket A \rrbracket$.*
- *Given $\tau(A, v)$, compute $\tau(A, v')$ where v' is the parent, first/last child or left/right sibling of v, or return $\perp$ if it does not exist.*
- *Given $\tau(A, v)$, return the symbol of node v.*

Navigation to a child. We extend Theorem 13 by the operation which, given a pointer $\tau(S, v)$ and a number $1 \le j \le d$, where v has degree d, moves the pointer to the j-th child of v in $\mathcal{O}(\log d)$ time. To this end we apply Theorem 1 to $\mathcal{G}_{\boxminus}$ so that every variable $A \in \mathcal{V}_0$ in the rib SLP has height $\mathcal{O}(\log |\llbracket A \rrbracket_{\boxminus}|)$, by adding only $\mathcal{O}(g)$ new variables. In particular, we can compute a pointer $\sigma_{\boxminus}(A, i)$ in $\mathcal{O}(\log |\llbracket A \rrbracket_{\boxminus}|)$ time by Theorem 12. Furthermore, we compute the length $|\llbracket A \rrbracket_{\boxminus}|$ for all $A \in \mathcal{V}_0$ in linear time.

Suppose we are given a pointer $\tau(S, v)$ to a node v with degree d for some variable $S \in \mathcal{V}_0$. We show how to compute $\tau(S, v_j)$ where v_j is the j-th child of v in $\mathcal{O}(\log d)$ time.

1. In the first case the last pointer in $\tau(S, v)$ is $\sigma_{\boxbar}(A, 1)$ where the rule of $A \in \mathcal{V}_0^{\perp}$ is of the form $A \to a(B)$. Here B derives the forest below the a-node and we need to move to the root of the j-th tree in the forest. We compute the pointer $\sigma_{\boxminus}(B, j)$ in $\mathcal{O}(\log |\llbracket B \rrbracket_{\boxminus}|) \le \mathcal{O}(\log d)$ time. Then we query the symbol $\underline{B_j}$ at pointer $\sigma_{\boxminus}(B, j)$ and compute the pointer $\sigma_{\boxbar}(B_j, 1)$ in constant time. Then we obtain $\tau(S, v_j) = \tau(S, v)\, \mathsf{M}\, \sigma_{\boxminus}(B, j)\, \sigma_{\boxbar}(B_j, 1)$.
2. In the second case the last pointer in $\tau(S, v)$ is $\sigma_{\boxbar}(A, i)$ where the rule of $A \in \mathcal{V}_0^{\perp}$ is of the form $A \to X\langle B \rangle$. We query the symbol $a_i(L_i x R_i)$ at pointer $\sigma_{\boxbar}(A, i)$. The j-th child v_j is either in L_i, R_i or at the position of the parameter x. If $j = |\llbracket L_i \rrbracket_{\boxminus}| + 1$ we replace $\sigma_{\boxbar}(A, i)$ by $\sigma_{\boxbar}(A, i+1)$ in constant time. If this is not successful then v_j is the root of B and we have $\tau(S, v_j) = \tau(S, v)\, \sigma_{\boxbar}(B, 1)$, which can be computed in constant time. If $j \le |\llbracket L_i \rrbracket_{\boxminus}|$ we compute $\sigma_{\boxminus}(L_i, j)$ in $\mathcal{O}(\log |\llbracket L_i \rrbracket_{\boxminus}|) \le \mathcal{O}(\log d)$ time. We query the symbol B_j at $\sigma_{\boxminus}(L_i, j)$ and compute $\sigma_{\boxbar}(B_j, 1)$ in constant time. Then we have $\tau(S, v_j) = \tau(S, v)\, \mathsf{L}\, \sigma_{\boxminus}(L_i, j)\, \sigma_{\boxbar}(B_j, 1)$. If $j \ge |\llbracket L_i \rrbracket_{\boxminus}| + 2$ we proceed similarly using $\sigma_{\boxminus}(R_i, j - |\llbracket L_i \rrbracket_{\boxminus}| - 1)$.

Remarks. In its original form the SLP navigation data structure from [18] is non-persistent, i.e. the operations modify the given pointer. However, it is not hard to adapt the structure so that an operation returns a fresh pointer, by representing paths in the derivation tree using linked lists that share common prefixes. In a similar fashion, Theorem 2 can be adapted so that a pointer is not modified by a navigation step.

