Hypersuccinct Trees – New Universal Tree Source Codes for Optimal Compressed Tree Data Structures and Range Minima

J. Ian Munro
University of Waterloo, Canada

Patrick K. Nicholson[1]
Nokia Bell Labs, Dublin, Ireland

Louisa Seelbach Benkner
Universität Siegen, Germany

Sebastian Wild
University of Liverpool, UK

Abstract

We present a new universal source code for distributions of unlabeled binary and ordinal trees that achieves optimal compression to within lower order terms for all tree sources covered by existing universal codes. At the same time, it supports answering many navigational queries on the compressed representation in constant time on the word-RAM; this is not known to be possible for any existing tree compression method. The resulting data structures, "hypersuccinct trees", hence combine the compression achieved by the best known universal codes with the operation support of the best succinct tree data structures.

We apply hypersuccinct trees to obtain a universal compressed data structure for range-minimum queries. It has constant query time and the optimal worst-case space usage of $2n + o(n)$ bits, but the space drops to $1.736n + o(n)$ bits on average for random permutations of n elements, and $2\lg\binom{n}{r} + o(n)$ for arrays with r increasing runs, respectively. Both results are optimal; the former answers an open problem of Davoodi et al. (2014) and Golin et al. (2016).

Compared to prior work on succinct data structures, we do not have to tailor our data structure to specific applications; hypersuccinct trees automatically adapt to the trees at hand. We show that they simultaneously achieve the optimal space usage to within lower order terms for a wide range of distributions over tree shapes, including: binary search trees (BSTs) generated by insertions in random order / Cartesian trees of random arrays, random fringe-balanced BSTs, binary trees with a given number of binary/unary/leaf nodes, random binary tries generated from memoryless sources, full binary trees, unary paths, as well as uniformly chosen weight-balanced BSTs, AVL trees, and left-leaning red-black trees.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Data structures design and analysis

Keywords and phrases analysis of algorithms, universal source code, compressed trees, succinct data structure, succinct trees, tree covering, random binary search trees, range-minimum queries

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.70

Related Version *Extended Version*: `https://arxiv.org/abs/2104.13457`

Funding This work has been supported in part by the Canada Research Chairs Programme and an NSERC Discovery Grant, the DFG research project LO 748/10-2 (QUANT-KOMP), and the NeST (Network Sciences and Technologies) EEECS School initiative of University of Liverpool.

Acknowledgements We thank Conrado Martínez and Markus Lohrey for valuable discussions and feedback on earlier drafts of this paper, as well as our anonymous reviewers for their comments.

[1] Research carried out while P. Nicholson was at Nokia Bell Labs, Ireland.

1 Introduction

As space usage and memory access become the bottlenecks in computation, working directly on a compressed representation ("computing over compressed data") has become a popular field. For text data, substantial progress over the last two decades culminated in compressed text indexing methods that had wide-reaching impact on applications and satisfy strong analytical guarantees. For structured data, the picture is much less developed and clear. In this paper, we develop the analog of entropy-compressed string indices for trees: a data structure that allows one to query a tree stored in compressed form, with optimal query times and space matching the best universal tree codes.

Computing over compressed data became possible by combining techniques from information theory, string compression, and data structures. The central object of study in (classical) information theory is that of a *source* of random strings, whose entropy rate is the fundamental limit for source coding. The ultimate goal in compressing such strings is a *universal code*, which achieves optimal compression (to within lower order terms) for *distributions* of strings from a large class of possible sources *without knowing the used source.*

A classic result in this area is that Lempel-Ziv methods are universal codes for finite-state sources, i.e., sources in which the next symbol's distribution depends on the previous k emitted symbols (see, e.g., [4, § 13]). The same is true for methods based on the Burrows-Wheeler-transform [6] and for grammar-based compression [30]. The latter two results were only shown around 2000, marking a renewed interest in compression methods.

The year 2000 also saw breakthroughs in compressed text indexing, with the first compressed self-indices that can represent a string and support pattern matching queries using $O(nH_0)$ bits of space [23, 24] and $O(nH_k) + o(n \log |\Sigma|)$ bits of space [8] for H_k the kth order empirical entropy of the string (for $k \geq 0$); many improvements have since been obtained on space and query time; (see [37, 1] for surveys and [15] for lower bounds on redundancy; [35, 36] summarizes more recent trends). For strings, computing over compressed data has mainly been achieved.

In this article, we consider structure instead of strings; focusing on one of the simplest forms of structured data: unlabeled binary and ordinal trees. Unlike for strings, the information theory of structured data is still in its infancy. Random sources of binary trees have (to our knowledge) first been suggested and analyzed in 2009 [31]; a more complete formalization then appeared in [43], together with a first universal tree source code.

For trees, computational results predate information-theoretic developments. *Succinct data structures* date back to 1989 [28] and have their roots in storing trees space-efficiently while supporting fast queries. A succinct data structure is allowed to use $\lg U_n(1 + o(1))$ bits of space to represent one out of U_n possible objects of size n – corresponding to a uniform distribution over these objects. This has become a flourishing field, and several succinct data structures for ordinal or cardinal (including binary) trees supporting many operations are known [34]. Apart from the exceptions discussed below (in particular [29, 5]), these methods do not achieve any compression beyond $\lg U_n$ no matter what the input is.

