Random Access in Grammar-Compressed Strings

Optimal Trade-Offs in Almost All Parameter Regimes

Anouk Duyster
Max Planck Institute for Informatics, SIC, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Sciences, SIC, Saarbrücken, Germany

Tomasz Kociumaka
Max Planck Institute for Informatics, SIC, Saarbrücken, Germany

Abstract

A Random Access query to a string T asks for the character $T[i]$ at a given position $i \in [0..|T|)$. This fundamental task admits a straightforward solution with constant-time queries and $\mathcal{O}(n \log \sigma)$ bits of space when $T \in [0..\sigma)^n$. While this is the best one can achieve in the worst case, much research has focused on the compressed setting: if T is compressible, one can hope for a much smaller data structure that still answers Random Access queries efficiently.

In this work, we investigate the grammar-compressed setting, where T is represented by a context-free grammar that produces only T. Our main result is a general trade-off that optimizes Random Access time as a function of the string length n, the grammar size (the total length of productions) g, the alphabet size σ, the data structure size M, and the word size $w \geq \Omega(\log n)$ of the word RAM model. For any data structure size M satisfying $g \log n < Mw < n \log \sigma$, we show an $\mathcal{O}(M)$-size data structure that answers Random Access queries in time

$$\mathcal{O}\left(\frac{\log \frac{n \log \sigma}{Mw}}{\log \frac{Mw}{g \log n}}\right).$$

We also prove a matching unconditional lower bound that holds for all parameter regimes except very small grammars ($g \leq w^{1+o(1)} \log n$) and relatively small data structures ($Mw \leq g \log n \cdot w^{o(1)}$). The lower bound applies to word-RAM query time and, more strongly, to the worst-case cell-probe complexity of nondeterministic or bounded-error randomized query algorithms.

Previous work focused on optimizing the query time as a function of n only, achieving $\mathcal{O}(\log n)$ time using $\mathcal{O}(g)$ space [Bille, Landau, Raman, Sadakane, Satti, Weimann; SIAM J. Comput. 2015] and $\mathcal{O}(\frac{\log n}{\log \log n})$ time using $\mathcal{O}(g \log^{\epsilon} n)$ space for any constant $\epsilon > 0$ [Belazzougui, Cording, Puglisi, Tabei; ESA 2015], [Ganardi, Jeż, Lohrey; J. ACM 2021]. Our result improves upon these bounds (strictly for $g = n^{1-o(1)}$) and generalizes them beyond $M \leq \mathcal{O}(g \operatorname{polylog} n)$, yielding a smooth interpolation with the uncompressed setting of $Mw = n \log \sigma$ bits.

Thus far, the only tight lower bound [Verbin and Yu; CPM 2013] was $\Omega(\frac{\log n}{\log \log n})$ for $w = \Theta(\log n)$, $n^{\Omega(1)} \leq g \leq n^{1-\Omega(1)}$, and $M = g \cdot \log^{\Theta(1)} n$. In contrast, our result yields a tight bound that accounts for all relevant parameters and is valid for almost all parameter regimes.

Our bounds remain valid for run-length grammars, where production sizes use run-length encoding. This lets us recover (and, for strings with small run-length grammars, improve) the trade-offs achieved by block trees, formulated in terms of the LZ77 size z [Belazzougui, Cáceres, Gagie, Gawrychowski, Kärkkäinen, Navarro, Ordóñez, Puglisi, Tabei; J. Comput. Syst. Sci. 2021] and substring complexity δ [Kociumaka, Navarro, Prezza; IEEE Trans. Inf. Theory 2023].

Our data structure admits an efficient deterministic construction algorithm. Beyond Random Access, its variants also support substring extraction (with optimal additive overhead $\mathcal{O}(\frac{m \log \sigma}{w})$ for a length-m substring, provided that $M \geq g$), as well as rank and select queries.

All our results rely on novel grammar transformations that generalize contracting grammars [Ganardi; ESA 2021] and achieve the optimal trade-off between grammar size and height while enforcing extra structure crucial for constant-time navigation in the parse tree.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Cell probe models and lower bounds

Keywords and phrases grammar-based compression, straight-line programs, random access problem

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 86; pp. 86:1–86:25

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.86

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2602.10864v2 [18]

Acknowledgements The authors thank an anonymous reviewer for exceptionally careful reading.

1 Introduction

Random Access is arguably the most basic query one can ask about a string and a fundamental building block of classic string-processing algorithms. For a string $T \in \Sigma^*$ and a position $i \in [0..|T|)$, it asks for the i-th character $T[i] \in \Sigma$ of T.[1] In the standard representation of strings as arrays, Random Access trivially takes constant time. For strings over integer alphabets $\Sigma = [0..\sigma)$, constant-time Random Access can still be supported using a *packed* representation in $\Theta(n \log \sigma)$ bits thanks to the bit-manipulation operations available in the standard word RAM model. Up to a constant factor, this simple data structure matches the information-theoretic lower bound of $\lceil n \log \sigma \rceil$ bits.[2] More advanced tools [15] come even closer, achieving $n \log \sigma + \mathcal{O}(\log^2 n)$ bits.

The complexity landscape of Random Access queries becomes much more interesting in the *compressed setting*, where we aim to achieve much smaller data structures for some "compressible" strings at the price of marginally larger data structures for the remaining "incompressible" strings. Every injective function $C : \Sigma^* \to \{0,1\}^*$ can be seen as a compression method, but few such functions admit an efficient query algorithm for Random Access. In particular, the most popular compressors such as arithmetic coding [27], Lempel–Ziv parsing [53], or run-length encoded Burrows–Wheeler transform [8] do not natively support Random Access in $n^{o(1)}$ time. Since Random Access is needed for most string-processing algorithms, a lot of research has been devoted to understanding how much extra space is needed to enable such support. This question formally asks what trade-offs are possible between the data structure size and its Random Access query time, where both complexities are measured not only in terms of the length n and alphabet size σ of the input string T but also a parameter $s(T)$ capturing the size of its compressed representation.

In this work, we focus on the so-called highly-compressible regime, where typical space complexities are of the form $\mathcal{O}(s(T))$ or $\mathcal{O}(s(T) \operatorname{poly} \log n)$ rather than $s(T) + o(n)$. In this setting, a clean formalization that captures many compressors is grammar-compression [46, 32]: the input string T is encoded using a context-free grammar $\mathcal{G}$ that produces T and no other string. Such a grammar is a *straight-line grammar* (SLG) because it cannot use circular dependencies between variables or define multiple productions for the same variable. The underlying compressibility measure, denoted $g(T)$, is the size (total production length) of the smallest grammar generating T. A long line of works has aimed to understand the complexity of Random Access in SLG-compressed strings:

SLG Random Access
Parameters: $n, g, \sigma \in \mathbb{Z}_{>0}$
Input: A string $T \in [0..\sigma)^n$ represented by an SLG of size at most g.
Queries: Given a position $i \in [0..n)$, return $T[i]$.

[1] We adhere to 0-based indexing of strings, that is, $T = T[0] \cdots T[|T| - 1]$. For $a, b \in \mathbb{R}$, we denote $[a..b] = \{k \in \mathbb{Z} : a \le k \le b\}$ and $(a..b) = \{k \in \mathbb{Z} : a < k < b\}$. Half-open integer intervals $[a..b)$ and $(a..b]$ are defined analogously.

[2] All logarithms in this work are binary unless another base is provided.

Time-space trade-offs for this problem are inherently related to the trade-offs between the grammar height (the height of the underlying parse tree, i.e., the worst-case number of productions needed to derive a given character of T) and its size. In 2002, parallel works of Charikar et al. [9] and Rytter [47] showed that if T is produced by an arbitrary SLG of size g, it is also produced by a *balanced* SLG of height $\mathcal{O}(\log n)$ and size $\mathcal{O}(g \log \frac{n}{g})$. This immediately implies $\mathcal{O}(\log n)$-time Random Access using a data structure of size $\mathcal{O}(g \log \frac{n}{g})$.[3] A decade later, a breakthrough of Bille et al. [7] achieved $\mathcal{O}(\log n)$-time Random Access using $\mathcal{O}(g)$ space. On the lower-bound side, an influential work of Verbin and Yu [51] showed that an $\mathcal{O}(g \operatorname{polylog} n)$-space data structure cannot support $o(\frac{\log n}{\log\log n})$-time Random Access in their hard regime. Shortly afterward, Belazzougui et al. [5] proved that $\mathcal{O}(\frac{\log n}{\log\log n})$ query time is achievable in $\mathcal{O}(g \log^{\epsilon} n)$ space for balanced SLGs and any constant $\epsilon > 0$. In a recent milestone, Ganardi, Jeż, and Lohrey [24] showed every SLG can be balanced while keeping its size at $\mathcal{O}(g)$; this immediately recovers the trade-off of [7] and proves that the trade-off of [5] is valid for every SLG. More generally, one can answer Random Access queries in $\mathcal{O}(\log_{\tau} n)$ time using $\mathcal{O}(g\tau)$ space for $2 \le \tau \le \operatorname{polylog} n$.

A common limitation of all these previous works is that they measure the Random Access time as a function of n only. The lower bound of [51] is only valid when $n^{\Omega(1)} \le g \le n^{1-\Omega(1)}$,[4] so we can hope for faster queries for the cases of $g \le n^{o(1)}$ and, much more importantly for applications, $g \ge n^{1-o(1)}$. Moreover, neither the lower bound nor the upper bound applies to data structures of size exceeding $\mathcal{O}(g \operatorname{polylog} n)$. In this work, we aim to characterize what trade-offs are possible.

> *What is the optimal Random Access time in grammar-compressed strings, as a function of string length n, grammar size g, alphabet size σ, and data structure size M?*

On the positive side, we show that, for any $\tau \ge 2$, Random Access queries can be answered in $\mathcal{O}(\log_{\tau} \frac{n \log \sigma}{g\tau \log n})$ time[5] using $\mathcal{O}(g\tau)$ space. Phrased in terms of M rather than τ, this reads as follows:

▶ **Theorem 1.1.** *Let $\mathcal{G}$ be an SLG of size g generating a string $T \in [0..\sigma)^n$. In the word RAM model with word size $w \ge \Omega(\log(n\sigma))$, given $\mathcal{G}$ and any value M with $g \log n < Mw < n \log \sigma$, one can in $\mathcal{O}(\frac{Mw}{\log n})$ time construct an $\mathcal{O}(M)$-size data structure supporting Random Access queries in time*

$$\mathcal{O}\left(\frac{\log \frac{n \log \sigma}{Mw}}{\log \frac{Mw}{g \log n}}\right).$$

Already in the regime of $\mathcal{O}(g)$ space, we improve the query time from $\mathcal{O}(\log n)$ to $\mathcal{O}(\log \frac{n \log \sigma}{g \log n})$. Our time bound smoothly interpolates between logarithmic for $g \le n^{1-\Omega(1)}$ and constant for $g \ge \Omega(\frac{n \log \sigma}{\log n})$. Since every string can be produced by an SLG of size $g \le \mathcal{O}(\frac{n \log \sigma}{\log n})$, we recover the standard uncompressed bound: $\mathcal{O}(1)$ time using $\mathcal{O}(n \log \sigma)$ bits. Allowing more space, we generalize the trade-off of [5] beyond $\tau \le \operatorname{polylog} n$ and strictly improve upon it when $g \ge n^{1-o(1)}$. In particular, we get $\mathcal{O}(1)$ query time already using $\mathcal{O}((g \log n)^{1-\epsilon}(n \log \sigma)^{\epsilon})$ bits for a constant $\epsilon > 0$.

---

[3] We state all results in the word RAM model. In the introduction, we typically use word size $w = \Theta(\log(n\sigma))$, and we measure space complexities in machine words. Later on, we switch to bits and treat w as a separate parameter.

[4] Verbin and Yu [51] also prove a separate lower bound for a particular value $g \le \log^{2+o(1)} n$, which depends on n and the data structure size $M \le n^{o(1)}$. That result only shows that the query time is at least $(\log n)^{1-o(1)} / \log M$.

[5] In the introduction, we follow the convention that time complexity $\mathcal{O}(t)$ should be read as $\mathcal{O}(\max(1, t))$.

On the lower-bound side, our main result is that the query time of Theorem 1.1 is optimal for all parameter regimes except for very small grammars and relatively small data structures.