Finally, let us comment on the space consumption of a single pointer in Theorem 2. A single pointer $\tau(A, v)$ consists of a sequence of pointers in $\mathcal{G}_{\boxminus}$ and $\mathcal{G}_{\boxbar}$ that almost describes a path in the derivation tree of A in $\mathcal{G}$. The sequence may contain pointers $\sigma_{\boxbar}(A, n)$ that point to the lowest node above the parameter of a context $[\![X]\!]$. However, in the representation of [18] such a pointer $\sigma_{\boxbar}(A, n)$ only uses $\mathcal{O}(1)$ space, since it is a rightmost path in the derivation tree of A in $\mathcal{G}_{\boxbar}$. Therefore $\tau(A, v)$ uses $\mathcal{O}(\mathsf{height}(A))$ space where $\mathsf{height}(A)$ is the height of the derivation tree of A in $\mathcal{G}$. By [10, Theorem VII.3] we can indeed assume that the FSLP $\mathcal{G}$ has $\mathcal{O}(\log N)$ height while retaining the size bound of $\mathcal{O}(|\mathcal{G}|)$. We also need the fact that the transformation into the normal form increases the height only by a constant factor. However, since the application of Theorem 1 to the rib SLP may possibly increase the total height of the FSLP $\mathcal{G}$ by more than a constant factor, it is unclear whether Theorem 2 can be achieved with $\mathcal{O}(\log N)$ sized pointers.

5 Finger search in SLP-compressed strings

In this section we present our solution (Theorem 3) for the finger search problem using contracting SLPs. Our finger data structure is an *accelerated path*, which compactly represents the path from root to the finger in the derivation tree using precomputed forests on the dag of the SLP. To move the finger we ascend to some variable on the path, branch off from the path, and descend in a subtree while computing the new accelerated path. We can maintain the accelerated path in a dynamic predecessor structure with constant update and query time, thanks to the $\mathcal{O}(\log N)$ height of the SLP. We follow the approach of [2] and present an improved $\mathcal{O}(tg)$ space solution for the *fringe access problem*: Given a variable A and a position $1 \le i \le |A|$, we can access the i-th symbol of $[\![A]\!]$ in time $\mathcal{O}(\log d + \log^{(t)} N)$ where $d = \min\{i, |A| - i + 1\}$ is the distance from the fringe of A, and t is any parameter.

Data structures. Recall that we assume the word RAM model with word size $w \ge \log N$ where N is the string length. Since all occurring sets and trees have size $n \le N$ we have $w \ge \log n$ in the following. We use a dynamic predecessor data structure by Pătraşcu-Thorup, which represents a dynamic set S of $n = w^{\mathcal{O}(1)}$ many w-bit integers in space $\mathcal{O}(n)$, supporting the following updates and queries in constant time [20]: $\mathsf{insert}(S, x) = S \cup \{x\}$, $\mathsf{delete}(S, x) = S \setminus \{x\}$, $\mathsf{pred}(S, x) = \max\{y \in S \mid y < x\}$, $\mathsf{succ}(S, x) = \min\{y \in S \mid y > x\}$, $\mathsf{rank}(S, x) = |\{y \in S \mid y < x\}|$, and $\mathsf{select}(S, i) = x$ with $\mathsf{rank}(S, x) = i$, if any. By enlarging the word size to $2w$ we can identify a number $x \cdot 2^w + y$, where x, y are w-bit numbers, with the key-value pair (x, y), allowing us to store key-value pairs in the data structure sorted by their keys. We remark that all standard operations on a $2w$-bit word RAM can be simulated by a constant number of w-bit operations. This dynamic predecessor structure is used to maintain the accelerated path to the finger. We extend the data structure by the operation $\mathsf{split}(S, x) = \{y \in S \mid y \le x\}$ for $\mathcal{O}(w)$-sized sets.