At the other end of the spectrum, more recent representations for highly repetitive trees [2, 3, 12, 14, 17, 18] can realize exponential space savings over $\lg U_n$ in extreme cases, but recent lower bounds [39] imply that these methods cannot simultaneously achieve constant time[2] for queries; they are also not known to be succinct when the tree is not highly compressible.

[2] All running times assume the word-RAM model with word size $w = \Theta(\log n)$.

Table 1 Navigational operations on succinct binary trees. (v denotes a node and i an integer).

parent(v)	the parent of v, same as anc(v, 1)
degree(v)	the number of children of v
left_child(v)	the left child of node v
right_child(v)	the right child of node v
depth(v)	the depth of v, i.e., the number of edges between the root and v
anc(v, i)	the ancestor of node v at depth depth(v) $- i$
nbdesc(v)	the number of descendants of v
height(v)	the height of the subtree rooted at node v
LCA(v, u)	the lowest common ancestor of nodes u and v
leftmost_leaf(v)	the leftmost leaf descendant of v
rightmost_leaf(v)	the rightmost leaf descendant of v
level_leftmost(ℓ)	the leftmost node on level ℓ
level_rightmost(ℓ)	the rightmost node on level ℓ
level_pred(v)	the node immediately to the left of v on the same level
level_succ(v)	the node immediately to the right of v on the same level
node_rank$_X$(v)	the position of v in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$, i.e., in a preorder, postorder, or inorder traversal of the tree
node_select$_X$(i)	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$
leaf_rank(v)	the number of leaves before and including v in preorder
leaf_select(i)	the ith leaf in preorder

In this paper, we fill this gap between succinct trees and dictionary-compressed trees by presenting the first data structure for unlabeled binary trees that answers all queries supported in previous succinct data structures (cf. Table 1) in $O(1)$ time and simultaneously achieves optimal compression over the same tree sources as the best previously known universal tree codes. We also extend the tree-source concepts and our data structure to unlabeled ordinal trees. In contrast to previous succinct trees, we give a single, *universal* data structure, the *hypersuccinct trees*[3], that does not need to be adapted to specific classes or distributions of trees.

Our hypersuccinct trees require only a minor modification of existing succinct tree data structures based on tree covering [20, 25, 7], (namely Huffman coding micro-tree types); the contribution of our work is the careful analysis of the information-theoretic properties of the tree-compression method, the "hypersuccinct code", that underlies these data structures.

As a consequence of our results, we solve an open problem for succinct range-minimum queries (RMQ): Here the task is to construct a data structure from an array $A[1..n]$ of comparable items at preprocessing time that can answer subsequent queries without inspecting A again. The answer to the query $\texttt{RMQ}(i, j)$, for $1 \le i \le j \le n$, is the index (in A) of the (leftmost) minimum in $A[i..j]$, i.e., $\texttt{RMQ}(i, j) = \arg\min_{i \le k \le j} A[k]$. We give a data structure that answers RMQ in constant time using the optimal expected space of $1.736n + o(n)$ bits when the array is a random permutation, (and $2n + o(n)$ in the worst case); previous work either had suboptimal space [5] or $\Omega(n)$ query time [21]. We obtain the same (optimal) space usage for storing a binary search tree (BST) built from insertions in random order ("random BSTs" hereafter). Finally, we show that the space usage of our RMQ data structure is also bounded by $2 \lg \binom{n}{r} + o(n)$ whenever A has r increasing runs, and that this is again best possible.

[3] The name "hypersuccinct trees" is the escalation of the "ultrasuccinct trees" of [29].

Outline. The rest of our article is structured as follows: A comprehensive list of the contributions appears below in Section 2. Section 3 describes our compressed tree encoding. In Section 4, we illustrate the techniques for proving universality of our hypersuccinct code on two well-known types of binary-trees shape distributions – random BSTs and weight-balanced trees – and sketch the extensions necessary for the general results. In Section 5, we present our RMQ data structures. Finally, Section 6 concludes the paper with future directions.

Appendix A contains missing proofs for the RMQ data structures. An extended version of this article is available online (`https://arxiv.org/abs/2104.13457`); it contains formal definitions of tree sources and full proofs for all universality statements.

2 Results

In a binary tree, each node has a left and a right child, either of which can be empty ("null"). For a binary tree t we denote by $|t|$ the number of nodes in t. Unless stated otherwise, $n = |t|$. A binary tree source $\mathcal{S}$ emits a tree t with a certain probability $\mathbb{P}_{\mathcal{S}}[t]$ (potentially $\mathbb{P}_{\mathcal{S}}[t] = 0$); we write $\mathbb{P}[t]$ if $\mathcal{S}$ is clear from the context. $\lg(1/0)$ is taken to mean $+\infty$.