▶ **Theorem 1.2.** *Consider integers $n, g, \sigma, M, w, t \in \mathbb{Z}_{>0}$ and a real constant $\epsilon > 0$ such that $n \geq g$, $n \geq \sigma \geq 2$, $Mw \geq g \cdot \log n \cdot (w \log n)^{\epsilon}$, and $g \geq 25 \cdot w^{1+\epsilon} \cdot \log n$. Suppose that, for every instance of SLG Random Access with parameters n, g, σ, there is a data structure of M w-bit machine words that answers each query by accessing t of these machine words. Then,*

$$t \geq \Omega\left(\frac{\log \frac{n \log \sigma}{Mw}}{\log \frac{Mw}{g \log n}}\right).$$

This bound remains valid for nondeterministic and randomized data structures with two-sided error.

In the most studied setting of $w = \Theta(\log n)$, our tight lower bound applies to all grammar sizes except $g \leq \log^{2+o(1)} n$ and all data structure sizes except $M \leq g \log^{o(1)} n$. As noted already in [51], $\log^2 n$ is a natural barrier: if $g \leq \log^{2-\delta} n$ for some constant $\delta > 0$, then the query algorithm could read the entire grammar using $\log^{1-\delta+o(1)} n$ memory accesses. The limitation $M \leq g \log^{o(1)} n$ is shared with most static data-structure lower bounds, which typically capture space complexity only up to sub-logarithmic factors, except for a handful of isolated problems; see [35]. Whether $o(\log n)$ query time can be achieved in $\mathcal{O}(g)$ space remains a long-standing open problem.

Beyond SLGs. In [10, 39], the $\mathcal{O}(\log n)$-time $\mathcal{O}(g)$-space trade-off for SLG Random Access [7, 24] has been generalized to run-length straight-line grammars (RLSLGs) [40], which differ from SLGs in that production sizes are measured in terms of run-length encoding size rather than length. Equivalently, in RLSLGs, productions of the form $A \to B^k$ for $k \in \mathbb{Z}_{>0}$ are assumed to be of size one. The size of the smallest RLSLG generating T satisfies $g_{\mathsf{rle}}(T) \leq g(T)$, so RLSLG Random Access is at least as hard as SLG Random Access, and the lower bound of Theorem 1.2 generalizes automatically. Even though we are not aware of any previous work generalizing the trade-off of [5] to RLSLGs, in [18, Section 6] we state and prove our upper bound of Theorem 1.1 already in the stronger RLSLG variant (without any penalty).

A notable advantage of $g_{\mathsf{rle}}(T)$ is that it is a tighter upper bound of further measures, including the LZ77 size $z(T)$ [53] and the substring complexity $\delta(T)$ [45, 33], which satisfy $\delta(T) \leq z(T) \leq g_{\mathsf{rle}}(T) \leq g(T)$. In particular, $g_{\mathsf{rle}}(T) \leq \mathcal{O}(\delta(T) \log \frac{n \log \sigma}{\delta(T) \log n}) \leq \mathcal{O}(z(T) \log \frac{n \log \sigma}{z(T) \log n})$, and the first of these bounds fails for $g(T)$ [33]. Thanks to this characterization of $g_{\mathsf{rle}}(T)$, Theorem 1.1 recovers the Random Access trade-offs of *block trees* formulated in [4, 33] using $z(T)$ and $\delta(T)$, respectively. Moreover, we derive from Theorem 1.2 that these trade-offs are optimal, as functions of $z(T)$ and $\delta(T)$, respectively, in a wide range of parameter regimes.

▶ **Corollary 1.3** (see [18, Section 9])**.** *For every string $T \in [0..\sigma)^n$ and all parameters $d \geq \delta(T)$, $\tau \geq 2$, and $d\tau \leq s \leq d\tau \log_\tau \frac{n \log \sigma}{d \log n}$, there is a data structure of size $\mathcal{O}(s + d\tau \log_\tau \frac{n \log \sigma}{s \log n})$ supporting Random Access queries in time $\mathcal{O}(\log_\tau \frac{n \log \sigma}{s \log n})$. This query time is optimal if $n \geq \sigma \geq 2$, $\tau \geq \log^{\epsilon} n$, and $d \geq \log^{2+\epsilon} n$ for a constant $\epsilon > 0$, even restricted to strings satisfying $d \geq g(T) \geq z(T) \geq \delta(T)$.*

Beyond Random Access. Many previous data structures for (RL)SLG Random Access support extracting not only individual characters but also substrings of T of arbitrary length m, with the additive time overhead of $\mathcal{O}(m)$ [7, 10, 39] or, if the substring is returned in the

packed representation, $\mathcal{O}(m/\log_\sigma n)$ [5] or $\mathcal{O}(m/\log_\sigma n \cdot \log_\tau \frac{n\log\sigma}{s\log n})$ [4, 33]. We generalize Theorem 1.1 to achieve the optimal overhead of $\mathcal{O}(m/\log_\sigma n)$. Moreover, ours is the first solution that outputs the packed representation and comes with an efficient construction.

▶ **Theorem 1.4.** *Let $\mathcal{G}$ be an RLSLG of size g generating a string $T \in [0..\sigma)^n$. In the word RAM model with word size $w \geq \Omega(\log(n\sigma))$, given $\mathcal{G}$ and any value M with $g\log n < Mw < n\log\sigma$, one can in $\mathcal{O}(\frac{Mw}{\log n})$ time construct an $\mathcal{O}(M)$-size data structure that extracts any length-m substring in time*

$$\mathcal{O}\left(\frac{\log\frac{n\log\sigma}{Mw}}{\log\frac{Mw}{g\log n}} + \frac{m\log\sigma}{w}\cdot\frac{M+g}{M}\right).$$

More generally, we support fast character iterators that, upon initialization at position i, can in constant time move in either direction by $\mathcal{O}(\log_\sigma n)$ positions and output the characters at the intermediate positions. None of the previous works achieves this without amortization for $\sigma \leq n^{o(1)}$.

Our approach readily generalizes to computing various aggregate information about the prefixes of T, modeled by the prefix sum problem in which every character $a \in \Sigma$ is mapped to an element $\Phi(a)$ of a monoid. In particular, similarly to [7, 5, 24, 39, 10, 33], we support rank and select queries, asking for the number of occurrences of a given character $a \in \Sigma$ in the prefix $T[0..i)$ and the position of the r-th leftmost occurrence of a in T, respectively.

▶ **Theorem 1.5.** *Let $\mathcal{G}$ be an RLSLG of size g generating a string $T \in [0..\sigma)^n$. In the word RAM model with $w \geq \Omega(\log(n\sigma))$, given $\mathcal{G}$ and any value M with $g\log n < Mw < n$, one can in $\mathcal{O}(\frac{Mw\sigma}{\log n} + g\sigma\log n)$ time construct an $\mathcal{O}(M\sigma)$-size data structure supporting rank_T and select_T queries in time*

$$\mathcal{O}\left(\frac{\log\frac{n}{Mw}}{\log\frac{Mw}{g\log n}}\right).$$

For constant alphabet size σ, this result matches our time-space trade-off for Random Access. It is also straightforward to see that, already for $\sigma = 2$, rank and select queries are at least as hard as Random Access,[6] so our bounds remain tight for almost all parameter regimes. The only penalty that we suffer already for $\sigma \leq \mathcal{O}(1)$ is the extra $\mathcal{O}(g\log n)$ term in the construction algorithm. As we discuss in [18, Section 8], this term arises only for select, and only when $g \geq n/\log^{1+o(1)} n$. It is an artifact of our usage of Elias–Fano encoding [19, 20], which currently lacks an efficient construction from the packed input representation using variable-length gap encoding.

In general, similarly to all previous data structures in grammar-compressed space, ours suffers from a σ-factor overhead in space complexity (and construction time). Reducing this overhead, possibly at the cost of slight increase in the query time, remains a major open question. As noted in [5, Section 6], this would require breakthroughs for counting paths between DAG vertices.

[6] For rank, note that $T[i] = 1$ if and only if $\mathsf{rank}_{T,1}(i) < \mathsf{rank}_{T,1}(i+1)$. For select, we map T through a morphism $0 \mapsto 10$ and $1 \mapsto 01$, preserving n and g up to a constant factor. The resulting string T' satisfies $\mathsf{select}_{T',1}(i) = 2i + T[i]$.

1.1 Our Techniques

We prove Theorems 1.1, 1.4, and 1.5 by converting the input (RL)SLG into an equivalent *shallow* grammar that admits fast *navigation* in its parse tree. Let $\mathcal{G}$ be an (RL)SLG and let $A \in \mathcal{A}_\mathcal{G}$ be a symbol (variable or terminal). We write $\mathsf{rhs}_\mathcal{G}(A)$ for the right-hand side of its production (a string over $\mathcal{A}_\mathcal{G}$) and $\mathsf{expand}_\mathcal{G}(A)$ for its full expansion (the unique string over terminals derived from A).

Random Access can be implemented recursively. Suppose A is a variable with production $A \to B_0 \cdots B_{k-1}$, and we are asked to retrieve $\mathsf{expand}_\mathcal{G}(A)[i]$. Then there is a unique $j \in [0..k)$ such that $|\mathsf{expand}_\mathcal{G}(B_0 \cdots B_{j-1})| \le i < |\mathsf{expand}_\mathcal{G}(B_0 \cdots B_j)|$, and the query recurses on the symbol B_j and the shifted index $i - |\mathsf{expand}_\mathcal{G}(B_0 \cdots B_{j-1})|$. We denote the task of finding this j by child_A. It is a rank query over the prefix sums $|\mathsf{expand}_\mathcal{G}(B_0 \cdots B_j)|$, asking for the number of such sums that are at most i.

Grammar balancing [24] yields grammars of height $\mathcal{O}(\log n)$, size $\mathcal{O}(g)$, and production lengths $k \le 2$, leading to $\mathcal{O}(\log n)$-time Random Access in $\mathcal{O}(g)$ space. To reduce the query time to $\mathcal{O}(\log \frac{n}{g})$, we would like to decrease the height to $\mathcal{O}(\log \frac{n}{g})$ by creating a new length-$\Theta(g)$ right-hand side of the starting symbol S. This is where our first obstacle appears: even if every variable A has recursion depth $\mathcal{O}(\log |\mathsf{expand}_\mathcal{G}(A)|)$, a natural process starting with S and exhaustively replacing all symbols longer than $\frac{n}{g}$ by their right-hand sides may create a sequence of $\Theta(g \log n)$ symbols, exceeding $\mathcal{O}(g)$ space. We avoid this by working with *contracting* grammars [23], where a variable B can appear on the right-hand side of a variable A only if $|\mathsf{expand}_\mathcal{G}(B)| \le \frac{1}{2}|\mathsf{expand}_\mathcal{G}(A)|$. We show in [18, Lemma 5.11] how this implies that the same process produces a sequence of $\mathcal{O}(g)$ symbols of length at most $\frac{n}{g}$, thus reducing child_S to a rank query on an $\mathcal{O}(g)$-element set of prefix sums. This query can then be implemented in $\mathcal{O}(\log \frac{n}{g})$ time [7] or even in $\mathcal{O}(\log\log \frac{n}{g})$ time [3].

For a more general trade-off with $\mathcal{O}(\log_\tau \frac{n}{g})$ query time, we want to similarly increase the fan-out of every other variable A to $\Theta(\tau)$ so that the height becomes $\mathcal{O}(\log_\tau \frac{n}{g}) \le \mathcal{O}(1 + \log_\tau \frac{n}{g\tau})$. Here, an obstacle is that we have to implement child_A in constant time in order to avoid extra factors in Random Access time. This is possible for $\tau \le \operatorname{polylog} n$ using fusion trees [21], which is why the previous trade-offs [5] were limited to small τ. Our key structural notion is τ-*niceness* ([18, Definition 5.10]): as for the start symbol, we replace $\mathsf{rhs}_\mathcal{G}(A)$ by a longer right-hand side $\mathsf{rhs}_\tau(A)$ of $\mathcal{O}(\tau)$ symbols of length at most $\frac{1}{\tau}|\mathsf{expand}_\mathcal{G}(A)|$ each. Crucially (see [18, Lemma 5.11]), the contracting property of $\mathcal{G}$ implies a strong *locality* condition: any substring of $\mathsf{expand}_\mathcal{G}(A)$ of length at most $\frac{1}{\tau}|\mathsf{expand}_\mathcal{G}(A)|$ intersects only $\mathcal{O}(\log \tau)$ consecutive symbols of $\mathsf{rhs}_\tau(A)$. Thus, by storing the answers to $\Theta(\tau)$ evenly spaced rank queries, we reduce every child_A to a rank query in a set of size $\mathcal{O}(\log \tau)$, which we answer in constant time using fusion trees (see [18, Lemma 6.1]). Combining this with a standard shortcut (storing $\mathsf{expand}_\mathcal{G}(A)$ explicitly whenever $|\mathsf{expand}_\mathcal{G}(A)| \le \log_\sigma n$) yields the trade-off of Theorem 1.1. The approach extends to RLSLGs once we generalize the transformation of [23] appropriately ([18, Corollary 5.8]; a parallel work [11] achieves almost the same generalization).