▶ **Theorem 14** ([20]). *There is a data structure representing a dynamic set S of at most $n = \mathcal{O}(w)$ many w-bit numbers in space $\mathcal{O}(n)$ supporting the operations* $\mathsf{insert}(S, x)$, $\mathsf{split}(S, x)$ *and* $\mathsf{pred}(S, x)$ *in constant time.*

A *weighted tree* T is a rooted tree where each node v carries a nonnegative integer $d(v)$, called the *weighted depth*, satisfying $d(u) \le d(v)$ for all nodes v with parent u. Given a node v and a number $p \in \mathbb{N}$, the *weighted ancestor query* (v, p) asks to return the highest ancestor

u of v with $d(u) > p$. Given a node v and $p \in \mathbb{N}$, we can also compute the highest ancestor u of v where the *weighted distance* $d(u,v) = d(v) - d(u)$ is less than p, by the weighted ancestor query $(v, d(v) - p)$. In our application the edges have nonnegative weights and the weighted depth of a node is computed as the sum of all edge weights on the path from the root.

Kopelowitz and Lewenstein [15] showed that weighted ancestor queries on a tree of size n can be answered in time $\mathcal{O}(\text{pred}(n) + \log^* n)$ where $\text{pred}(n)$ is the query time of a predecessor data structure. It was claimed in [14] that the $\log^* n$-term can be eliminated without giving an explicit proof. We refer to [9, Proposition 17] for a proof in the setting where $n \le w$ using the predecessor structure from [20]. Furthermore, we can also support constant time weighted ancestor queries if the tree height is $\mathcal{O}(w)$.

▶ **Proposition 15.** *A weighted tree T with n nodes and height $h = \mathcal{O}(w)$ can be preprocessed in $\mathcal{O}(n)$ space and time so that weighted ancestor queries can be answered in constant time.*

The fringe access problem. Consider an SLP $\mathcal{G}$ with the variable set $\mathcal{V}$ containing g variables for a string of length N. Using Theorem 1 we assume that $\mathcal{G}$ is in Chomsky normal form and that every variable A has height $\mathcal{O}(\log |A|)$. We precompute in linear time the length of all variables in $\mathcal{G}$. To simplify notation we assume that the variables B and C in all rules $A \to BC$ are distinct, which can be established by doubling the number of variables. We assign to each edge e in $\mathsf{dag}(\mathcal{G})$ a *left weight* $\lambda(e)$ and a *right weight* $\rho(e)$: For every rule $A \to BC$ in $\mathcal{G}$, the edge $e = (A, B)$ has left weight $\lambda(e) = 0$ and right weight $\rho(e) = |C|$, whereas the edge $e = (A, C)$ has left weight $\rho(e) = |B|$ and right weight $\rho(e) = 0$.

Let $\mathcal{F}$ be a finite set of subforests of $\mathsf{dag}(\mathcal{G})$ with node set $\mathcal{V}$ whose edges point towards the roots (as for example in the heavy forest). The forests will be computed later in Proposition 16. For every forest $F \in \mathcal{F}$ we define two edge-weighted versions F_{L} and F_{R} where the edges inherit the left weights and the right weights from $\mathsf{dag}(\mathcal{G})$, respectively, yielding $2|\mathcal{F}|$ many weighted forests. Let $\lambda_F(A)$ and $\rho_F(A)$ be the weighted depths of A in F_{L} and F_{R}, respectively. In $\mathcal{O}(|\mathcal{F}| \cdot g)$ time we compute for all $A \in \mathcal{V}_0$ the weighted depths $\lambda_F(A)$ and $\rho_F(A)$ and the root $\mathsf{root}_F(A)$ of the subtree of F containing A. We write $\lambda_F(A, B)$ and $\rho_F(A, B)$ for the weighted distances between A and B in F_{L} and F_{R}, respectively. We preprocess all $2|\mathcal{F}|$ weighted forests in time and space $\mathcal{O}(|\mathcal{F}| \cdot g)$ to support weighted ancestor queries in constant time according to Proposition 15. This is possible because the height of the forests is $\mathcal{O}(\log N) = \mathcal{O}(w)$.