▶ **Theorem 1** (Hypersuccinct binary trees). *Let t be a binary tree over n nodes. The hypersuccinct representation of t supports all queries from Table 1 in $O(1)$ time and uses $|\mathsf{H}(t)| + o(n)$ bits of space, where*

$$|\mathsf{H}(t)| \quad \le \quad \min\left\{2n+1,\ \min_{\mathcal{S}} \lg\left(\frac{1}{\mathbb{P}_{\mathcal{S}}[t]}\right) + o(n)\right\},$$

and $\mathbb{P}_{\mathcal{S}}[t]$ is the probability that t is emitted by source $\mathcal{S}$. The minimum is taken over all binary-tree sources $\mathcal{S}$ in the following families (which are explained in Table 4):

(i) *memoryless node-type processes,*
(ii) *kth-order node-type processes (for $k = o(\log n)$),*
(iii) *monotonic fixed-size sources,*
(iv) *worst-case fringe-dominated fixed-size sources,*
(v) *monotonic fixed-height sources,*
(vi) *worst-case fringe-dominated fixed-height sources,*
(vii) *tame uniform subclass sources.*

▶ **Corollary 2** (Hypersuccinct binary trees: Examples & Empirical entropies). *Hypersuccinct trees achieve optimal compression to within lower order terms for all example distributions listed in Table 3. Moreover, for every binary tree t, we have:*

(i) $|\mathsf{H}(t)| \le H_k^{\text{type}}(t) + o(n)$ *with $H_k^{\text{type}}(t)$ the (unnormalized) kth-order empirical entropy of node types (leaf, left-unary, binary, or right-unary) for $k = o(\log n)$.*
(ii) $|\mathsf{H}(t)| \le H_{st}(t) + o(n)$ *with $H_{st}(t)$ the "subtree-size entropy", i.e., the sum of the logarithm of the subtree size of v for all nodes v in t, (a.k.a. the splay-tree potential).*

The hypersuccinct code is a *universal code* for the families of binary-tree sources listed in Theorem 1 with bounded maximal pointwise redundancy. We also present a more general class of sources, for which our code achieves $o(n)$ *expected* redundancy in the appendix; see also Table 4.

To our knowledge, the list in Theorem 1 is a comprehensive account of *all* concrete binary-tree sources for which any universal code is known. Remarkably, in all cases the bounds on redundancies proven for the hypersuccinct code are identical (up to constant factors) to those known for existing universal binary-tree codes. *Our hypersuccinct code thus achieves the same compression as all previous universal codes, but simultaneously supports constant-time queries on the compressed representation with $o(n)$ overhead.*

Table 2 Overview of random tree sources for binary and ordinal trees.

Name	Notation	**Intuition**	**Formal Definition** of $\mathbb{P}[t]$
Memoryless Processes	τ	A binary tree is constructed top-down, drawing each node's *type* (0 = leaf, 1 = left-unary, 2 = binary, 3 = right-unary) i.i.d. according to the distribution $(\tau_0, \tau_1, \tau_2, \tau_3)$.	$\mathbb{P}[t] = \prod_{v \in t} \tau(\text{type}(v))$
Higher-order Processes	$(\tau_z)_z$	A binary tree is constructed top-down, drawing node v's *type* according to $\tau_{h_k(v)} : \{0,1,2,3\} \to [0,1]$, which depends on the types of the k closest *ancestors* of v.	$\mathbb{P}[t] = \prod_{v \in t} \tau_{h_k(v)}(\text{type}(v))$
Fixed-size Binary Tree Sources	$\mathcal{S}_{fs}(p)$	A binary tree of size n is constructed top-down, asking source p at each node for its left- and right *subtree size*.	$\mathbb{P}[t] = \prod_{v \in t} p(\lvert t_\ell(v)\rvert, \lvert t_r(v)\rvert)$ $t_{\ell/r}(v)$= left/right subtree of v
Fixed-height Binary Tree Sources	$\mathcal{S}_{fh}(p)$	A binary tree of height h is constructed top-down, asking source p at each node for a left and right subtree *height*.	$\mathbb{P}[t] = \prod_{v \in t} p(h(t_\ell(v)), h(t_r(v)))$ $h(t)$ = height of t
Uniform Subclass Sources	$\mathcal{U}_{\mathcal{P}}$	A binary tree is drawn uniformly at random from the set $\mathcal{T}_n(\mathcal{P})$ of all binary trees of size n that satisfy property $\mathcal{P}$.	$\mathbb{P}[t] = \dfrac{1}{\lvert\mathcal{T}_n(\mathcal{P})\rvert}$
Memoryless Ordinal Tree Sources	d	An ordinal tree is constructed top-down, drawing each node v's degree $\deg(v)$ according to distribution $d = (d_0, d_1, \ldots)$.	$\mathbb{P}[t] = \prod_{v \in t} d_{\deg(v)}$
Fixed-size Ordinal Tree Sources	$\mathfrak{S}_{fs}(p)$	An ordinal tree of size n is constructed top-down, asking source p at each node for the number and sizes of the subtrees.	$\mathbb{P}[t] = \prod_{v \in t} p(\lvert t_1[v]\rvert, \ldots, \lvert t_{\deg(v)}[v]\rvert)$

In terms of queries, previous solutions either have suboptimal query times [2, 3, 14], higher space usage [39], or rely on tailoring the representation to a specific subclass of trees [29, 7] to achieve good space and time for precisely these instances, but they fail to generalize to other use cases. Some also do not support all queries. We give a detailed comparison with the state of the art in the extended version of this article.

