Optimal Time Random Access to Grammar-Compressed Strings in Small Space

Patrick Hagge Cording
Technical University of Denmark
phaco@dtu.dk

November 3, 2014

Abstract

The random access problem for compressed strings is to build a data structure that efficiently supports accessing the character in position \( i \) of a string given in compressed form. Given a grammar of size \( n \) compressing a string of size \( N \), we present a data structure using \( O(n\Delta \log \Delta \frac{N}{n} \log N) \) bits of space that supports accessing position \( i \) in \( O(\log \Delta N) \) time for \( \Delta \leq \log^{O(1)} N \). The query time is optimal for polynomially compressible strings, i.e., when \( n = O(N^{1-\epsilon}) \).

1 Introduction

Let \( S \) be a string of length \( N \) compressed into a context-free grammar \( S \) of size \( n \). The random access problem on grammar-compressed strings is to build a data structure that efficiently supports accessing \( S[i] \). In this paper we present a new random access data structure for grammar-compressed strings with a new trade-off between the time and space complexity.

Assume w.l.o.g. that \( S \) is a Straight Line Program (SLP), i.e., a context-free grammar where a rule has either two variables or a terminal on its right-hand side. We regard an SLP as a rooted directed acyclic graph (DAG) with outdegree 2 and we refer to rules as nodes and terminals as leaves when appropriate.

If we store the size of the string generated by each node in the SLP we may access the \( i \)-th character in \( O(h) \) time by doing a top-down search. Here, \( h \) is the height of the grammar and the data structure requires \( O(n) \) space. The concept of balance generalizes from binary trees to SLPs. A balanced SLP has height \( h = O(\log N) \). We may apply the algorithm by Rytter [8] or Charikar et al. [3] to balance the SLP and then access \( S[i] \) in \( O(\log N) \) time. The culprit is that the number of nodes in the SLP resulting from applying either algorithm is \( O(n \log \frac{N}{n}) \). A major breakthrough was done when Bille et al. [2] published a data structure using \( O(n) \) space and supporting access in \( O(\log N) \) time. Their work introduced many new techniques and did not rely on the algorithms by Rytter and Charikar et al. Later, Verbin and Yu [10] proved that if \( n = \Omega(N^{1-\epsilon}) \) any data structure of size \( O(n \log N) \) must use \( \Omega(\log n/\log \log n) \) time to access \( S[i] \). For other compression ratios, they also show a lower bound of \( \Omega(n^{1/2-\epsilon}) \) query time for any \( \epsilon > 0 \) and any data structure of polynomial size. In the wake of this, Belazzougui, Puglisi, and Tabei [1] presented a data structure where \( O(\log_b N) \) access time is achieved using \( O(nb \log \frac{N}{n} \log \log N) \) bits of space which is matching the lower bound by Verbin and Yu for polynomially compressible strings.

Note that for the most recent result by Belazzougui et al., the space bound is in bits and not words of memory. In the remainder of this paper all space bounds will be in bits to comply with their work. Furthermore, all results assume the RAM model of computation with a word size of at least \( \log N \). We present a data structure using \( O(n\Delta \log \Delta \frac{N}{n} \log N) \) bits of space that supports access to \( S[i] \) in \( O(\log \Delta N) \) time. Moreover the time bound optimal for polynomially compressible strings, i.e., \( n = \Theta(N^{1-\epsilon}) \), when we set \( \Delta = \log \frac{N}{n} \).

The key idea behind our data structure is to increase the out-degree of a balanced SLP, exactly as done in the work by Belazzougui et al. This decreases the height of the grammar, and we use the basic top-down search to perform access queries. To get fast navigation in each node we use a data structure known from the fusion tree. Pivotal to our improvement over Belazzougui et al. in terms of space usage is that we exploit some properties of the balanced SLP resulting from applying Rytter’s [8] algorithm.
2 Preliminaries

A string $S = S[1..|S|]$ is a sequence of $|S| = N$ characters. $S[j]$ is a single character.