Substring Extraction. To extract any substring of T with constant delay per character using an $\mathcal{O}(g)$-size data structure, we could simply use the character iterators of [25] or [37], with slight adaptations to support RLSLGs and have construction time matching our Random Access. Outputting blocks of $b = \Theta(\log_\sigma n)$ characters with constant delay is much more challenging; see [5]. The underlying issue is that a long variable A may contain a very short child B (even of constant expansion length), so a character-by-character traversal cannot be turned into a block-by-block traversal by local shortcuts alone. We address this by converting

$\mathcal{G}$ into a *b-leafy* grammar: *leaf variables* have expansion length $\Theta(b)$, while the remaining *top variables* have constant-size right-hand sides over top and leaf variables only. Treating leaf variables as terminals yields a grammar for a shorter string of length $\Theta(n/b)$, whose characters represent blocks of $\Theta(b)$ characters of T. Hence, standard character iterators can output one leaf per step, i.e., $\Theta(b)$ characters at a time. To preserve lengths under this reinterpretation, we lift most constructions to *weighted strings*, assigning each leaf a weight equal to its original expansion length. This is how we prove Theorem 1.4.

Prefix Aggregation, rank, and select. Prefix aggregation queries can be handled similarly to Random Access: during the same root-to-leaf traversal, we maintain the monoid aggregate of the already traversed prefix, yielding $\mathcal{O}(\log_\tau \frac{n}{g\tau})$ time once child_A is supported in constant time. The $\mathsf{rank}_{T,a}$ query is the special case where each character is mapped to 0 or 1 and the aggregate is the sum. As in previous works [5, 43], for $\mathsf{select}_{T,a}$, we first transform T so that each character encodes a block of non-a characters followed by an a, reducing select to a prefix aggregation query on the transformed string. Obtaining the bound in Theorem 1.5 (with $g\tau \log n$ in the denominator) requires a succinct structure for the leaf variables; we use Elias–Fano encoding [19, 20].

Lower Bound. We prove Theorem 1.2 in the *nondeterministic cell-probe model*, formulated in terms of *certificates* by Wang and Yin [52], as well as in the *randomized cell-probe model with two-sided bounded error*. We use a reduction from Blocked Lopsided Set Disjointness (BLSD) [41], extending the construction of Verbin and Yu [51] behind the lower bound discussed in Footnote 4 by introducing a tunable splitting parameter Q. The key idea is that, instead of answering a single BLSD query using one Random Access query in a highly compressible string ($g \approx \log^2 n$), we answer it using Q Random Access queries in a moderately compressible string (roughly $g \approx Q \log^2 \frac{n}{Q}$). This simple idea not only recovers the other lower bound of [51] (for $n^{\Omega(1)} \le g \le n^{1-\Omega(1)}$) but also readily generalizes it to almost all parameter regimes. The original proof of Verbin and Yu [51] also ultimately relies on the hardness of BLSD, but it passes through a several-step chain of reductions from [41, 51]. In the randomized setting, errors could accumulate across all Q Random Access queries, so our reduction uses an extra subroutine that repeats some queries using multiple copies of the data structure in order to derive a bounded-error communication protocol for BLSD that violates the lower bound of [41]. No such subroutine appears in [41], which is why their randomized cell-probe lower bounds only work in the zero-error models. In the nondeterministic setting, this issue disappears; the remaining annoyance, shared with the randomized setting, is the amount of calculations needed to map between all the parameters in the reduction.

1.2 Related Work

Random Access has been studied from a multitude of perspectives for dozens of compressibility measures $s(T)$. Early milestone results provide constant-time random access to entropy-compressed strings, achieving $nH_0(T) + o(n)$ bits for binary strings [44] and $nH_k(T) + o(n \log \sigma)$ bits for strings over larger alphabets [48] (for $k < o(\log_\sigma n)$); see also [26] for a survey. Unfortunately, low-order entropy does not capture large-scale repetitions, and the lower-order terms in these data structures are relatively large, which makes them unsuitable for highly compressible strings. That regime brings a whole zoo of compressibility (repetitiveness) measures; see [38] for a dedicated survey, which discusses which measures support efficient random access. In space $\mathcal{O}(s(T) \operatorname{polylog} n)$, the best one can typically hope for (in highly compressible strings) is $\mathcal{O}(\frac{\log n}{\log \log n})$ Random Access time: see [13] for a

work that generalizes the lower bound of [51] to many measures that are not bounded by $\mathcal{O}(g(T)\operatorname{polylog} n)$. An exception is LZ78 [54], for which $\mathcal{O}(\log\log n)$-time access is possible [16]. Unfortunately, it is not known how to support Random Access queries efficiently (e.g., in $n^{o(1)}$ time) in space proportional to the size $z(T)$ of the LZ77 parsing or $r(T)$ of the run-length encoded Burrows–Wheeler transform. The latter is particularly surprising given $\mathcal{O}(r(T))$-space data structures supporting pattern-matching queries [22], which often makes Random Access the sole bottleneck; see [2]. There exists, however, a trade-off between Random Access time and an additive term on top of a succinct encoding of $\mathsf{rle}(\mathsf{BWT}(T))$ [50]. The lack of efficient access in $\mathcal{O}(z(T))$ space motivated the introduction of more restricted variants like LZ-End [34], which support Random Access in $\mathcal{O}(\operatorname{poly}\log n)$ time [31], height-bounded LZ [1], bounded-access-time LZ [36], and LZBE [49]. Further works for Random Access in grammar-compressed strings include schemes providing faster access to characters that are close to static bookmarks [12] or dynamic fingers [6, 23], or located in incompressible regions [11]. Very recently, Random Access has also been studied for two-dimensional grammar-compressed strings [14].

1.3 Open Problems

The most important open problem about grammar-compressed Random Access remains whether $o(\log n)$-time queries can be achieved in $\mathcal{O}(g)$ space for any $g \le n^{1-\Omega(1)}$. We hope that our direct reduction from BLSD will inspire renewed efforts to tackle this question from the lower-bound side. A more structured variant that seems as difficult as the general case is when each production is of length two and each expansion length is a power of two; in that case, no balancing is needed and child_A queries remain trivial for every τ. With appropriate parameters, our lower bound of Theorem 1.2 only produces grammars of this form, albeit ones supporting $\mathcal{O}(\frac{\log n}{\log\log n})$-time queries.

Another task stemming from the limitations of Theorem 1.2 is to understand whether SLG Random Access becomes easier for very small g (e.g., $g \le \mathcal{O}(\log n)$) in the word RAM model.

Since every SLG can be encoded in $\mathcal{O}(g\log g)$ bits, for very small grammars ($g \le n^{o(1)}$) it is also open to achieve efficient (e.g., polylogarithmic-time) Random Access queries in $\mathcal{O}(g\log g)$ bits.

Compressibility measures closely related to $g(T)$ pose many further interesting and long-standing problems. In particular, it is not known how to answer Random Access queries in $n^{o(1)}$ time and in $\mathcal{O}(z(T))$ or $\mathcal{O}(r(T))$ space (where $r(T)$ is the number of runs in the Burrows–Wheeler transform).

Very recently [30], $\Omega(\frac{\log n}{\log\log n})$ lower bounds for moderately compressible strings have been proved for many queries beyond Random Access, including suffix array and inverse suffix array queries. Can one generalize these lower bounds to almost all parameter regimes, similarly to Theorem 1.2? Further queries, such as Longest Common Extension and Internal Pattern Matching, are at least as hard as Random Access, so the lower bounds translate automatically. However, for those queries it remains open if the existing upper bounds [28, 29, 17] (formulated in terms of $\delta(T)$ or $z(T)$) can be improved for weakly compressible strings and generalized to smooth interpolations with the uncompressed setting, in the spirit of Corollary 1.3.

Finally, we leave for future work whether our techniques can help in settings where the characters at some positions are supposed to be more easily accessible than others, e.g., due to bookmarks [12] or fingers [6, 23]. Currently, we only achieve this functionality by varying character weights, but the weight needs to be the same across occurrences of the character.

2 Preliminaries

Let Σ be some finite set. Let $T = t_0 \cdots t_{n-1} \in \Sigma^*$. We call T a *string* (or *sequence*) of length $|T| \coloneqq n$ over the *alphabet* Σ. We use character notation $T[i] \coloneqq t_i$ for $i \in [0..|T|)$ and substring notation $T[i..j) \coloneqq t_i \cdots t_{j-1}$ for $0 \le i \le j \le |T|$.

Let a^k for character $a \in \Sigma$ and integer $k \in \mathbb{Z}_{>0}$ denote the k-fold repetition of a; we also refer to it as the k-th *power* of a. Given any sequence $T \coloneqq a_0^{k_0} a_1^{k_1} \cdots a_{m-1}^{k_{m-1}} \in \Sigma^*$, where $a_i \neq a_{i+1}$ for $i \in [0..m-1)$ and $k_i \in \mathbb{Z}_{>0}$ for $i \in [0..m)$, we define its run-length encoding $\mathsf{rle}(T) = (a_0, k_0)(a_1, k_1) \cdots (a_{m-1}, k_{m-1})$. Note that $|\mathsf{rle}(T)| \coloneqq m$.

Straight-Line Grammars. We call $\mathcal{G} \coloneqq (\mathcal{V}_\mathcal{G}, \Sigma_\mathcal{G}, \mathsf{rhs}_\mathcal{G} : \mathcal{V}_\mathcal{G} \to (\mathcal{V}_\mathcal{G} \cup \Sigma_\mathcal{G})^*, S, \|\cdot\|_\mathcal{G} : \Sigma_\mathcal{G} \to \mathbb{Z}_{\ge 1})$ a *weighted straight-line grammar* (SLG) with variables $\mathcal{V}_\mathcal{G}$, terminals $\Sigma_\mathcal{G}$, right-hand sides $\mathsf{rhs}_\mathcal{G}$, start symbol $S \in \mathcal{V}_\mathcal{G} \cup \Sigma_\mathcal{G}$, and weight function $\|\cdot\|_\mathcal{G}$ if there is an order $\prec_\mathcal{G}$ on $\mathcal{V}_\mathcal{G}$ such that $B \prec_\mathcal{G} A$ holds whenever B appears in $\mathsf{rhs}_\mathcal{G}(A)$. We call $\mathcal{A}_\mathcal{G} \coloneqq \mathcal{V}_\mathcal{G} \cup \Sigma_\mathcal{G}$ the set of symbols. When clear from context we drop the subscript $\cdot_\mathcal{G}$. The *size* $|\mathcal{G}|$ of an SLG $\mathcal{G}$ is the sum of the lengths of all right-hand sides.

Straight-line grammars whose right-hand sides are stored in run-length encoded form are called run-length straight-line grammars (RLSLGs). Their semantics are unchanged; only the size is measured as the sum of the lengths of the run-length-encoded right-hand sides. Whenever we construct an (RL)SLG from another (RL)SLG, we assume that if the input is an SLG (i.e., without runs), then the output is also an SLG. Usually, we bound the size of every individual rule. Note that this size is measured differently for SLGs and RLSLGs.

The *expansion* of a terminal $a \in \Sigma_\mathcal{G}$ is $\mathsf{expand}_\mathcal{G}(a) \coloneqq a$. We extend expansions recursively to variables and sequences of symbols:

$$\mathsf{expand}_\mathcal{G}(A) \coloneqq \begin{cases} \mathsf{expand}_\mathcal{G}(\mathsf{rhs}(A)) & \text{if } A \in \mathcal{V}_\mathcal{G}, \\ \bigodot_{i \in [0..|A|)} \mathsf{expand}_\mathcal{G}(A[i]) & \text{if } A \in \mathcal{A}_\mathcal{G}^*. \end{cases}$$

We say that a symbol or a sequence of symbols A *expands to* the string $\mathsf{expand}_\mathcal{G}(A)$. The grammar $\mathcal{G}$ *produces* the string $\mathsf{expand}_\mathcal{G}(S)$. It *defines* the strings $\mathsf{expand}_\mathcal{G}(A)$ for all $A \in \mathcal{A}_\mathcal{G}$.