We focus here on our results for binary trees. In the extended version of this article, we extend the above notions of tree sources (except fixed-height sources) to ordinal trees, which has not been done to our knowledge. Moreover, we extend both our code and data structure to ordinal trees, and show their universality for these sources.

3 From Tree Covering to Hypersuccinct Trees

Our universally compressed tree data structures are based on *tree covering* [20, 25, 7]: A (binary or ordinal) tree t is decomposed into *mini trees*, each of which is further decomposed into *micro trees*; the size of the latter, $B = B(n) = \lg n/8$, is chosen so that we can tabulate all possible shapes of micro trees and the answers to various micro-tree-local queries in one global lookup table (the "Four-Russian Table" technique). For each micro tree, its local shape is stored, e.g., using the balanced-parenthesis (BP) encoding, using a total of exactly $2n$ bits (independent of the tree shape). Using additional data structures occupying only $o(n)$ bits of space, a long list of operations can be supported in constant time (Table 1). The space usage of this representation is optimal to within lower order terms for the worst case, since $\lg C_n \sim 2n$ bits are necessary to distinguish all $C_n = \binom{2n}{n}/(n+1)$ trees of n nodes. (This worst-case bound applies both to ordinal trees and binary trees).

Table 3 An overview over the concrete examples of tree-shape distributions that our hypersuccinct code compresses optimally (up to lower-order terms).

Tree-Shape Distribution	**Entropy**	**Corresponding Source**
(Uniformly random) binary trees of size n	$2n$	Memoryless binary, monotonic fixed-size binary
(Uniformly random) full binary trees of size n	n	Memoryless binary
(Uniformly random) unary paths of length n	n	Memoryless binary
(Uniformly random) Motzkin trees of size n	$1.585n$	Memoryless binary
Binary search trees generated by insertions in random order ("random BSTs")	$1.736n$	Monotonic fixed-size binary
Binomial random trees	$P(\lg n)n$[a)]	Average-case fringe-dominated fixed-size binary
Almost paths	—[b)]	Monotonic fixed-size binary
Random fringe-balanced binary search trees	—[b)]	Average-case fringe-dominated fixed-size binary
(Uniformly random) AVL trees of height h	—[b)]	Worst-case fringe-dominated fixed-height binary
(Uniformly random) weight-balanced binary trees of size n	—[b)]	Worst-case fringe-dominated fixed-size binary
(Uniformly random) AVL trees of size n	$0.938n$	Uniform-subclass
(Uniformly random) left-leaning red-black trees of size n	$0.879n$	Uniform-subclass
(Uniformly random) full m-ary trees of size n	$\lg(\frac{m}{m-1})n$	Memoryless ordinal
Uniform composition trees	—[b)]	Monotonic fixed-size ordinal
Random LRM-trees	$1.736n$	Monotonic fixed-size ordinal

a) Here P is a nonconstant, continuous, periodic function with period 1.
b) No (concise) asymptotic approximation known.

Table 4 Sufficient conditions under which we show universality of our hypersuccinct code H for binary trees. Proofs are given in the extended version of this article (`https://arxiv.org/abs/2104.13457`).

Family of sources	**Restriction**	**Redundancy**
Memoryless node-type	—	$O(n \log\log n/\log n)$
kth-order node-type	—	$O((nk + n\log\log n)/\log n)$
Monotonic fixed-size	$p(\ell,r) \ge p(\ell+1,r)$ and $p(\ell,r) \ge p(\ell,r+1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log\log n/\log n)$
Worst-case fringe-dominated fixed-size	$n_{\ge B}(t) = o(n/\log\log n)$ for all t with $\mathbb{P}[t] > 0$; $n_{\ge B}(t) = \#$nodes with subtree size in $\Omega(\log n)$	$O\big(n_{\ge B}(t)\log\log n + n\log\log n/\log n\big)$
Weight-balanced fixed-size	$\sum_{\frac{n}{c} \le \ell \le n-\frac{n}{c}} p(\ell-1, n-\ell-1) = 1$ for constant $c \ge 3$	$O(n \log\log n/\log n)$
Average-case fringe-dominated fixed-size	$\mathbb{E}[n_{\ge B}(T)] = o(n/\log\log n)$ for random T generated by source $\mathcal{S}$	$O\big(n_{\ge B}(t)\log\log n + n\log\log n/\log n\big)$[a)]
Monotonic fixed-height	$p(\ell,r) \ge p(\ell+1,r)$ and $p(\ell,r) \ge p(\ell,r+1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log\log n/\log n)$
Worst-case fringe-dominated fixed-height	$n_{\ge B}(t) = o(n/\log\log n)$ for all t with $\mathbb{P}[t] > 0$	$O\big(n_{\ge B}(t)\log\log n + n\log\log n/\log n\big)$
Tame uniform-subclass	class of trees $\mathcal{T}_n(\mathcal{P})$ is hereditary (i.e., closed under taking subtrees), $n_{\ge B}(t) = o(n/\log\log n)$ for $t \in \mathcal{T}_n(\mathcal{P})$, $\lg\lvert\mathcal{T}_n(\mathcal{P})\rvert = cn + o(n)$ for constant $c > 0$, heavy-twigged: if v has subtree size $\Omega(\log n)$, v's subtrees have size $\omega(1)$	$o(n)$