An SLP $S$ is a context-free grammar in Chomsky normal form that we represent as a node-labeled, directed acyclic graph. Each leaf in $S$ is labelled with a character and corresponds to a terminal grammar production rule. Each internal node in $S$ is labelled with a non-terminal rule from the grammar. We call a node that is reachable from $v$ in one step a child of $v$.

The string $S(v)$ is produced by a depth-first left-to-right traversal from $v$ and consists of the characters on the leaves in the order they are visited. The tree that emerges from this we call the parse tree for $S(v)$. Let $\text{root}(S)$ denote the start rule of $S$, and $\text{left}(v)$ and $\text{right}(v)$ denote the left and right child of an internal node $v \in S$, respectively. The height of a node $v$ is the length of the longest path from $v$ to a leaf reachable from $v$. We denote this by $\text{height}(v)$. We sometimes use $S$ as shorthand for $\text{root}(S)$ when used as parameter in any of the definitions. We say that $v$ has depth $d$ if there exist a path of length $d$ from the root to $v$. Notice that in this definition, $v$ can have multiple depths, and again we denote this simply by $\text{depth}(v)$. $S(v)$ is the subgraph rooted in $v$.

An SLP is height-balanced if $|\text{height}(\text{left}(v)) - \text{height}(\text{right}(v))| \leq 1$ for all $v \in S$. In $[8]$, Rytter presents an approximation algorithm for the smallest grammar problem. The SLP produced by his algorithm is height-balanced and has $O(z \log \frac{N}{z})$ nodes, where $z$ is the number of factors in the LZ77 parse of $S$. Let $n'$ be the size of the smallest grammar. We will use his algorithm to balance an SLP, and since $z \leq n' \leq n$ for any given SLP of size $n$, the number of nodes is $O(n \log \frac{N}{n})$ after applying it.

In addition, to accomplish our goal, we use the predecessor data structure used in nodes in Fredman and Willard’s fusion trees $[6]$, where predecessor queries can be performed in $O(1)$ time if the number of elements in the data structure is at most $w^{1/4}$.

3 Construction of a $\Delta$-SLP

A $\Delta$-SLP is an SLP where we allow each node to have up to $\Delta$ children. We first highlight some properties of Rytter’s $[8]$ AVL grammar, and then describe how to transform an SLP to a $\Delta$-SLP.

3.1 AVL grammar

In this section we highlight the properties of Rytter’s approximation algorithm that we will use. Perhaps the most important property is that the algorithm produces an AVL-balanced SLP. An SLP is AVL-balanced if $|\text{height}(\text{left}(v)) - \text{height}(\text{right}(v))| \leq 1$ for all $v \in S$.

Rytter’s algorithm generates the LZ77 parse of the string and builds an AVL-balanced SLP by processing the factors of the parse left to right. It maintains an AVL-balanced SLP of the string generated by all factors preceeding some factor $f_k$. Then it builds an AVL-balanced grammar of the string generated by $f_k$ by selecting a logarithmic number of non-terminals and joining them. By carefully choosing the order in which these are joined, this step can be done in time linear in the height of the tallest SLP, which is bounded by $O(\log N)$. The SLP for $f_k$ is then joined with the existing grammar.

The algorithm produces an SLP of size $O(z \log \frac{N}{z})$ where $z$ is the size of the greedy LZ77 parse $[4]$. Rytter then shows that $z \leq n'$ where $n'$ is the size of the smallest SLP generating $S$, and we therefore have the following lemma.

Lemma 1. (8) Given an SLP of size $n$ compressing a string of length $N$, we can generate an equivalent SLP of size $O(n \log \frac{N}{n})$, such that the height of the grammar is $O(\log N)$.

We will also use the following two lemmata from Rytter’s paper. One of the key observations in the analysis of Rytter’s algorithm is that the merging of two AVL-balanced SLPs adds only a number of new nodes proportional to the difference in their heights.