Let $L_\mathcal{G} \coloneqq \{\mathsf{expand}_\mathcal{G}(A) : A \in \mathcal{A}_\mathcal{G}\}$ denote the set of strings defined by $\mathcal{G}$. If $L_\mathcal{G} \subseteq L_\mathcal{H}$, then we say that $\mathcal{H}$ defines all strings defined by $\mathcal{G}$. We call the function $f : \mathcal{A}_\mathcal{G} \to \mathcal{A}_\mathcal{H}$ a grammar homomorphism from $\mathcal{G}$ to $\mathcal{H}$ if $\mathsf{expand}_\mathcal{G}(A) = \mathsf{expand}_\mathcal{H}(f(A))$ holds for every $A \in \mathcal{A}_\mathcal{G}$. The existence of such a homomorphism implies $L_\mathcal{G} \subseteq L_\mathcal{H}$.

If the weight function $\|\cdot\|_\mathcal{G} : \Sigma_\mathcal{G} \to \mathbb{Z}_{>0}$ is not stated explicitly, we assume it is the unit function $\|\cdot\|_\mathcal{G} \coloneqq 1$. We call such a grammar unweighted. We extend the weight function recursively to variables and sequences of symbols:

$$\|A\|_\mathcal{G} \coloneqq \begin{cases} \|\mathsf{rhs}(A)\|_\mathcal{G} & \text{if } A \in \mathcal{V}_\mathcal{G}, \\ \sum_{i \in [0..|A|)} \|A[i]\|_\mathcal{G} & \text{if } A \in \mathcal{A}_\mathcal{G}^*. \end{cases}$$

For every sequence of symbols A, the property $\|A\| = \|\mathsf{expand}_\mathcal{G}(A)\| \ge |\mathsf{expand}_\mathcal{G}(A)|$ holds.

The *height* $h_\mathcal{G}(A)$ of every symbol $A \in \mathcal{A}_\mathcal{G}$ is also defined recursively:

$$h_\mathcal{G}(A) \coloneqq \begin{cases} 0 & \text{if } A \in \Sigma_\mathcal{G}, \\ 0 & \text{if } A \in \mathcal{V}_\mathcal{G} \text{ and } \mathsf{rhs}_\mathcal{G}(A) = \varepsilon, \\ 1 + \max_i h_\mathcal{G}(A_i) & \text{if } A \in \mathcal{V}_\mathcal{G} \text{ and } \mathsf{rhs}_\mathcal{G}(A) = A_0 \cdots A_{k-1} \text{ for } k \ge 1. \end{cases}$$

Parse Trees. We call the following directed graph the *parse tree* of a weighted SLG. Here, each node is a pair (A, a) consisting of a symbol A and a weighted offset a into the expansion of the start symbol.

▶ **Definition 2.1.** *Let $\mathcal{G} = (\mathcal{V}, \Sigma, \mathsf{rhs}, S, \|\cdot\|)$ be an (RL)SLG. We call the directed graph $P_{\mathcal{G}}$ the parse tree of $\mathcal{G}$ if it is the smallest graph such that:*

1. *$P_{\mathcal{G}}$ contains a node $(S, 0)$;*
2. *for every node $u := (A, a)$ with $A \in \mathcal{V}$ and $\mathsf{rhs}(A) = B_0 \cdots B_{k-1}$, and for every $i \in [0..k)$, the parse tree $P_{\mathcal{G}}$ contains a node $v_i := \left(B_i, a + \sum_{j=0}^{i-1} \|B_j\|\right)$ and an edge (u, v_i).*

Note that, for every node (A, a) of the parse tree $P_{\mathcal{G}}$, the height $h(A)$ is equal to the number of edges of the longest path from (A, a) to a leaf of $P_{\mathcal{G}}$ (the height of the subtree rooted at (A, a)).

3 The Upper Bounds: Overview

This section overviews [18, Sections 5 to 8], which contain the upper-bound proofs behind Theorems 1.1, 1.4, and 1.5. As discussed in Section 1.1, the core theme is to transform an input grammar into a form that admits *constant-time local navigation*, while keeping the grammar *shallow* and the overall size under control. All transformations are phrased in terms of grammar homomorphisms, so they preserve expansions and let us switch to more structured grammars producing the same strings.

Random Access in Weighted Strings and Parse-Tree Viewpoint. Most of our results are stated for strings over weighted alphabets $(\Sigma, \|\cdot\|)$. In that setting, a random access query is given $i \in [0..\|T\|)$ and returns $T[j]$ for the unique position $j \in [0..|T|)$ such that $\|T[0..j)\| \le i < \|T[0..(j+1))\|$. If needed, we can also output the offset $\|T[0..j)\|$ and the position j.

Random Access in a string T produced by an (RL)SLG $\mathcal{G}$ corresponds to descending the parse tree $P_{\mathcal{G}}$: we maintain a node (A, a) with $i \in [a..a + \|A\|)$ and repeatedly move to the child (B, b) whose interval $[b..b+\|B\|)$ contains i, until a terminal is reached [18, Algorithm 1]. This transition step is implemented by a child_A query. For $\mathsf{rhs}_{\mathcal{G}}(A) = B_0 \cdots B_{k-1}$, child_A reduces to a rank query in $X_A = \{\|B_0 \cdots B_{j-1}\| : j \in [1..k]\}$ on input $i - a$; see [18, Lemma 6.1]. Thus, the overall query time is governed by (i) the per-node cost of child_A and (ii) the depth of $P_{\mathcal{G}}$; the goal is to optimize both.

Contracting Grammars. The first step is to obtain a grammar in which every variable shrinks in a controlled way along parse-tree edges. Ganardi [23] introduced *contracting* SLGs, requiring that every child B of a variable A satisfies $\|B\| \le \|A\|/2$. He proved that every SLG $\mathcal{G}$ can be homomorphically embedded into a contracting SLG of size $\mathcal{O}(|\mathcal{G}|)$ and production lengths bounded by some constant d. Although the final result in [23] is established for the unweighted case, the key ingredients of its proof already use weighted SLGs. In [18, Theorem 5.5], we extend the statement and the remaining arguments to fully incorporate the weighted perspective. This extension makes it easy to lift the result to RLSLGs in [18, Corollary 5.8].[7] We treat these results as black boxes, relying only on the constant bound d on right-hand side size (number of symbols or runs of symbols).

[7] A parallel independent work [11] proves the same result, albeit for unweighted RLSLGs only.

From Contracting to Nice. The contracting property already implies logarithmic height $h(A) \le \mathcal{O}(\log|\mathsf{expand}_{\mathcal{G}}(A)|)$ for every symbol $A \in \mathcal{A}_{\mathcal{G}}$, but to obtain the desired trade-off we need faster shrinkage. For $\tau > 1$, a child B of A is *τ-heavy* if $\|B\| > \frac{1}{\tau}\|A\|$, and τ-light otherwise; see [18, Definition 5.9]. We also impose a local regularity condition that later yields $\mathcal{O}(1)$-time child_A implementation.

In [18, Definition 5.10], we call a variable A *τ-nice* if:

(a) every variable child in $\mathsf{rhs}(A)$ is τ-light;
(b) $\mathsf{rhs}(A)$ has at most $2d\tau$ runs (or symbols, for SLGs);
(c) every substring X of $\mathsf{rhs}(A)$ of weight at most $\frac{1}{\tau}\|A\|$ has at most $2d\lceil\log\tau\rceil$ runs.

The key construction in [18, Lemma 5.11] starts from a *contracting* grammar and replaces $\mathsf{rhs}(A)$ by a sequence $\mathsf{rhs}_\tau(A) \in \mathcal{A}_{\mathcal{G}}^*$, obtained by exhaustively replacing all τ-heavy children with their right-hand sides. The construction runs in time $\mathcal{O}(\tau)$ and ensures (b) and (c); its proof is an induction on $\lceil\log\tau\rceil$ via the recursion $\tau \mapsto \tau/2$. The only difference between $\mathsf{rhs}_{\tau/2}(A)$ and $\mathsf{rhs}_\tau(A)$ is that τ-heavy children are replaced by their right-hand sides; thanks to the contracting property, these replacements can happen in parallel, and there are $\mathcal{O}(\tau)$ of them in total and $\mathcal{O}(1)$ in a substring of weight at most $\frac{1}{\tau}\|A\|$. Each replacement produces at most d runs, so the bounds in (b) and (c) grow by $\mathcal{O}(d\tau)$ and $\mathcal{O}(d)$ compared to $\tau/2$.

We apply this process to all variables, with τ_r for the starting symbol and τ_v for the remaining ones, to obtain what we call a (τ_r, τ_v)-nice grammar of size $\mathcal{O}(\tau_r + |\mathcal{G}|\tau_v)$. In the resulting parse tree, the depth of every node (A, a) is at most $2 + \max\left(0,\ \log_{\tau_v}\frac{\|S_{\mathcal{G}}\|}{\tau_r\|A\|}\right)$; see [18, Lemma 5.14].

Leafy Grammars: Packing b Characters per Leaf. To exploit bit-parallelism when σ is small, we introduce *b-leafy* grammars; see [18, Definition 5.15]. Here variables split into: *leaf variables* $\mathcal{V}_{\mathcal{H}}^{\mathsf{leaf}}$ expanding to explicit terminal strings of length in $[b..2b)$, and *top variables* $\mathcal{V}_{\mathcal{H}}^{\mathsf{top}}$ whose right-hand sides contain only (top and leaf) variables. The top part $\mathcal{H}^{\mathsf{top}}$ treats leaf variables as terminals and captures the global structure, while leaf variables store the explicit blocks (packed into $\mathcal{O}(b\log\sigma)$ bits, so $\mathcal{O}(1)$ words for our choices of b). The construction in [18, Lemma 5.17] guarantees $|\mathcal{H}^{\mathsf{top}}|, |\mathcal{V}_{\mathcal{H}}^{\mathsf{leaf}}| \in \mathcal{O}(|\mathcal{G}|)$ and runs in time $\mathcal{O}(|\mathcal{G}| \cdot (1 + b\log\sigma/w))$: we maintain only $\mathcal{O}(1)$ helper variables per original variable and generate each explicit block directly as a packed string. The produced top rules have a deliberately restricted form: the homomorphic image of each variable of $\mathcal{G}$ with expansion length at least b is a top variable whose right-hand side consists of at most two leaf variables plus possibly a middle top variable in between, and every helper top rule has constant run-length size. This prevents the parse tree from containing long chains of tiny explicit pieces and is the structural reason why the traversal routines in [18, Section 7] can output $\Theta(b)$ characters per step.

We then combine leafiness with niceness: [18, Corollary 5.18] builds, from an unweighted RLSLG, a b-leafy grammar whose weighted top part $\mathcal{H}^{\mathsf{top}}$ is (τ_r, τ_v)-nice; this is where we need the preceding niceness construction to work for weighted strings. Leafiness yields a further height reduction, giving a height bound of $3 + \max\left(0,\ \log_{\tau_v}\frac{\|S_{\mathcal{G}}\|}{\tau_r b}\right)$.

Constant-Time child Queries from Niceness. After the above transformations, the remaining algorithmic bottleneck is answering child_A at a τ-nice variable A. [18, Lemma 6.1] shows that we can preprocess A in $\mathcal{O}(\tau)$ time into $\mathcal{O}(\tau\log\|A\|)$ bits so that child_A is answered in $\mathcal{O}(1)$ time. At a high level, we store the run-length encoding $C_A = \mathsf{rle}(\mathsf{rhs}(A))$ and the cumulative run boundaries P_A (prefix weights of runs). Then $\mathsf{child}_A((A, a), i)$ reduces to computing the rank of $i - a$ in the set of run boundaries plus arithmetic operations (integer division) to locate i within the run. Condition (c) in τ-niceness implies a *local sparsity*

property: every interval of weighted length $\frac{1}{\tau}\|A\|$ intersects only $\mathcal{O}(\log \tau)$ run boundaries. This enables a standard bucketing strategy: we partition the universe $[0..\|A\|)$ into τ buckets, and inside each bucket store the $\mathcal{O}(\log \tau)$ boundaries in a constant-time rank structure (fusion tree [21, 42]). The total size of the τ buckets is still $\mathcal{O}(\tau)$, so these rank structures take $\mathcal{O}(\tau \log\|A\|)$ bits in total, and they can be constructed in $\mathcal{O}(\tau)$ total time.