a) Stated redundancy is achieved in expectation for a random tree t generated by the source.

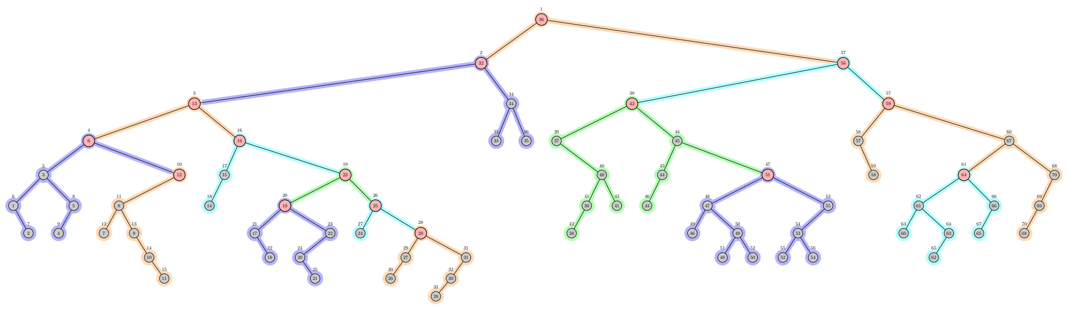

Figure 1 Example binary tree with $n = 70$ nodes and micro trees computed by the Farzan-Munro tree-covering algorithm [7] with parameter $B = 6$. The micro trees are indicated by colors. The algorithm guarantees that each node is part of exactly one micro tree and that each micro tree has at most three edges shared with other micro trees, namely to a parent, a left- and a right-child micro tree.

A core observation is that the dominant space in tree-covering data structures comes from storing the micro-tree types, and these can be further compressed using a different code. This has been used in an ad-hoc manner for specific tree classes [7, 5, 16], but has not been investigated systematically. A natural idea is to use a Huffman code for the micro tree types to simultaneously beat the compression of all these special cases; we dub this as the "Four Russians and One American"[4] trick. Applying it to the data structures based on the Farzan-Munro tree-covering algorithm [7] yields our hypersuccinct trees.

The main contribution of our present work is the careful analysis of the potential of the Four Russians and One American trick for (binary and ordinal) tree source coding. As an immediate corollary, we obtain a single data structure that achieves optimal compression for all special cases covered in previous work, plus a much wider class of distributions over trees for which no efficient data structure was previously known.

Our analysis builds on previous work on tree compression, specifically DAG compression and tree straight-line programs (TSLPs) [32]. Our core idea is to interpret (parts of the) tree-covering data structures as a *code* for trees, the "hypersuccinct code": it stores the type, i.e., the local shape, of all micro trees separately from how they interface to form the entire tree. Intuitively, our hypersuccinct code is a restricted version of a grammar-based tree code, where we enforce having nonterminals for certain subtrees;[5] we strengthen and extend existing universality proofs from general grammar-based tree codes to the restricted hypersuccinct code.

4 Universality for Fixed-Size Sources

In this section, we sketch the proof that our hypersuccinct trees achieve optimal compression for two exemplary tree-shape distributions: random binary search trees and uniform weight-balanced trees (defined below). These examples serve to illustrate the proof techniques and

[4] It deems us only fair to do D. A. Huffman the same questionable honor of reducing the person to a country of residence that V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev have experienced ever since their table-lookup technique has become known as the "Four-Russians trick".

[5] Differences in technical details make the direct comparison difficult, though: in TSLPs, holes in contexts must be stored (and encoded) alongside the local shapes as they are both part of the right-hand side of productions; in our hypersuccinct code, we separately encode the shapes of micro trees and the positions of portals, potentially gaining a small advantage. Our comment thus remains a motivational hint as to why similar analysis techniques are useful in both cases, but falls short of providing a formal reduction.

to showcase the versatility of the approach. The extension to the general sufficient conditions from Table 4 and full details of computations are spelled out in the extended version of the article.