Lemma 2. (8) Given two AVL balanced grammars $S_1$ and $S_2$, we can create an AVL balanced grammar $S$ generating the concatenation of the strings generated by $S_1$ and $S_2$ by adding $O(\text{height}(S_1) - \text{height}(S_2))$ new nodes. When doing so, we add a constant number of nodes with depth $d$ for any $1 \leq d \leq \text{height}(S_1) - \text{height}(S_2) + 1$ in $S$.

\footnote{The greedy LZ77 parse is optimal among all LZ77 factorizations; see $[4]$, $[5]$, $[9]$.}
The final lemma is implicit in section 5 of his paper.

**Lemma 3.** Given a grammar produced by Rytter’s approximation algorithm we can select n nodes \( v_1, \ldots, v_n \) such that \( S \) can be written as the concatenation of \( S(v_1), \ldots, S(v_n) \), where for each \( v_i \), \( |S(v_i)| \leq \frac{\Delta}{n} \) and \( \text{height}(v_i) = O(\log \frac{n}{\Delta}) \). Furthermore, \( |\text{depth}(v_i) - \text{depth}(v_j)| \leq 1 \) for any \( 1 \leq i, j \leq n \).

### 3.2 From SLP to \( \Delta \)-SLP

We now show how to construct a \( \Delta \)-SLP from an SLP.

**Lemma 4.** Given an SLP of size \( n \) with height \( h \), we can produce a \( \Delta \)-SLP that generates the same string and has \( O(n) \) nodes, \( O(n \cdot \Delta) \) edges, and height \( O(h / \log \Delta) \), for any \( 2 \leq \Delta \leq N \).

**Proof.** Our recursive algorithm works as follows. Let \( \text{root}(S) \) be the root in the \( \Delta \)-SLP. Let \( v_1, \ldots, v_k \) be all nodes that are at depth at most \( i \) from \( \text{root}(S) \) in \( S \). Clearly, there can be at most \( 2^i \) of those. Attach these nodes directly to \( \text{root}(S) \) and recurse on each. The new grammar has height \( O(h / \log \Delta) \) since \( S \) has height \( O(h) \). \( \square \)

If we apply the construction of Lemma 4 to an AVL balanced grammar produced by Rytter’s algorithm we obtain a \( \Delta \)-SLP with \( O(n \log \frac{n}{\Delta}) \) nodes, \( O(\Delta \cdot n \log \frac{n}{\Delta}) \) edges, and height \( O(\log \Delta \cdot N) \). In fact, by careful analysis, this can be improved.

**Lemma 5.** Let \( S \) be an SLP produced by Lemma 4 and let \( 1 \leq d \leq \text{height}(S) \), then the number of non-terminals at depth \( d \) in \( S \) is bounded by \( O(n) \).

**Proof.** The key observation needed for this proof is Lemma 2. First, recall that Rytter’s algorithm processes the LZ77 factorization \( f_1, \ldots, f_s \) of \( S \) from left to right, such that, before processing \( f_i \), we have an AVL-balanced grammar \( G_{i-1} \) for the string produced by \( f_1, \ldots, f_{i-1} \). We refer to the processing of \( f_i \) as round \( i \) of the algorithm.

We will prove that the number of nodes added to depth \( d \) in each round is constant. Rytter’s algorithm picks \( v_1, \ldots, v_j, j = O(\log N) \), from \( G_{i-1} \) such that the concatenation of \( S(v_1), \ldots, S(v_j) \) generates the string represented by \( f_i \). For some \( k \), it holds that \( \text{height}(v_1) \leq \text{height}(v_2) \leq \ldots \leq \text{height}(v_k) \) and \( \text{height}(v_k) \geq \text{height}(v_{k+1}) \geq \ldots \geq \text{height}(v_j) \). In addition, it holds that if \( \text{height}(v_i) = \text{height}(v_{i+1}) \) then \( \text{height}(v_{i+1}) \neq \text{height}(v_{i+2}) \). The algorithm joins the graphs from lowest to tallest, i.e., we first join \( v_1 \) with \( v_2 \) up to \( v_k \) and then \( v_j \) with \( v_{j-1} \) down to \( v_k \).