Random Access: Putting It Together. For weighted strings, [18, Lemma 6.3] constructs a $(|\mathcal{G}|\tau, \tau)$-nice grammar and the per-variable child_A structures, and then answers queries by the parse-tree traversal above. The time becomes the nice height bound, namely $\mathcal{O}\Big(\max\Big(1,\ \log_\tau \frac{\|T\|}{|\mathcal{G}|\tau\|c\|}\Big)\Big)$ for the returned character c. For unweighted $T \in [0..\sigma)^n$, we additionally apply the leafy transformation with $b = \Theta(\log_\sigma n)$ so each leaf block fits in $\mathcal{O}(1)$ words (when $w = \Theta(\log n)$), yielding [18, Lemma 6.4] and the trade-off in Theorem 1.1. We also prove a weighted analog of [18, Lemma 6.4] in [18, Theorem 6.5].

Traversal and Substring Extraction. [18, Section 7] generalizes random access to a pointer interface that supports constant-time forward/backward traversal. For weighted strings, we show how to traverse characters one by one, building upon [37]. For unweighted strings over small alphabets, we crucially exploit b-leafy grammars. The key is to apply character pointers to the weighted top string T^{top} while maintaining an offset inside the current explicit leaf block; a character pointer to T is represented as "(a pointer to T^{top}) + (an offset in the current block)". Because each leaf block has length $\Theta(b)$ and is stored packed, each traversal step can return $\Theta(b)$ characters in constant time. As a result, [18, Corollary 7.10] supports extracting m consecutive characters around a position in time $\mathcal{O}(1 + m/b)$, with construction time matching the random-access trade-off; [18, Corollary 7.11] turns this directly into substring extraction. The final bounds choose $b = \Theta(\min(w, \tau \log n)/\log \sigma)$, so the block operations fit in $\mathcal{O}(1)$ words and the additive extraction term becomes $m \log \sigma / \min(w, \tau \log n)$.

Rank/Select via Prefix Sums. Finally, [18, Section 8] lifts the entire framework from extracting characters to computing arbitrary *monoid-valued* aggregates. Given a mapping $\Phi : \Sigma \to (\mathbf{M}, \oplus, \mathbf{0})$, prefix-sum queries ask for $\Phi(T[0..i))$. [18, Lemma 8.4 and 8.5] mirror Random Access: we store for each variable the monoid value of its expansion and combine it along the traversed path. Rank follows by instantiating Φ appropriately (mapping a queried character to 1 and the rest to 0). For select, we first transform the string so that each character encodes the number of non-occurrences between consecutive occurrences (i.e., a block of non-a characters followed by an a), reducing $\mathsf{select}_{T,a}$ to a prefix-sum query on the transformed string; see [18, Lemma 8.9]. To obtain the tightest bounds, we additionally build succinct rank/select structures for the short leaf blocks: we use constant-time rank within packed bitmasks and constant-time select in the Elias–Fano encoding ([18, Facts 8.1 and 8.2]), so the extra work and storage at leaves do not introduce additional overhead.

4 The Lower Bound

In this section, we prove cell-probe lower bounds for SLG Random Access against both nondeterministic and bounded-error randomized query algorithms. The lower bounds share the high-level structure and many steps, so we present them in parallel. We first explain the formal settings.

Any data structure problem can be modeled as a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, where $x \in \mathcal{X}$ is a query, $y \in \mathcal{Y}$ is an input, and $f(x, y)$ is the correct answer. A deterministic cell-probe data structure with M cells of w bits consists of an encoding

$$D : \mathcal{Y} \to (\{0,1\}^w)^M$$

and a query algorithm that probes at most t cells. Formally, the query algorithm is given by probe functions $A_0, \ldots, A_{t-1}$ and an output function A_{out}, where

$$A_j : \mathcal{X} \times (\{0,1\}^w)^j \to [0..M) \quad \text{for } j \in [0..t) \qquad \text{and} \qquad A_{\mathsf{out}} : \mathcal{X} \times (\{0,1\}^w)^t \to \mathcal{Z}.$$

For a fixed query $x \in \mathcal{X}$ and input $y \in \mathcal{Y}$, the probed addresses are

$$i_j \coloneqq A_j(x, D(y)[i_0], \ldots, D(y)[i_{j-1}]) \qquad \text{for } j \in [0..t),$$

and the returned answer is

$$D_{\mathsf{out}}(x, y) \coloneqq A_{\mathsf{out}}(x, D(y)[i_0], \ldots, D(y)[i_{t-1}]).$$

Correctness means $D_{\mathsf{out}}(x, y) = f(x, y)$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

In the nondeterministic cell-probe model, the query algorithm additionally receives a hint h from an unbounded domain. Equivalently, the functions $A_0, \ldots, A_{t-1}, A_{\mathsf{out}}$, and hence also the derived outcome function D_{out}, receive h as an extra parameter. Moreover, the output function is allowed to return an extra value $\perp$. Correctness requires that for every query $x \in \mathcal{X}$ and input $y \in \mathcal{Y}$,

- there exists a hint h such that $D_{\mathsf{out}}(x, y, h) = f(x, y)$, and
- for every hint h, either $D_{\mathsf{out}}(x, y, h) = f(x, y)$ or $D_{\mathsf{out}}(x, y, h) = \perp$.

Since hints are unbounded, we may assume that a hint specifies the probed cells together with their expected contents, and the algorithm outputs $\perp$ as soon as one probe disagrees with the hint. Thus, a successful hint is simply a set of t cells whose contents determine the answer uniquely. This leads to the notion of *certificates* due to Wang and Yin [52].

▶ **Definition 4.1** (Certificates [52]). *A data structure problem $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ has (M, w, t)-certificates if there exists a function $D : \mathcal{Y} \to (\{0,1\}^w)^M$ such that, for every query $x \in \mathcal{X}$ and every input $y \in \mathcal{Y}$, there exists a set $C \subseteq [0..M)$ with $|C| = t$ such that, for every $y' \in \mathcal{Y}$, if $D(y)[i] = D(y')[i]$ for all $i \in C$, then $f(x, y') = f(x, y)$.*

In the randomized setting, the data structure and the query algorithm share a random seed r drawn uniformly from a finite domain $\mathcal{R}$. Formally, both D and the functions $A_0, \ldots, A_{t-1}, A_{\mathsf{out}}$ receive r as an extra argument. Hence the derived outcome function D_{out} also depends on r. The correctness requirement is that, for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we have $\Pr_{r \sim U(\mathcal{R})}[D_{\mathsf{out}}(x, y, r) = f(x, y)] \geq \frac{2}{3}$.

▶ **Definition 4.2.** *A data structure problem $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ admits an (M, w, t)-bounded-error structure if there exist a finite seed domain $\mathcal{R}$, an encoding $D : \mathcal{Y} \times \mathcal{R} \to (\{0,1\}^w)^M$, probe functions $A_j : \mathcal{X} \times \mathcal{R} \times (\{0,1\}^w)^j \to [0..M)$ for $j \in [0..t)$, and an output function $A_{\mathsf{out}} : \mathcal{X} \times \mathcal{R} \times (\{0,1\}^w)^t \to \mathcal{Z}$ such that $\Pr_{r \sim U(\mathcal{R})}[D_{\mathsf{out}}(x, y, r) = f(x, y)] \geq \frac{2}{3}$ for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.*

We will reduce Blocked Lopsided Set Disjointness to SLG Random Access. Recall that, in Lopsided Set Disjointness (LSD), the goal is to decide whether two sets X and Y over a common universe are disjoint. In the data-structure version, one of the sets, say Y, can

be preprocessed, while the other set X is revealed at query time. In Blocked LSD (BLSD), the universe is partitioned into N blocks of size B, and the set X must contain exactly one element from each block. Equivalently, an instance of BLSD can be represented by functions S and T, so that, for each $i \in [0..N)$, the value $S(i) \in [0..B)$ specifies the unique element of X in the i-th block, whereas $T(i) \subseteq [0..B)$ specifies the intersection of Y with that block.

Blocked Lopsided Set Disjointness (BLSD) [41]
Parameters: $N, B \in \mathbb{Z}_{>0}$
Input: A function $T : [0..N) \to 2^{[0..B)}$.
Queries: Given $S : [0..N) \to [0..B)$, decide if there exists $i \in [0..N)$ such that $S(i) \in T(i)$.

Wang and Yin [52, Theorem 26] proved a certificate lower bound for 2-Blocked-LSD, which imposes an additional distinctness condition within every group of B consecutive blocks, namely $S(i) \neq S(i')$ whenever $\lfloor i/B \rfloor = \lfloor i'/B \rfloor$. Since this is a more restrictive version of BLSD, it yields the following lower bound for BLSD as well:

▶ **Corollary 4.3.** *Consider $N, B, M, w, t \in \mathbb{Z}_{>0}$ such that $M > t$, and a real constant $\lambda > 0$. If BLSD with parameters N, B has (M, w, t)-certificates, then $t \geq \Omega\Big(\frac{NB^{1-\lambda}}{w}\Big)$ or $t \geq \Omega\Big(\frac{N \log B}{\log \frac{M}{t}}\Big)$.*

For a lower bound against randomized algorithms, we interpret BLSD as a communication problem with two players: Alice, who receives S, and Bob, who receives T. Pătraşcu [41] proved asymmetric communication complexity lower bounds for this problem:

▶ **Theorem 4.4** ([41, Theorem 1.4])**.** *Consider $N, B \in \mathbb{Z}_{>0}$ and a real constant $\lambda > 0$. In any bounded-error communication protocol for BLSD with parameters N, B, Alice sends at least $\Omega(N \log B)$ bits or Bob sends at least $\Omega(NB^{1-\lambda})$ bits.*

Next, we reduce BLSD to SLG Random Access. We follow the encoding of Verbin and Yu [51] with a crucial extension that allows answering a single BLSD query using several Random Access queries. The following lemma captures the original reduction of [51].

▶ **Lemma 4.5** ([51, Lemma 2])**.** *Let $N, B \in \mathbb{Z}_{>0}$. For every function $T : [0..N) \to 2^{[0..B)}$, there exists an SLG of size at most $4NB$ producing a binary string V of length B^N such that, for every function $S : [0..N) \to [0..B)$, we have*

$$V\left[\sum_{i \in [0..N)} S(i) \cdot B^i\right] = 1 \quad \text{if and only if} \quad \bigvee_{i \in [0..N)} S(i) \in T(i).$$

Our generalization splits the input BLSD instances into smaller subinstances before applying Lemma 4.5. The parameter Q controls the number of Random Access queries needed to answer one BLSD query. While keeping the compressed size of the produced string V at $\Theta(NB)$, this allows reducing the length of V from B^N to $Q \cdot B^{N/Q}$. As a result, we obtain a smooth trade-off between storing T explicitly (NB bits) and storing the answers to all queries explicitly (B^N bits).

▶ **Lemma 4.6.** *Let $P, Q, B \in \mathbb{Z}_{>0}$. For every function $T : [0..PQ) \to 2^{[0..B)}$, there exists an SLG of size at most $5PQB$ that produces a binary string V of length $Q \cdot B^P$ such that, for every function $S : [0..PQ) \to [0..B)$, we have*

$$\bigvee_{q \in [0..Q)} V\left[qB^P + \sum_{p \in [0..P)} S(qP + p) \cdot B^p\right] = 1 \quad \text{if and only if} \quad \bigvee_{i \in [0..PQ)} S(i) \in T(i).$$

Proof. Given a BLSD instance of PQ blocks of size B each, we split the instance into Q parts of P consecutive blocks each. Each such part, indexed with $q \in [0..Q)$, corresponds to blocks $i \in [qP..(q+1)P)$ and, after shifting this interval by qP, can be interpreted as a BLSD instance of size (P, B). Lemma 4.5 yields a binary string V_q of length B^P produced by a grammar $\mathcal{G}_q$ of size at most $4PB$ such that

$$V_q\left[\sum_{p\in[0..P)} S(qP+p)\cdot B^p\right] = \mathtt{1} \quad \text{if and only if} \quad \bigvee_{p\in[0..P)} S(qP+p) \in T(qP+p).$$

The final string V is the concatenation $V = \bigodot_{q\in[0..Q)} V_q$; since $V_q = V[qB^P..(q+1)B^P)$, this immediately yields the claimed equivalence. Moreover, $|V| = Q \cdot B^P$ and V is produced by a grammar $\mathcal{G}$ obtained by taking a disjoint union of grammars $\mathcal{G}_q$ and introducing a new start symbol (with a production of length Q) capturing the top-level concatenation. The total size of $\mathcal{G}$ is at most $Q \cdot 4PB + Q \le 5PQB$. ◀

▶ **Example 4.7.** If $P = Q = B = 2$, then the universe consists of four blocks of size two, which we split into two subinstances on blocks $0, 1$ and $2, 3$. The reduction builds a string $V = V_0V_1$ of length $2 \cdot 2^2 = 8$, where each V_q has length 4. Consider the following instance:

$$T(0) = \{1\}, \qquad T(1) = \emptyset, \qquad T(2) = \{0\}, \qquad T(3) = \{1\}.$$

Here, the first subinstance has hits exactly for $(S(0), S(1)) \in \{(1,0),(1,1)\}$, so $V_0 = \mathtt{0101}$. Likewise, the second subinstance has hits exactly for $(S(2), S(3)) \in \{(0,0),(0,1),(1,1)\}$, so $V_1 = \mathtt{1011}$. Thus, $V = \mathtt{01011011}$. For a query $S : [0..4) \to [0..2)$, the two accessed positions are

$$v_0 = S(0) + 2S(1) \quad \text{and} \quad v_1 = 4 + S(2) + 2S(3).$$

For instance, if $S = (1, 0, 1, 0)$, then $(v_0, v_1) = (1, 5)$, so the BLSD answer is $V[1] \vee V[5] = \mathtt{1} \vee \mathtt{0} = \mathtt{1}$. On the other hand, if $S = (0, 1, 1, 0)$, then $(v_0, v_1) = (2, 5)$, so the BLSD answer is $V[2] \vee V[5] = \mathtt{0} \vee \mathtt{0} = \mathtt{0}$.