We denote by $\langle A, i \rangle$ the state in which we aim to compute a compact representation of the path from A to the i-th leaf in the derivation tree of A. Starting from state $\langle A, i \rangle$ we can take short steps and long steps. A *short step* considers the rule of A: If it is a terminal rule $A \to a$ we have found the symbol a. If it is a binary rule $A \to BC$ we compare i with $|B|$: If $i \le |B|$ then the short step leads to $\langle B, i \rangle$, and otherwise to $\langle C, i - |B| \rangle$. A *left long step in* $F \in \mathcal{F}$ is possible if $i \le \lambda_F(A) + |\mathsf{root}_F(A)|$. Put differently, the path from A to $A[i]$ branches off to the left on the path from A to $\mathsf{root}_F(A)$, or continues below $\mathsf{root}_F(A)$. We determine the highest ancestor X of A in F_{L} with $\lambda_F(A, X) < i$ and move to $\langle X, i - \lambda_F(A, X) \rangle$. Using the weighted ancestor data structure on F the variable X can be determined in constant time. Symmetrically, a *right long step in* F is possible if $|A| - i + 1 \le \rho_F(A) + |\mathsf{root}_F(A)|$. Put differently, the path from A to $A[i]$ branches off to the right on the path from A to $\mathsf{root}_F(A)$, or continues below $\mathsf{root}_F(A)$. After finding the highest ancestor X of A in F_{R} with $\rho_F(A, X) < |A| - i + 1$ we move to $\langle X, |A| - i + 1 - \rho_F(A, X) \rangle$.

If we take a long step in a forest F then a subsequent short step moves us from one subtree in F to a different subtree, by maximality of the answer from the weighted ancestor query. A sequence of short and long steps is summarized in an *accelerated path* $(e_1, \dots, e_m)$ of short and long edges. A *short edge* is an edge (A,B) in $\mathsf{dag}(\mathcal{G})$ whereas a *long edge* is a triple (A,F,B) such that $F \in \mathcal{F}$ contains a (unique) path from A to B. In the triple (A,F,B) we store only an identifier of F instead of the forest itself. The left weight and the right weight of a long edge $e=(A,F,B)$ are $\lambda(e)=\lambda_F(A,B)$ and $\rho(e)=\rho_F(A,B)$, respectively.

▶ **Proposition 16.** *Let $t \geq 1$. One can compute and preprocess in $\mathcal{O}(tg)$ time a set of forests $\mathcal{F}$ with $|\mathcal{F}|=\mathcal{O}(t)$ so that given a variable A and a position $1 \leq i \leq |A|$, one can compute an accelerated path from A to $A[i]$ in time $\mathcal{O}(\log d + \log^{(t+1)} N)$ where $d=\min\{i,|A|-i+1\}$.*

Proof sketch. First let us assume that $i \leq |A|/2$. To this end we construct forests $\mathcal{F}=\{F_0,\dots,F_{t-1}\}$ in $\mathcal{O}(tg)$ time so that an accelerated path from A to $A[i]$ can be computed in time $\mathcal{O}(\log i + \log^{(t+1)} N)$. For all $F \in \mathcal{F}$ we construct two constant time weighted ancestor data structures (for F_{L} and F_{R}), and compute $\lambda_F(A)$, $\rho_F(A)$ and $\mathsf{root}_F(A)$ for all $A \in \mathcal{V}$.

The simple algorithm which only uses short steps takes time $\mathcal{O}(\log|A|)$. We first improve the running time to $\mathcal{O}(\log i + \log\log|A|)$. Let $\mathsf{rk}(A)=\min\{k \in \mathbb{N} : |A| \leq 2^{2^k}\}$, which is at most $1+\log\log|A|$. The forest F_0 contains for every rule $A \to BC$ in $\mathcal{G}$ either the edge (A,B), if $\mathsf{rk}(A)=\mathsf{rk}(B)$, or the edge (A,C) if $\mathsf{rk}(A)=\mathsf{rk}(C)>\mathsf{rk}(B)$. If $\mathsf{rk}(A)$ is strictly greater than both $\mathsf{rk}(B)$ and $\mathsf{rk}(C)$ then no edge is added for the rule $A \to BC$. To query $A[i]$ where $\mathsf{rk}(A)=k$ we make a case distinction. If $i \leq \lambda_{F_0}(A)+|\mathsf{root}_{F_0}(A)|$ we take a left long step in F_0 to some state $\langle X,j\rangle$ with $\mathsf{rk}(X)<k$ and $j \leq i$, and repeat the procedure from there. Otherwise $i > |\mathsf{root}_{F_0}(A)| > 2^{2^{k-1}} \geq \sqrt{|A|}$ and we query $A[i]$ using short steps in time $\mathcal{O}(\log|A|) \leq \mathcal{O}(\log i)$. Since the rank is reduced in the former case this procedure takes time $\mathcal{O}(\log i + k) \leq \mathcal{O}(\log i + \log\log|A|)$.