By *random BSTs*, we mean the distribution of tree shapes obtained by successively inserting n keys in random order into an (initially empty) unbalanced binary search tree (BST). We obtain random BSTs from a fixed-size tree source $\mathcal{S}_{fs}(p_{bst})$ with $p_{bst}(\ell, n-1-\ell) = \frac{1}{n}$ for all $\ell \in \{0, \ldots, n-1\}$ and $n \in \mathbb{N}_{\geq 1}$, i.e., making every possible split equally likely. (Any left subtree size ℓ is equally likely in a random BST of a given size n.) Hence, $\mathbb{P}[t] = \prod_{v \in t} 1/|t[v]|$ where $t[v]$ is the subtree rooted at v and $|t[v]|$ its size (in number of nodes).

The second example are the shapes of uniformly random *weight-balanced BSTs* (BB[α]-trees, [38]): A binary tree t is α-weight-balanced if we have for every node v in t that $\min\{|t_\ell[v]|, |t_r[v]|\} + 1 \geq \alpha(|t[v]| + 1)$. Here $t_\ell[v]$ resp. $t_r[v]$ are the left resp. right subtrees of $t[v]$. We denote the set of α-weight-balanced trees of size n by $\mathcal{T}_n(\mathcal{W}_\alpha)$. We obtain random α-weight-balanced trees from another fixed-size source $\mathcal{S}_{fs}(p_{wb})$ with

$$p_{wb}(\ell, n-1-\ell) \quad = \quad \begin{cases} \dfrac{|\mathcal{T}_\ell(\mathcal{W}_\alpha)||\mathcal{T}_{n-1-\ell}(\mathcal{W}_\alpha)|}{|\mathcal{T}_n(\mathcal{W}_\alpha)|} & \text{if } \min\{\ell+1, n-\ell\} \geq \alpha(n+1), \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that this yields the uniform probability distribution on $\mathcal{T}_n(\mathcal{W}_\alpha)$, i.e., with $\mathbb{P}[t] = 1/|\mathcal{T}_n(\mathcal{W}_\alpha)|$ for $t \in \mathcal{T}_n(\mathcal{W}_\alpha)$ and $\mathbb{P}[t] = 0$ otherwise. We note that computing $|\mathcal{T}_n(\mathcal{W}_\alpha)|$ is a formidable challenge in combinatorics, but we never have to do so; we only require the *existence* of the fixed-size source for weight-balanced BSTs.

The *hypersuccinct code* $\mathsf{H}(t)$ is formed by partitioning the nodes of a given binary tree t into $m = \Theta(n/\log n)$ micro trees $\mu_1, \ldots, \mu_m$, each of which is a connected subtree of at most $\mu = O(\log n)$ nodes; an example is shown in Figure 1. Previous work on tree covering shows how to compute these and how to encode everything but the local shape of the micro trees in $o(n)$ bits of space [7]. (For a mere encoding, $O(n \log\log n/\log n)$ bits suffice.

The dominant part of the hypersuccinct code is the list of *types* of all micro trees, i.e., the (local) shapes of the induced subtrees formed by the set of nodes in the micro trees. Let C be a Huffman code for the string $\mu_1, \ldots, \mu_m$, where we identify micro trees with their types. For a variety of different tree sources $\mathcal{S}$, we can prove that $\sum_{i=1}^m |C(\mu_i)|$, the total length of codewords for the micro trees, is upper bounded by $\lg(1/\mathbb{P}[t])$ + lower-order terms, where $\mathbb{P}[t]$ is the probability that t is emitted by $\mathcal{S}$; this is the best possible code length to within lower order terms achievable for that source. We will now show this for our two example distributions.

4.1 Random BSTs

The proof consists of four steps that can be summarized as follows:

Step 1
Construct a source-specific micro-tree encoding
$D_{\mathcal{S}}\colon \{\mu_1, \ldots, \mu_m\} \to \{0,1\}^\star$
Goal: $|D_{\mathcal{S}}(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$

Step 2
By optimality of Huffman codes:
$$\sum_{i=1}^m |C(\mu_i)| \leq \sum_{i=1}^m |D_{\mathcal{S}}(\mu_i)|$$

Step 3
Use properties of $\mathcal{S}$ to show that
$$\prod_{i=1}^m \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$$

Step 4
Conclude
$$\sum_{i=1}^m |C(\mu_i)| \approx \lg(1/\mathbb{P}[t])$$

Steps 2 and 4 do not depend on the source and indeed follow immediately; Steps 1 and 3 are the creative parts. Ignoring proper tracing of error terms, the result then follows as

$$|\mathsf{H}(t)| \;\sim\; \sum_{i=1}^{n} |C(\mu_i)| \;\le\; \sum_{i=1}^{n} |D_{\mathcal{S}}(\mu_i)| \;\le\; \sum_{i=1}^{n} \lg(1/\mathbb{P}[\mu_i]) \;\lesssim\; \lg(1/\mathbb{P}[t]).$$

Let us consider $\mathcal{S}_{fs}(p_{bst})$, the fixed-size source producing (shapes of) random BSTs, and address these steps independently.