We will prove by induction that joining \( v_1 \) to \( v_k \) only adds a constant number of nodes with depth \( d \) for any \( d \). For the base case we have that \( \text{height}(v_1) \leq \text{height}(v_2) \). If \( \text{height}(v_1) < \text{height}(v_2) \) we know from Lemma 2 that the number of nodes added at each depth is \( O(1) \). If \( \text{height}(v_1) = \text{height}(v_2) \) the two are joined by adding at new node with two children, and clearly only one new node is added.

For the inductive step, let \( \mathcal{L} \) be the result of joining \( v_1, \ldots, v_{i-1} \). It follows from the aforementioned properties that \( \text{height}(\mathcal{L}) \leq \text{height}(v_i) + 1 \). If \( \text{height}(\mathcal{L}) < \text{height}(v_i) \) then we will add a constant number of nodes with depth \( d \) for \( 1 \leq d \leq \text{height}(v_i) - \text{height}(\mathcal{L}) + 1 \). No new nodes were previously added to \( v_i \) and the induction hypothesis says that only \( O(1) \) nodes with depth \( \text{height}(\mathcal{L}) \leq d' \leq 1 \) were added to \( \mathcal{L} \), so only \( O(1) \) nodes are added at any depth in the resulting grammar. If \( \text{height}(\mathcal{L}) \geq \text{height}(v_i) \) then we add one new node as a root.

We know from Lemma 2 that when joining \( G_{i-1} \) with the grammar obtained from joining \( v_1, \ldots, v_j \) we add only a constant number of new nodes at each depth. Since there are \( z \leq n \) factors we conclude that the number of nodes added at each depth of the grammar produced by Rytter’s algorithm is \( O(n) \). \( \square \)

By combining the construction of Lemma 4 and the above lemma, we get the following result.

**Corollary 1.** Given an AVL balanced grammar produced by Rytter’s algorithm. The construction of Lemma 4 produces a \( \Delta \)-SLP with \( O(n \log \Delta \cdot N) \) nodes, \( O(\Delta \cdot n \log \Delta \cdot N) \) edges, and height \( O(\log \Delta \cdot N) \).

**Proof.** First note that the number of terminals in a grammar produced by Rytter’s algorithm is at most \( n \) and this is also the case for the \( \Delta \)-SLP. Consider a set of non-terminals at depth \( d \) that have been picked as nodes in the \( \Delta \)-SLP. From these we pick another set of nodes that are of at most depth \( \log \Delta \) from the nodes in the first set. If we pick a node of depth less than \( d + \log \Delta \), then this is a leaf, and there are at most \( n \) of these. If we pick a node at \( d + \log \Delta \) then we know from Lemma 5 that there are \( O(n) \) distinct nodes at this depth. In total, we select \( O(n) \) for every level of the \( \Delta \)-SLP and therefore the total number of nodes is \( O(n \log \Delta \cdot N) \). \( \square \)
4 Access data structure

Assume that we are given a grammar produced by Rytter’s algorithm, and let $v_1, \ldots, v_n$ be the nodes specified by Lemma 3. We store a predecessor data structure containing the values $\sum_{j=1}^i |S(v_j)|$ for each $v_i$ (using the $y$-fast trie due to Willard [11]). Then we construct a $\Delta$-SLP starting from the nodes $v_1, \ldots, v_n$. For each node $v$ in the $\Delta$-SLP we store the values $\sum_{j=1}^i |S(c_j)|$, where $c_1, c_2, \ldots, c_k, k \leq \Delta$, are the children of $v$, and a constant time predecessor data structure [6] (see also [7]) over these.

To access $S[i]$ we first use the $y$-fast trie to find the node $v_k$ among $v_1, \ldots, v_n$ to start the traversal of the $\Delta$-SLP from. Assume that we are at node $v_k$ and we know that the substring it generates starts at position $j$. We then query for the predecessor of $i - j$ in the local predecessor data structure of $v_k$ to find the child of $v_k$ to continue the search from. We continue doing this until we reach a terminal and the label of this is the answer to the query.