We can replace $\log\log|A|$ by $\log^{(t+1)} N$ by adding forests $F_1,\dots,F_{t-1}$ to $\mathcal{F}$: The forest F_k where $1 \leq k \leq t-1$ contains for every rule $A \to BC$ in $\mathcal{G}$ either the edge (A,B), if $|B| > \log^{(k)} N$, or the edge (A,C), if $|B| \leq \log^{(k)} N$ and $|C| > \log^{(k)} N$. To query $A[i]$ we compute the maximal $k \in [0,t-1]$ such that $i \leq \log^{(k)} N$. We will compute the accelerated path in time $\mathcal{O}(\log i + \log^{(k+2)} N) \leq \mathcal{O}(\log i + \log^{(t+1)} N)$. If $|A| \leq \log^{(k)} N$ we can query $A[i]$ in time $\mathcal{O}(\log i + \log\log|A|) \leq \mathcal{O}(\log i + \log^{(k+2)} N)$. If $i \leq \log^{(k)} N < |\mathsf{root}_{F_k}(A)|$ we can take a left long step in F_k and then a short step to some state $\langle X,j\rangle$ where $|X| \leq \log^{(k)} N$ and $j \leq i$. We can query $X[j]$ in time $\mathcal{O}(\log j + \log\log|X|) \leq \mathcal{O}(\log i + \log^{(k+2)} N)$.

Finally, for every forest $F \in \mathcal{F}$ we include a mirrored right-skewed version of F, which then supports access to symbol $A[i]$ in time $\mathcal{O}(\log(|A|-i+1) + \log^{(t+1)} N)$. ◀

Solving the finger search problem. We are ready to prove Theorem 3. We maintain an accelerated path $\pi=(e_1,\dots,e_m)$ from the start variable S to the current finger position f with its left weights and right weights as follows. Let $\ell_j=\sum_{k=1}^{j}\lambda(e_k)$ and $r_j=\sum_{k=1}^{j}\rho(e_k)$ be the prefix sums of the weights. Observe that $f=\ell_m+1$. We store a stack $\gamma=((e_1,\ell_1,r_1),(e_2,\ell_2,r_2),\dots,(e_m,\ell_m,r_m))$, implemented as an array. Given $i \in [1,m]$, one can pop all elements at positions $i+1,\dots,m$ in constant time. We store the set of distinct prefix sums $L=\{\ell_j \mid 0 \leq j \leq m\}$ in a dynamic predecessor data structures from Theorem 14 where a prefix sum ℓ is stored together with the maximal index j such that $\ell=\ell_j$. Similarly $R=\{r_j \mid 0 \leq j \leq m\}$ is stored in a predecessor data structure.

For $\mathsf{setfinger}(f)$ we compute an arbitrary accelerated path from S to $S[f]$, say only using only short steps, and set up the list γ and the predecessor data structures for L and R in time $\mathcal{O}(\log N)$. For $\mathsf{movefinger}(i)$ we can assume that $f-i=d>0$ since the data structures are left-right symmetric. By a predecessor query on L we can find the unique index j with $\ell_j < i \leq \ell_{j+1}$. Then we restrict γ to its prefix of length j, and perform $\mathsf{split}(L,\ell_j)$ and $\mathsf{split}(R,r_j)$, all in constant time. Using fringe access we can compute an accelerated path

π' from A to $S[i] = A[i']$ where $i' = i - \ell_j$: If e_{j+1} is a short edge we take a short step and then use Proposition 16 for the remaining path. If e_{j+1} is a left or right long edge in a forest $F \in \mathcal{F}$ we take a *left* long step, followed by a short step, and then use Proposition 16 for the remaining path. Finally, we update the stack γ and the prefix sums in L and R in time $\mathcal{O}(|\pi'|)$. This concludes the proof of Theorem 3.

We leave it as an open question whether there exists a linear space finger search data structure, supporting $\mathsf{access}(i)$ and $\mathsf{movefinger}(i)$ in $\mathcal{O}(\log d)$ time. For path balanced SLPs such a solution does exist.