With this reduction, we can translate certificates for random access into certificates for BLSD.

▶ **Corollary 4.8.** *Consider $P, Q, B, M, w, t \in \mathbb{Z}_{>0}$ such that $M \ge tQ$. If SLG Random Access with parameters $n := Q \cdot B^P$, $g := 5PQB$, and $\sigma := 2$ has (M, w, t)-certificates, then BLSD with parameters $N := PQ$ and B has (M, w, tQ) certificates.*

Proof. Consider a BLSD input $T : [0..PQ) \to 2^{[0..B)}$ and the corresponding string V obtained using Lemma 4.6. By construction, this string is an SLG Random Access instance with parameters $n = Q \cdot B^P$, $g = 5PQB$, and $\sigma = 2$. We use the hypothetical SLG Random Access data structure for V as a data structure for BLSD queries in T. By Lemma 4.6, every BLSD query S to T can be answered using Q random access queries to V, asking for $V[v_q]$ for some positions $v_q \in [0..|V|)$ indexed with $q \in [0..Q)$. By our assumption, for each $q \in [0..Q)$, there is a certificate $C_q \subseteq [0..M)$ of size t that determines the value of $V[v_q]$. Taking the union $C := \bigcup_{q\in[0..Q)} C_q$, padded with arbitrary elements to reach size $t \cdot Q \le M$, we obtain a certificate that determines $V[v_q]$ for all $q \in [0..Q)$, and hence also their disjunction. By Lemma 4.6, this disjunction is exactly the BLSD answer, that is, whether $\bigvee_{i\in[0..N)} S(i) \in T(i)$ holds. Thus, BLSD with parameters PQ and B has (M, w, tQ)-certificates. ◀

In order to derive a counterpart of Corollary 4.8 for bounded-error randomized data structures, extra care is needed to avoid accumulating error probabilities from multiple random access queries used to answer a single BLSD query. In [41], similar one-to-many reductions miss analogous error analysis and thus almost all cell-probe lower bounds hold only against zero-error query algorithms. The central piece that allows supporting bounded-error structures is the following abstract result.

▶ **Proposition 4.9.** *Consider a bit sequence $x_0, \ldots, x_{q-1}$ and a randomized oracle that, given any subset $A \subseteq [0..q)$, returns bits $(y_a)_{a\in A}$ such that $\Pr[y_a = x_a] \geq \frac{2}{3}$ for every $a \in A$. The returned bits may be arbitrarily dependent within a single oracle call, but the randomness used by different oracle calls is independent. Then, the disjunction $\bigvee_{i\in[0..q)} x_i$ can be computed correctly with probability at least $\frac{2}{3}$ using $\mathcal{O}(1 + \log q)$ oracle calls, where the jth oracle call is made on a subset of size $\mathcal{O}(q/2^{\Omega(j)})$.*

Proof. Our high-level strategy is to maintain and gradually shrink a subset $A \subseteq [0..q)$ such that $\bigvee_{i\in[0..q)} x_i = \bigvee_{a\in A} x_a$. Each step of this process is implemented using the following subroutine:

▷ Claim 4.10. There exists a randomized procedure that, given a set $A \subseteq [0..q)$ and an integer $\rho \geq 2$, uses at most $r \coloneqq \lceil 108 \ln \rho \rceil$ oracle calls, all on the set A, and returns a subset $B \subseteq A$ satisfying

$$|B| \leq \lceil |A|/\rho \rceil \qquad \text{and} \qquad \Pr\left[\bigvee_{i\in A} x_i = \bigvee_{i\in B} x_i\right] \geq 1 - \tfrac{1}{6\rho}.$$

Proof. If $A = \emptyset$, we return $B = \emptyset$, which satisfies all requirements. Henceforth, suppose $A \neq \emptyset$.

We make r independent oracle calls on A. For every $i \in A$, let $z_i \in \{0, 1\}$ be the majority value among the r obtained bits y_i. Construct $C \coloneqq \{i \in A : z_i = 1\}$, and return an arbitrary subset $B \subseteq C$ of size $\min(|C|, \lceil |A|/\rho \rceil)$.

For every $i \in A$ and every $\ell \in [0..r)$, let $X_{i,\ell} \in \{0, 1\}$ indicate whether the ℓ-th returned bit for i is wrong. Then $\mathbb{E}[X_{i,\ell}] \leq \frac{1}{3}$, and the random variables $X_{i,0}, \ldots, X_{i,r-1}$ are independent. Since $z_i \neq x_i$ implies $\sum_{\ell=0}^{r-1} X_{i,\ell} \geq \frac{r}{2}$, Hoeffding's inequality yields

$$\Pr[z_i \neq x_i] \leq \Pr\left[\frac{1}{r}\sum_{\ell=0}^{r-1} X_{i,\ell} - \mathbb{E}\left[\frac{1}{r}\sum_{\ell=0}^{r-1} X_{i,\ell}\right] \geq \frac{1}{6}\right] \leq e^{-2r(1/6)^2} = e^{-r/18}.$$

Since $r = \lceil 108 \ln \rho \rceil$, we conclude that the following holds for every $i \in A$:

$$\Pr[z_i \neq x_i] \leq e^{-r/18} \leq e^{-6\ln\rho} = \rho^{-6} \leq \tfrac{1}{16\rho^2}.$$

If $\bigvee_{i\in A} x_i = 0$, then every subset $B \subseteq A$ satisfies $\bigvee_{i\in B} x_i = 0 = \bigvee_{i\in A} x_i$. It remains to consider the case $\bigvee_{i\in A} x_i = 1$. Define the set of false positives

$$A_{\mathsf{FP}} \coloneqq \{i \in A : x_i = 0 \text{ and } z_i = 1\}.$$

Then, $\mathbb{E}[|A_{\mathsf{FP}}|] \leq \frac{|A|}{16\rho^2}$, and by Markov's inequality,

$$\Pr\left[|A_{\mathsf{FP}}| \geq \tfrac{|A|}{\rho}\right] \leq \tfrac{|A|}{16\rho^2} \cdot \tfrac{\rho}{|A|} \leq \tfrac{1}{16\rho}.$$

Moreover, for a fixed $j \in A$ with $x_j = 1$, we have

$$\Pr[z_j = 0] \leq \tfrac{1}{16\rho^2} \leq \tfrac{1}{16\rho}.$$

Hence, with probability at least $1 - \frac{1}{16\rho} - \frac{1}{16\rho} > 1 - \frac{1}{6\rho}$, we have both $z_j = 1$ and $|A_{\mathsf{FP}}| < \frac{|A|}{\rho}$. Conditioned on this event, C contains at least one true positive and strictly fewer than $\frac{|A|}{\rho}$ false positives. Therefore, every subset of C of size $\min(|C|, \lceil |A|/\rho \rceil)$ still contains a true positive, and so the returned set B satisfies $\bigvee_{i\in A} x_i = \bigvee_{i\in B} x_i$. ◁

Let us continue with the proof of Proposition 4.9. We define $L \coloneqq \lceil \log(1 + \log q) \rceil$ and construct a descending sequence of sets $[0\,..q) =: A_0 \supseteq A_1 \supseteq \cdots \supseteq A_L$ so that, for each $b \in [0\,..L)$, the set A_{b+1} is obtained from A_b by applying Claim 4.10 with parameter $\rho_b \coloneqq 2^{2^b}$.

Since $\lceil \lceil m/\rho \rceil / \rho' \rceil = \lceil m/(\rho \cdot \rho') \rceil$ holds for all positive integers $m, \rho, \rho' \in \mathbb{Z}_{>0}$, an induction on $b \in [0\,..L]$ shows that

$$|A_b| \le \left\lceil \frac{q}{\prod_{i=0}^{b-1} \rho_i} \right\rceil = \left\lceil \frac{q}{2^{2^b-1}} \right\rceil$$

holds for every $b \in [0\,..L]$ and, in particular,

$$|A_L| \le \left\lceil \frac{q}{2^{2^L-1}} \right\rceil \le \left\lceil \frac{q}{2^{2^{\log(1+\log q)}-1}} \right\rceil = \left\lceil \frac{q}{2^{1+\log q-1}} \right\rceil = 1.$$

If $A_L = \emptyset$, we output 0. Otherwise, we make $\mathcal{O}(1)$ independent oracle calls on the singleton set A_L, and output the majority value among the obtained estimates for the unique element of A_L.

If all reduction steps succeed, then $\bigvee_{i\in[0\,..q)} x_i = \bigvee_{i\in A_L} x_i$. By Claim 4.10 and the union bound, the probability that all reduction steps succeed is at least

$$1 - \sum_{b=0}^{L-1} \tfrac{1}{6\rho_b} \ge 1 - \tfrac{1}{6} \sum_{b\ge 0} 2^{-2^b} > 1 - \tfrac{1}{6}.$$

The final majority vote on A_L is correct with probability at least $\frac{5}{6}$, after increasing the hidden constant in its $\mathcal{O}(1)$ repetitions if necessary. Therefore, the overall success probability is at least $1 - \frac{1}{6} - \frac{1}{6} = \frac{2}{3}$.

It remains to justify the complexity bound. In batch $b \in [0\,..L)$, the algorithm queries the set A_b, which we know to be of size $\mathcal{O}\left(\frac{q}{2^{2^b}}\right)$, and it makes $\mathcal{O}(\log \rho_b)$ oracle calls. The number of oracle calls used in the reduction steps is proportional to

$$\sum_{b=0}^{L-1} \log \rho_b = \sum_{b=0}^{L-1} 2^b \le \mathcal{O}(2^L) \le \mathcal{O}(\log q).$$

Together with the final $\mathcal{O}(1)$ singleton calls, this gives $\mathcal{O}(1 + \log q)$ oracle calls in total.

Finally, enumerate all oracle calls in chronological order. If the sth oracle call belongs to batch b, then the first $b+1$ batches contain only $\mathcal{O}(2^b)$ oracle calls in total, so $s \le \mathcal{O}(2^b)$, and therefore $2^b \ge \Omega(s)$. Hence, this call is made on a subset of size $\mathcal{O}\left(\frac{q}{2^{2^b}}\right) \le \frac{q}{2^{\Omega(s)}}$. If the sth oracle call is one of the final singleton calls, then $s \le \mathcal{O}(1 + \log q)$, and its queried set has size $1 \le \mathcal{O}(q/2^{\Omega(s)})$, after decreasing the constant hidden in the $\Omega(s)$ exponent if necessary. ◀

We are now ready to provide a counterpart of Corollary 4.8 for bounded-error data structures.

▶ **Corollary 4.11.** *Consider $P, Q, B, M, w, t \in \mathbb{Z}_{>0}$ such that $M \geq Q$. Suppose that SLG* Random Access *with parameters $n \coloneqq Q \cdot B^P$, $g \coloneqq 5PQB$, and $\sigma \coloneqq 2$ admits an (M, w, t)-bounded-error data structure. Then, BLSD with parameters $N \coloneqq PQ$ and B admits a bounded-error communication protocol in which Alice sends $\mathcal{O}(tQ \log \frac{2M}{Q})$ bits and Bob sends $\mathcal{O}(tQw)$ bits.*

Proof. Fix a BLSD input $T : [0..PQ) \to 2^{[0..B)}$ held by Bob. Let V be the binary string produced from T using Lemma 4.6. This string constitutes an SLG Random Access instance with parameters $n = Q \cdot B^P$, $g = 5PQB$, and $\sigma = 2$; by assumption, it thus admits an (M, w, t)-bounded-error data structure.