Theorem 1. Our data structure uses $O(n \Delta \log \frac{N}{n} \log N)$ bits of space and supports random access to a string compressed by an SLP in $O(\log \Delta N)$ time.

Proof. The predecessor data structure over the accumulated strings generated by the nodes $v_1, \ldots, v_n$ requires $O(n \log N)$ space. From Lemma 3 we know that the number of nodes that are kept in the $\Delta$-SLP at some depth $d$ from the root is $O(n)$. Since the depth of the starting nodes $v_1, \ldots, v_n$ differ by at most 1, every level in the $\Delta$-SLP has nodes from at most two levels in the balanced SLP, and therefore the $\Delta$-SLP has at most $O(n \log \frac{N}{n})$ nodes (requiring $O(n \log \frac{N}{n} \log N)$ bits to represent). Storing the accumulated sums of string lengths and the constant time predecessor data structure local to $v$ uses $O(\Delta \log N)$ bits per node totalling $O(n \Delta \log \frac{N}{n} \log N)$ bits. Finally, we need to store the children of all nodes which requires a total of $O(n \log \frac{N}{n} \cdot \Delta \log (n \log \frac{N}{n}))$ bits.

When performing a query we first spend $O(\log \log N)$ time for the first predecessor query and then we traverse the grammar using $O(1)$ time per $O(\log \Delta \frac{N}{n})$ levels. Thus, the query time is bounded by $O(\log \Delta N)$.

By setting $\Delta = \log^t N$ our data structure uses $O(n \log \frac{N}{n} \cdot \frac{\log^{1+\epsilon} N}{\log \log N})$ bits of space and the query time becomes $O(\log N / \log \log N)$.

5 Concluding remarks

We have presented a random access data structure with optimal query time for polynomially compressible strings. In turn, our data structure uses superlinear space and is not optimal in terms of space. Several interesting questions remain open:

- Is there a data structure with optimal $O(n^{1/2-\epsilon})$ query time?
- Is there a linear space data structure with optimal query time $O(\log N / \log \log N)$ for polynomially compressible strings?
- Can we construct a data structure with a time-space product of $n \log N$? That is, a data structure that offers the same space and time bounds as Bille et al. [2] as one extreme and optimal query time, i.e., $O(\log N / \log \log N)$ time and $O(n \log \log N)$ words of space, as the other.

Acknowledgements

The author thanks Noy Rotbart, Pawel Gawrychowski, Philip Bille, and Søren Vind for helpful discussions.

References

[1] D. Belazzougui, S. J. Puglisi, and Y. Tabei. Rank, select and access in grammar-compressed strings. arXiv preprint arXiv:1408.3093, 2014.
[2] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings. In Proc. 22nd SODA, pages 373–389. SIAM, 2011.

[3] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. IEEE Trans. Inf. Theory, 51(7):2554–2576, 2005.

[4] M. Cohn and R. Khazan. Parsing with suffix and prefix dictionaries. In Proc. 6th DCC, pages 180–180, 1996.

[5] M. Crochemore, A. Languila, and F. Mignosi. Note on the greedy parsing optimality for dictionary-based text compression. Theoret. Comp. Sci., 525(0):55 – 59, 2014.

[6] M. L. Fredman and D. E. Willard. Blasting through the information theoretic barrier with fusion trees. In Proc. 22nd STOC, pages 1–7, 1990.

[7] M. Patrascu and M. Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. arXiv preprint arXiv:1408.3045, 2014.

[8] W. Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. Theoret. Comput. Sci., 302(1):211–222, 2003.

[9] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. J. ACM, 29(4):928–951, 1982.

[10] E. Verbin and W. Yu. Data structure lower bounds on random access to grammar-compressed strings. In Proc. 24th CPM, pages 247–258, 2013.

[11] D. E. Willard. Log-logarithmic worst-case range queries are possible in space Θ(N). Inf. Proc. Let., 17(2):81–84, 1983.

[12] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. Trans. Inf. Theory, 23(3):337–343, 1977.