Our task in Step 1 is to find a code $D_{\mathcal{S}}$ for the micro-tree types that can occur in t, so that $|D_{\mathcal{S}}(\mu_i)| = \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i]) + O(\log\log n)$. This code may rely on the decoder to have knowledge of $\mathcal{S}$. For random BSTs, $D_{\mathcal{S}}(t)$ can be constructed as follows: We initially store n using Elias gamma code[6] and then, following a depth-first (preorder) traversal of the tree, we encode the size of the left subtree using *arithmetic coding*. Inductively, the size of the currently encoded node is always known, and the source-specific code is allowed to use the probability distributions hardwired into $\mathcal{S}$ without storing them; for random BSTs, we simply encode a number uniformly distributed in $[0..s-1]$ at a node with subtree size s, using exactly $\lg s$ bits. Apart from storing the initial size and the small additive overhead from arithmetic coding, the code length of this "depth-first arithmetic tree code" is best possible: $|D_{\mathcal{S}}(t)| \le \lg(1/\mathbb{P}[t]) + O(\log|t|)$. This concludes Step 1.

For Step 3, we have to show that the probability for the entire tree t is at most the product of the probabilities for all micro-trees. Recall that $\mu_1, \ldots, \mu_m$ are the micro trees in t. We can write $\mathbb{P}[t]$ as a product over contributions of individual nodes, and can collect factors in $\mathbb{P}[t]$ according to micro trees; this works for any fixed-size source. For random BSTs, we can use the "monotonicity" of node contributions to show

$$\mathbb{P}[t] \;=\; \prod_{v\in t} \frac{1}{|t[v]|} \;=\; \prod_{i=1}^{m}\prod_{v\in\mu_i} \frac{1}{|t[v]|} \;\le\; \prod_{i=1}^{m}\prod_{v\in\mu_i} \frac{1}{|\mu_i[v]|} \;=\; \prod_{i=1}^{m} \mathbb{P}[\mu_i].$$

That completes Step 3, and hence the proof that $|\mathsf{H}(t)| \le \mathbb{P}_{\mathcal{S}_{fs}(p_{bst})}[t] + o(n)$.

4.2 Weight-balanced trees

Let us now consider uniformly random weight-balanced trees, i.e., the source $\mathcal{S} = \mathcal{S}_{fs}(p_{wb})$. We would like to follow the same template as above; however, this is not possible: Step 3 from above is in general not true anymore. The reason is that it is not clear whether the "non-fringe" micro trees, i.e., those that do not contain all descendants of the micro-tree root, have non-zero probability under $\mathcal{S}$. (A subtree of a tree is called *fringe*, if it consists of a node and all its descendants). Such micro trees will also make Step 1 impossible as they would require a code length of 0. While this issue is inevitable in general (Remark 4), we can under certain conditions circumvent Steps 1 and 3 altogether by directly bounding $\sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \le \mathbb{P}[t] + o(n)$.

As a first observation, note that it suffices to have $|D_{\mathcal{S}}(\mu_i)| = \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i]) + O(\log\log n)$ for *all but a vanishing fraction* of the micro trees in any tree t; then we can still hope to show $\sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \le \mathbb{P}[t] + o(n)$ overall. Second, it is known [12] that weight-balanced trees are *"fringe dominated"* in the following sense: Denoting by $n_{\ge B}(t)$ the number of "heavy" nodes, i.e., v in t with $|t[v]| \ge B = \lg n/8$, we have $n_{\ge B}(t) = O(n/B) = o(n)$ for

[6] Elias gamma code $\gamma : \mathbb{N} \to \{0,1\}^\star$ encodes an integer $n \ge 1$ using $2\lfloor \lg n \rfloor + 1$ bits by prefixing the binary representation of n with that representation's length encoded in unary.

every weight-balanced tree $t \in \mathcal{T}_n(\mathcal{W}_\alpha)$. Since only a vanishing fraction of nodes are heavy, one might hope that also only a vanishing fraction of micro trees are non-fringe, making the above route succeed. Unfortunately, that is not the case; the non-fringe micro trees can be a constant fraction of all micro trees.

Notwithstanding this issue, a more sophisticated micro-tree code $D_\mathcal{S}$ allows us to proceed. $D_\mathcal{S}$ encodes any *fringe* micro tree using a depth-first arithmetic code as for random BSTs. Any non-fringe micro tree μ_i, however, is broken up into the subtree of heavy nodes, the "boughs" of μ_i, and (fringe) subtrees $f_{i,j}$ hanging off the boughs. It is a property of the Farzan-Munro algorithm that every micro-tree root is heavy, hence all $f_{i,j}$ are indeed entirely contained within μ_i.

$D_\mathcal{S}(\mu_i)$ then first encodes the bough nodes using 2 bits per node (using a BP representation for the boughs subtree) and then appends the depth-first arithmetic code for the $f_{i,j}$ (in left-to-right order). While this does not actually achieve $|D_\mathcal{S}(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$ for entire micro trees μ_i, it does so for all the fringe subtrees $f_{i,j}$. Any node not contained in a fringe subtree $f_{i,j}$ must be part of a bough and hence heavy; by the fringe-dominance property, these nodes form a vanishing fraction of all nodes and hence contribute $o(n)$ bits overall. This shows that $|\mathsf{H}(t)| \le \mathbb{P}_{\mathcal{S}_{fs}(p_{wb})}[t] + o(n)$.