For the BLSD query $S : [0..PQ) \to [0..B)$, held by Alice, and for every $q \in [0..Q)$, define

$$v_q \coloneqq qB^P + \sum_{p \in [0..P)} S(qP + p) \cdot B^p \qquad \text{and} \qquad x_q \coloneqq V[v_q].$$

By Lemma 4.6, the BLSD answer is exactly $\bigvee_{q \in [0..Q)} x_q$. Hence, by Proposition 4.9, it suffices to implement the oracle from that lemma with communication costs matching the claimed bounds.

Consider one oracle call on a subset $A \subseteq [0..Q)$. Using public randomness, Alice and Bob sample one random seed r for the data structure. Bob constructs the data structure $D(V, r)$, and together they simulate all Random Access query executions asking for the positions v_q for $q \in A$. The seed r is sampled independently for each oracle call, so different oracle calls are independent. Within one oracle call, the queried bits may be dependent, which is allowed by Proposition 4.9.

We simulate the t probe rounds in lockstep. In round $\ell \in [0..t)$, Alice knows the queried positions v_q, the seed r, and all answers received in earlier rounds. Hence, she can determine the set $C_\ell \subseteq [0..M)$ of all memory cells that at least one of the queries requests in round ℓ. Since $|C_\ell| \leq |A| \leq Q \leq M$, the number of possibilities for C_ℓ is $\binom{M}{\leq |A|} \coloneqq \sum_{s=0}^{|A|} \binom{M}{s}$, so Alice can encode C_ℓ using $\lceil \log \binom{M}{\leq |A|} \rceil \leq \mathcal{O}(|A| \log \frac{2M}{|A|})$ bits and send this encoding to Bob. Bob replies with the contents of the cells in C_ℓ, in a canonical order, using $\mathcal{O}(|C_\ell| w) \leq \mathcal{O}(|A| w)$ bits. After t rounds, Alice can reconstruct the outputs y_q of all simulated Random Access queries for $q \in A$.

For each $q \in A$, the value y_q equals x_q with probability at least $\frac{2}{3}$, because each simulated query is exactly one execution of the bounded-error Random Access data structure on V. Thus, this simulation realizes the oracle required by Proposition 4.9.

Consequently, Proposition 4.9 yields a bounded-error algorithm for $\bigvee_{q \in [0..Q)} x_q$, and hence for the BLSD answer, that uses $\mathcal{O}(\log Q)$ oracle calls in total. By Proposition 4.9, there exist constants $C, c > 0$ such that the subset size a_j in the jth oracle call satisfies $a_j \leq C \cdot Q / 2^{cj}$ for each j. The total number of bits sent by Bob is therefore at most

$$\mathcal{O}\left(tw \cdot \sum_j a_j \right) \leq \mathcal{O}\left(twQ \cdot \sum_{j=0}^{\infty} 2^{-cj} \right) \leq \mathcal{O}(tQw).$$

For Alice, the total communication is

$$\mathcal{O}\left(t \sum_j a_j \log \tfrac{2M}{a_j} \right).$$

We choose $j_0 \coloneqq \lceil \frac{1}{c} \log(Ce) \rceil$ so that $a_j \leq C \cdot Q / 2^{cj} \leq \frac{Q}{e}$ for every $j \geq j_0$ and utilize the fact that $u \mapsto u \log \frac{Q}{u}$ is increasing on $(0, \frac{Q}{e}]$ and decreasing on $[\frac{Q}{e}, Q]$.

For $j < j_0$, we use only the trivial bound $a_j \le Q$ to obtain

$$a_j \log \tfrac{2M}{a_j} = a_j \log \tfrac{2M}{Q} + a_j \log \tfrac{Q}{a_j} \le Q \log \tfrac{2M}{Q} + \tfrac{Q}{e} \log e \le \mathcal{O}\Big(Q \log \tfrac{2M}{Q}\Big),$$

because $M \ge Q$ implies $\log \frac{2M}{Q} \ge 1$.

For $j \ge j_0$, we use $a_j \le C \cdot Q/2^{cj}$ to obtain

$$a_j \log \tfrac{2M}{a_j} = a_j \log \tfrac{2M}{Q} + a_j \log \tfrac{Q}{a_j} \le \tfrac{CQ}{2^{cj}} \log \tfrac{2M}{Q} + \tfrac{CQ(cj - \log C)}{2^{cj}} \le \mathcal{O}\Big(\tfrac{Q}{2^{cj}} \log \tfrac{2M}{Q}\Big) + \mathcal{O}\Big(\tfrac{Qj}{2^{cj}}\Big).$$

Since the series $\sum_{j=0}^{\infty} 2^{-cj}$ and $\sum_{j=0}^{\infty} j2^{-cj}$ converge, the tail contributes

$$\mathcal{O}\Big(Q \log \tfrac{2M}{Q}\Big) + \mathcal{O}(Q) = \mathcal{O}\Big(Q \log \tfrac{2M}{Q}\Big).$$

Adding the $\mathcal{O}(1)$ values of $j < j_0$, we conclude that Alice sends $\mathcal{O}(tQ \log \frac{2M}{Q})$ bits, as claimed. ◀

We also need a technical claim about the monotonicity of SLG Random Access, proved only in the full version [18].

▶ **Lemma 4.12** ([18, Lemma 4.12]). *Consider $n, g, \sigma, n', g', \sigma', M, w, t \in \mathbb{Z}_{>0}$ with $n' \ge n$, $g' \ge g + 3\log n' + 2$, and $\sigma' \ge \sigma$.*

- *If SLG Random Access with parameters (n', g', σ') has (M, w, t)-certificates, then SLG Random Access with parameters (n, g, σ) also has (M, w, t)-certificates.*
- *If SLG Random Access with parameters (n', g', σ') admits an (M, w, t)-bounded-error data structure, then SLG Random Access with parameters (n, g, σ) also admits an (M, w, t)-bounded-error data structure.*

In the final step, we put everything together and change the parametrization to n, g, σ.

▶ **Theorem 4.13.** *Consider integers $n, g, \sigma, M, w, t \in \mathbb{Z}_{>0}$ and a real constant $\epsilon > 0$ such that $n \ge g$, $n \ge \sigma \ge 2$, $Mw \ge g \cdot \log n \cdot (w \log n)^{\epsilon}$, and $g \ge 25 \cdot w^{1+\epsilon} \cdot \log n$.*

If Random Access with parameters n, g, and σ has (M, w, t)-certificates or admits an (M, w, t)-bounded-error data structure, then

$$t \ge \Omega\left(\frac{\log \frac{n}{g}}{\log \frac{Mw}{g \log n}}\right) \ge \Omega\left(\frac{\log \frac{n \log \sigma}{Mw}}{\log \frac{Mw}{g \log n}}\right).$$

Proof. Set $B \coloneqq 1 + \lfloor w^{1+\epsilon} \rfloor$, $P \coloneqq 1 + \lfloor \log_B \frac{n}{g} \rfloor$, and $Q \coloneqq \lfloor \frac{g - 5\log n}{5PB} \rfloor$. Due to $1 \le w$, we have

$$1 \le w^{1+\epsilon} \le B \le 2w^{1+\epsilon}. \tag{1}$$

Moreover, $2 \le g \le n$ implies

$$\max(1, \log_B \tfrac{n}{g}) \le P \le 1 + \log_B \tfrac{n}{g} \le 1 + \log \tfrac{n}{g} \le 1 + \log n \le 2 \log n. \tag{2}$$

The assumption $g \ge 25 \cdot w^{1+\epsilon} \cdot \log n$, together with (1) and (2), yields

$$Q = \left\lfloor \frac{g - 5\log n}{5PB} \right\rfloor \ge \left\lfloor \frac{25 \cdot w^{1+\epsilon} \cdot \log n - 5 \log n}{20 \cdot w^{1+\epsilon} \cdot \log n} \right\rfloor = \left\lfloor \frac{5 \cdot w^{1+\epsilon} - 1}{4 \cdot w^{1+\epsilon}} \right\rfloor \ge 1. \tag{3}$$

Since $\frac{1}{2}x < \lfloor x \rfloor \le x$ holds for $\lfloor x \rfloor \ge 1$, the assumption $g \ge 25 \cdot w^{1+\epsilon} \cdot \log n \ge 25 \log n$ further yields

$$\frac{2g}{25PB} = \frac{g - \frac{1}{5}g}{10PB} \le \frac{g - 5\log n}{10PB} < Q \le \frac{g - 5\log n}{5PB} < \frac{g}{5PB}. \tag{4}$$

Define $n' \coloneqq Q \cdot B^P$ and $g' \coloneqq 5PQB$. Combining (4) and (2) implies

$$n' = Q \cdot B^P \leq \frac{g}{5PB} \cdot B^{1+\log_B \frac{n}{g}} = \frac{g}{5PB} \cdot B \cdot \frac{n}{g} = \frac{n}{5P} \leq n. \tag{5}$$

Moreover, (4) yields

$$g' = 5PQB \leq g - 5\log n \leq g - 3\log n - 2. \tag{6}$$

The bounds in (5) and (6) let us apply Lemma 4.12 to conclude that Random Access with parameters $(n', g', 2)$ has (M, w, t)-certificates or admits a bounded-error data structure.

If $M \leq tQ$, then (4), (1), (2), and the theorem assumption $Mw \geq g \log n \cdot (w \log n)^\epsilon$ imply

$$t \geq \frac{M}{Q} > \frac{5MPB}{g} \geq \frac{5Mw^{1+\epsilon}}{g} > \frac{Mw}{g} \geq \log^{1+\epsilon} n \cdot w^\epsilon > \log n. \tag{7}$$

Since $\log n \geq \log \frac{n}{g}$ and $\log \frac{Mw}{g \log n} \geq \epsilon \log(w \log n) \geq \Omega(1)$, this further yields

$$t > \log n \geq \Omega\left(\frac{\log \frac{n}{g}}{\log \frac{Mw}{g \log n}}\right). \tag{8}$$

We may henceforth assume $M > tQ$. Set $\lambda \coloneqq \frac{\epsilon}{1+\epsilon}$ and $N \coloneqq PQ$.

If Random Access with parameters $(n', g', 2)$ has (M, w, t)-certificates, then Corollary 4.8 implies that BLSD with parameters (N, B) has (M, w, tQ)-certificates. Hence, Corollary 4.3 yields

$$tQ \geq \Omega\left(\frac{NB^{1-\lambda}}{w}\right) \quad \text{or} \quad tQ \geq \Omega\left(\frac{N \log B}{\log \frac{M}{tQ}}\right) \geq \Omega\left(\frac{N \log B}{\log \frac{2M}{Q}}\right).$$

If instead Random Access with parameters $(n', g', 2)$ admits an (M, w, t)-bounded-error data structure, then Corollary 4.11 yields a bounded-error communication protocol for BLSD with parameters (N, B) in which Alice sends $\mathcal{O}\left(tQ \log \frac{2M}{Q}\right)$ bits and Bob sends $\mathcal{O}(tQw)$ bits. The randomized BLSD lower bound therefore implies

$$tQ \geq \Omega\left(\frac{NB^{1-\lambda}}{w}\right) \quad \text{or} \quad tQ \geq \Omega\left(\frac{N \log B}{\log \frac{2M}{Q}}\right).$$

In either case, dividing by Q and using $N = PQ$, we obtain

$$t \geq \Omega\left(\frac{PB^{1-\lambda}}{w}\right) \quad \text{or} \quad t \geq \Omega\left(\frac{P \log B}{\log \frac{2M}{Q}}\right). \tag{9}$$

We first handle the former case. By (1) and the definition of $\lambda = \frac{\epsilon}{1+\epsilon}$, we have

$$B^{1-\lambda} \geq w^{(1+\epsilon)(1-\lambda)} = w,$$

and hence (9) implies $t \geq \Omega(P)$. Moreover, the theorem assumption gives

$$\frac{Mw}{g \log n} \geq (w \log n)^\epsilon \geq (w \log n)^{\Omega(1)} \geq w^{\Omega(1)}. \tag{10}$$