▶ **Theorem 17.** *Given an (α, β)-path balanced SLP of size g for a string of length N, one can support* $\mathsf{setfinger}(i)$ *in $\mathcal{O}(\log N)$ time, and* $\mathsf{access}(i)$ *and* $\mathsf{movefinger}(i)$ *in $\mathcal{O}(\log d)$ time, where d is the distance between i and the current finger position, after $\mathcal{O}(g)$ preprocessing time and space.*

References

1. Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-Grained Complexity of Analyzing Compressed Data: Quantifying Improvements over Decompress-and-Solve. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 192–203. IEEE Computer Society, 2017. `doi:10.1109/FOCS.2017.26`.
2. Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger Search in Grammar-Compressed Strings. *Theory Comput. Syst.*, 62(8):1715–1735, 2018. `doi:10.1007/s00224-017-9839-9`.
3. Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random Access to Grammar-Compressed Strings and Trees. *SIAM J. Comput.*, 44(3):513–539, 2015. `doi:10.1137/130936889`.
4. Gerth Stølting Brodal. Finger Search Trees. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. `doi:10.1201/9781420035179.ch11`.
5. Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005. `doi:10.1109/TIT.2005.850116`.
6. Martin Farach and S. Muthukrishnan. Perfect Hashing for Strings: Formalization and Algorithms. In Daniel S. Hirschberg and Eugene W. Myers, editors, *Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Laguna Beach, California, USA, June 10-12, 1996, Proceedings*, volume 1075 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 1996. `doi:10.1007/3-540-61258-0_11`.
7. Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A Faster Grammar-Based Self-index. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications - 6th International Conference, LATA 2012, A Coruña, Spain, March 5-9, 2012. Proceedings*, volume 7183 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2012. `doi:10.1007/978-3-642-28332-1_21`.
8. Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-Based Self-indexing with Faster Pattern Matching. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 731–742. Springer, 2014. `doi:10.1007/978-3-642-54423-1_63`.
9. Moses Ganardi. Compression by Contracting Straight-Line Programs. *CoRR*, abs/2107.00446, 2021. URL: `https://arxiv.org/abs/2107.00446`.
10. Moses Ganardi, Artur Jeż, and Markus Lohrey. Balancing Straight-Line Programs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*

2019, Baltimore, Maryland, USA, November 9-12, 2019, pages 1169–1183. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00073.

11 Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-Based Compression of Unranked Trees. *Theory Comput. Syst.*, 64(1):141–176, 2020. doi:10.1007/s00224-019-09942-y.

12 Leszek Gasieniec, Roman M. Kolpakov, Igor Potapov, and Paul Sant. Real-Time Traversal in Grammar-Based Compressed Files. In *2005 Data Compression Conference (DCC 2005), 29-31 March 2005, Snowbird, UT, USA*, page 458. IEEE Computer Society, 2005. doi:10.1109/DCC.2005.78.

13 Pawel Gawrychowski. Pattern Matching in Lempel-Ziv Compressed Strings: Fast, Simple, and Deterministic. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2011. doi:10.1007/978-3-642-23719-5_36.

14 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted Ancestors in Suffix Trees. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.

15 Tsvi Kopelowitz and Moshe Lewenstein. Dynamic weighted ancestors. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 565–574. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283444.

16 N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

17 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

18 Markus Lohrey, Sebastian Maneth, and Carl Philipp Reh. Constant-Time Tree Traversal and Subtree Equality Check for Grammar-Compressed Trees. *Algorithmica*, 80(7):2082–2105, 2018. doi:10.1007/s00453-017-0331-3.

19 Craig G. Nevill-Manning and Ian H. Witten. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997. doi:10.1613/jair.374.

20 Mihai Patrascu and Mikkel Thorup. Dynamic Integer Sets with Optimal Rank, Select, and Predecessor Search. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 166–175. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.26.

21 Carl Philipp Reh and Kurt Sieber. Navigating Forest Straight-Line Programs in Constant Time. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 11–26. Springer, 2020. doi:10.1007/978-3-030-59212-7_2.

22 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

23 Robert Endre Tarjan. A Class of Algorithms which Require Nonlinear Time to Maintain Disjoint Sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979. doi:10.1016/0022-0000(79)90042-4.

24 Terry A. Welch. A Technique for High-Performance Data Compression. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 17(6):8–19, 1984. doi:10.1109/MC.1984.1659158.

25 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.