Since (1) implies $B \leq w^{\mathcal{O}(1)}$, (10) yields $\log B \leq \mathcal{O}(\log \frac{Mw}{g \log n})$. Therefore, using (2), we obtain

$$t \geq \Omega(P) \geq \Omega\left(\frac{\log \frac{n}{g}}{\log B}\right) \geq \Omega\left(\frac{\log \frac{n}{g}}{\log \frac{Mw}{g \log n}}\right). \tag{11}$$

It remains to consider the second alternative in (9). Let us first focus on the logarithmic denominator. Due to (4), (1), and (2), we have

$$\log \tfrac{2M}{Q} \le \log \tfrac{25MPB}{g} \le \log \tfrac{M \log n \cdot w^{1+\epsilon}}{g} + \mathcal{O}(1) \le \log \tfrac{Mw}{g \log n} + \log(w^{\epsilon} \cdot \log^2 n) + \mathcal{O}(1).$$

By (10),

$$\log \tfrac{2M}{Q} \le \log \tfrac{Mw}{g \log n} + \log(w^{\epsilon} \cdot \log^2 n) + \mathcal{O}(1) \le \mathcal{O}\Big(\log \tfrac{Mw}{g \log n}\Big). \tag{12}$$

Combining (9), (2), and (12), we conclude that

$$t \ge \Omega\Big(\frac{P \log B}{\log \frac{2M}{Q}}\Big) \ge \Omega\Big(\frac{\log \frac{n}{g}}{\log \frac{Mw}{g \log n}}\Big). \tag{13}$$

Equations (8), (11), and (13) together prove the first inequality in the theorem. For the second inequality, it suffices to plug in the following bound:

$$\log \tfrac{n}{g} = \log \Big(\tfrac{n \log \sigma}{Mw} \cdot \tfrac{Mw}{g \log n} \cdot \tfrac{\log n}{\log \sigma}\Big) \ge \log \tfrac{n \log \sigma}{Mw}, \tag{14}$$

where the inequality uses the assumptions $Mw > g \log n$ and $n \ge \sigma \ge 2$. ◀

Note that Theorem 4.13 does not apply in the following cases:

$g < 25 \cdot w^{1+\epsilon} \cdot \log n$. This corresponds to very small SLGs, which do not model BLSD inputs.

$Mw < g \log n \cdot (w \log n)^{\epsilon}$. In that case, we can derive a weaker lower bound by monotonicity with respect to M (the certificate size can only increase as the data structure size decreases, with everything else unchanged). Applying the bound for $M := \lceil g \cdot w^{\epsilon-1} \cdot \log^{1+\epsilon} n \rceil$ yields

$$t \ge \Omega\Big(\frac{\log \frac{n}{g}}{\log(w \log n)}\Big).$$

The same bound could have been obtained by directly adapting the proof of Theorem 4.13.

Restating Theorem 4.13 in terms of word RAM algorithms, we obtain Theorem 1.2 from the introduction. Here, we present its simplified version for the case of $w = \Theta(\log n)$.

▶ **Corollary 4.14** ([18, Corollary 4.14]). *Consider integers $n, g, \sigma, M, t \in \mathbb{Z}_{>0}$ and a real constant $\epsilon > 0$ such that $n \ge g$, $n \ge \sigma \ge 2$ and $g \ge \Omega(\log^{2+\epsilon} n)$. Suppose that, for every instance of SLG* Random Access *with parameters n, g, σ, there is a data structure of M machine words of $\Theta(\log n)$ bits each whose query algorithm runs in time t in the word RAM model. Then*

$$t \ge \Omega\left(\frac{\log \frac{n}{g}}{\log \max(\frac{M}{g}, \log n)}\right).$$

References

1 Hideo Bannai, Mitsuru Funakoshi, Diptarama Hendrian, Myuji Matsuda, and Simon J. Puglisi. Height-bounded Lempel-Ziv encodings. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 18:1–18:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.18.

2 Ruben Becker, Davide Cenzato, Travis Gagie, Sung-Hwan Kim, Ragnar Groot Koerkamp, Giovanni Manzini, and Nicola Prezza. Compressing suffix trees by path decompositions. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis, editors, *53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026)*, volume 374 of *LIPIcs*, pages 24:1–24:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ICALP.2026.24.

3 Djamal Belazzougui, Paolo Boldi, and Sebastiano Vigna. Predecessor search with distance-sensitive query time, 2012. arXiv:1209.5441.

4 Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *Journal of Computer and System Sciences*, 117:1–22, 2021. doi:10.1016/J.JCSS.2020.11.002.

5 Djamal Belazzougui, Patrick Hagge Cording, Simon J. Puglisi, and Yasuo Tabei. Access, rank, and select in grammar-compressed strings. In Nikhil Bansal and Irene Finocchi, editors, *23rd Annual European Symposium on Algorithms, ESA 2015*, volume 9294 of *LNCS*, pages 142–154. Springer, 2015. doi:10.1007/978-3-662-48350-3_13.

6 Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger search in grammar-compressed strings. *Theory of Computing Systems*, 62(8):1715–1735, 2018. doi:10.1007/S00224-017-9839-9.

7 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, 2015. doi:10.1137/130936889.

8 Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, University of Cambridge, 1994. URL: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

9 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

10 Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Transactions on Algorithms*, 17(1):8:1–8:39, 2021. doi:10.1145/3426473.

11 Ferdinando Cicalese, Zsuzsanna Lipták, Travis Gagie, Gonzalo Navarro, Nicola Prezza, and Cristian Urbina. Incongruity-sensitive access to highly compressed strings, 2026. doi:10.48550/arXiv.2602.04523.

12 Patrick Hagge Cording, Paweł Gawrychowski, and Oren Weimann. Bookmarks in grammar-compressed strings. In Shunsuke Inenaga, Kunihiko Sadakane, and Tetsuya Sakai, editors, *23rd International Symposium on String Processing and Information Retrieval, SPIRE 2016*, volume 9954 of *LNCS*, pages 153–159, 2016. doi:10.1007/978-3-319-46049-9_15.

13 Rajat De and Dominik Kempa. Grammar boosting: A new technique for proving lower bounds for computation over compressed data. In David P. Woodruff, editor, *35th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 3376–3392. SIAM, 2024. doi:10.1137/1.9781611977912.121.

14 Rajat De and Dominik Kempa. Optimal random access and conditional lower bounds for 2D compressed strings. In *37th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026*, pages 1903–1915. Society for Industrial and Applied Mathematics, 2026. doi:10.1137/1.9781611978971.69.

15 Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing base without losing space. In Leonard J. Schulman, editor, *42nd Annual ACM Symposium on Theory of Computing, STOC 2010*, pages 593–602. ACM, 2010. doi:10.1145/1806689.1806771.

16 Akashnil Dutta, Reut Levi, Dana Ron, and Ronitt Rubinfeld. A simple online competitive adaptation of Lempel-Ziv compression with efficient random access support. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2013*, pages 113–122. IEEE, 2013. doi:10.1109/DCC.2013.19.

17 Anouk Duyster and Tomasz Kociumaka. Logarithmic-time internal pattern matching queries in compressed and dynamic texts. In *31st International Symposium on String Processing and Information Retrieval, SPIRE 2024*, LNCS, pages 102–117. Springer, 2024. doi:10.1007/978-3-031-72200-4_8.

18 Anouk Duyster and Tomasz Kociumaka. Random access in grammar-compressed strings: Optimal trade-offs in almost all parameter regimes, 2026. arXiv:2602.10864v2.

19 Peter Elias. Efficient storage and retrieval by content and address of static files. *Journal of the ACM*, 21(2):246–260, 1974. doi:10.1145/321812.321820.

20 Robert Mario Fano. On the number of bits required to implement an associative memory. Technical report, Massachusetts Institute of Technology, 1971. URL: https://csg.csail.mit.edu/CSGArchives/memos/Memo-61.pdf.

21 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

22 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

23 Moses Ganardi. Compression by Contracting Straight-Line Programs. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 45:1–45:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.45.

24 Moses Ganardi, Artur Jeż, and Markus Lohrey. Balancing straight-line programs. *Journal of the ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.

25 Leszek Gasieniec, Roman M. Kolpakov, Igor Potapov, and Paul Sant. Real-time traversal in grammar-based compressed files. In *Data Compression Conference, DCC 2005*, page 458. IEEE Computer Society, 2005. doi:10.1109/DCC.2005.78.

26 Roberto Grossi. *Random Access to High-Order Entropy Compressed Text*, pages 199–215. Springer, 2013. doi:10.1007/978-3-642-40273-9_14.

27 Paul G. Howard and Jeffrey Scott Vitter. Arithmetic coding for data compression. *Proceedings of the IEEE*, 82(6):857–865, 1994. doi:10.1109/5.286189.

28 Tomohiro I. Longest common extensions with recompression. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.18.

29 Dominik Kempa and Tomasz Kociumaka. Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. In *64th Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 1877–1886. IEEE, 2023. doi:10.1109/FOCS57990.2023.00114.

30 Dominik Kempa and Tomasz Kociumaka. Tight lower bounds for central string queries in compressed space. In *37th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026*, pages 1824–1840. Society for Industrial and Applied Mathematics, 2026. doi:10.1137/1.9781611978971.65.

31 Dominik Kempa and Barna Saha. An upper bound and linear-space queries on the LZ-end parsing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2847–2866. SIAM, 2022. doi:10.1137/1.9781611977073.111.

32 John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000. doi:10.1109/18.841160.

33 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive compressibility measure for repetitive sequences. *IEEE Transactions on Information Theory*, 69(4):2074–2092, 2023. doi:10.1109/TIT.2022.3224382.

34 Sebastian Kreft and Gonzalo Navarro. LZ77-like compression with fast random access. In James A. Storer and Michael W. Marcellin, editors, *Data Compression Conference, DCC 2010*, pages 239–248. IEEE Computer Society, 2010. doi:10.1109/DCC.2010.29.

35 Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *53rd Annual Symposium on Foundations of Computer Science, FOCS 2012*, pages 293–301. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.21.

36 Zsuzsanna Lipták, Francesco Masillo, and Gonzalo Navarro. BAT-LZ out of hell. In *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024*, volume 296 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CPM.2024.21.

37 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, 2017. doi:10.1007/s00453-017-0331-3.

38 Gonzalo Navarro. Indexing highly repetitive string collections, part i: Repetitiveness measures. *ACM Computing Surveys*, 54(2):1–31, 2021. doi:10.1145/3434399.

39 Gonzalo Navarro, Francisco Olivares, and Cristian Urbina. Balancing run-length straight-line programs. In Diego Arroyuelo and Barbara Poblete, editors, *29th International Symposium on String Processing and Information Retrieval, SPIRE 2022*, volume 13617 of *LNCS*, pages 117–131. Springer, 2022. doi:10.1007/978-3-031-20643-6_9.

40 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPIcs*, pages 72:1–72:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.72.

41 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. doi:10.1137/09075336X.

42 Mihai Pătraşcu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *55th Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 166–175, 2014. doi:10.1109/FOCS.2014.26.

43 Nicola Prezza. Optimal rank and select queries on dictionary-compressed text. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, pages 4:1–4:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.4.

44 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43, 2007. doi:10.1145/1290672.1290680.

45 Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam D. Smith. Sublinear algorithms for approximating string compressibility. *Algorithmica*, 65(3):685–709, 2013. doi:10.1007/S00453-012-9618-6.

46 Frank Rubin. Experiments in text file compression. *Communications of the ACM*, 19(11):617–623, 1976. doi:10.1145/360363.360368.

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

48 Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 1230–1239. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109693.

49 Hiroki Shibata, Yuto Nakashima, Yutaro Yamaguchi, and Shunsuke Inenaga. LZBE: An LZ-Style Compressor Supporting O(log n)-Time Random Access. In Philip Bille and Nicola Prezza, editors, *37th Annual Symposium on Combinatorial Pattern Matching (CPM 2026)*, volume 369 of *LIPIcs*, pages 34:1–34:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.CPM.2026.34.

50 Sandip Sinha and Omri Weinstein. Local decodability of the Burrows-Wheeler transform. In *51st Annual ACM Symposium on Theory of Computing, STOC 2019*, pages 744–755. ACM, 2019. doi:10.1145/3313276.3316317.

51 Elad Verbin and Wei Yu. Data structure lower bounds on random access to grammar-compressed strings. In Johannes Fischer and Peter Sanders, editors, *24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013*, LNCS, pages 247–258. Springer, 2013. doi:10.1007/978-3-642-38905-4_24.

52 Yaoyu Wang and Yitong Yin. Certificates in data structures. In *41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*, LNCS, pages 1039–1050. Springer, 2014. doi:10.1007/978-3-662-43948-7_86.

53 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

54 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.