▶ Remark 3 (A simple code whose analysis isn't). It is worth pointing out that the source specific code $D_\mathcal{S}$ is only a vehicle for the *analysis* of $|\mathsf{H}(t)|$; the complicated encodings $D_\mathcal{S}$ do not ever need to be computed when using our codes or data structures.

4.3 Other Sources

For memoryless sources, the analysis follows the four-step template, and is indeed easier than the random BSTs since Step 3 becomes trivial. For higher-order sources, in order to know the node types of the k closest ancestors (in t) of all nodes of depth $\le k$ in μ_i, we prefix the depth-first arithmetic code by the node types of the k closest ancestors of the root of μ_i. Then the k ancestor types are known inductively for all nodes in a preorder traversal of μ_i.

The tame uniform-subclass sources require the most technical proof, but it is conceptually similar to the weight-balanced trees from above. The source-specific encoding for fringe subtrees is trivial here: we can simply use the rank in an enumeration of all trees of a given size, prefixed by the size of the subtree. Using the tameness conditions, one can show that a similar decomposition into boughs and fringe subtrees yields an optimal code length for almost all nodes. Details are deferred to the extended version of this article.

* * *

Together with the observations from Section 3 this yields Theorem 1. We obtained similar results for ordinal trees; details are deferred to the extended version of this article.

▶ Remark 4 (Restrictions are inevitable). We point out that some restrictions like the ones discussed above cannot possibly be overcome in general. Zhang, Yang, and Kieffer [43] prove that the unrestricted class of fixed-size sources (leaf-centric binary tree sources in their terminology) does not allow a universal code, even when only considering expected redundancy. The same is true for unrestricted fixed-height and uniform-subclass sources. While each is a natural formalism to describe possible binary-tree sources, additional conditions are strictly necessary for any interesting compression statements to be made. Our sufficient conditions are the weakest such restrictions for which any universal source code is known to exist ([43, 13, 40]), even without the requirement of efficient queries.

5 Hypersuccinct Range-Minimum Queries

We now show how hypersuccinct trees imply an optimal-space solution for the range-minimum query (RMQ) problem.[7] Let $A[1..n]$ store the numbers $x_1, \ldots, x_n$, i.e., x_j is stored at index j for $1 \le j \le n$. While duplicates naturally arise in some applications, e.g., in the longest-common extension (LCE) problem, we assume here that $x_1, \ldots, x_n$ are n distinct numbers to simplify the presentation. However, our RMQ solution works regardless of which minimum-value index is to be returned so long as the tie breaking rule is deterministic and fixed at construction time.

5.1 Cartesian Trees

The *Cartesian tree* T for $x_1, \ldots, x_n$ (resp. for $A[1..n]$) is a binary tree defined recursively as follows: If $n = 0$, it is the empty tree ("null"). Otherwise it consists of a root whose left child is the Cartesian tree for $x_1, \ldots, x_{j-1}$ and its right child is the Cartesian tree for $x_{j+1}, \ldots, x_n$ where j is the position of the minimum, $j = \arg\min_k A[k]$. A classic observation of Gabow et al. [11] is that range-minimum queries on A are equivalent to lowest-common-ancestor (LCA) queries on T when identifying nodes with their inorder rank:

$$\mathtt{RMQ}_A(i, j) \quad = \quad \mathtt{node\_rank}_{\mathtt{IN}}\Big(\mathtt{LCA}\big(\mathtt{node\_select}_{\mathtt{IN}}(i), \mathtt{node\_select}_{\mathtt{IN}}(j)\big)\Big).$$

We can thus reduce an RMQ instance (on an arbitrary input) to an LCA instance on binary trees of the same size; (the number of nodes in T equals the length of the array).

5.2 Random RMQ

We first consider the random permutation model for RMQ: Every (relative) ordering of the elements in $A[1..n]$ is equally likely. Without loss of generality, we identify the n elements with their ranks, i.e., $A[1..n]$ contains a random permutation of $[1..n]$. We refer to this as a random RMQ instance.

We can characterize the distribution of the Cartesian tree associated with such a random RMQ instance: Since the minimum in a random permutation is located at every position $i \in [n]$ with probability $\frac{1}{n}$, the inorder index of the root is uniformly distributed in $[n]$. Apart from renaming, the subarrays $A[1..i-1]$ (resp. $A[i+1..n]$) contain a random permutation of $i-1$ (resp. $n-i$) elements, and these two permutations are independent of each other conditional on their sizes. Cartesian trees of random RMQ instances thus have the same distribution as random BSTs, and in particular shape t arises with probability $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{|t[v]|}$. The former are also known as random increasing binary trees [10, Ex. II.17 & Ex. III.33]).

Since the sets of answers to range-minimum queries is in bijection with Cartesian trees, the entropy H_n of the distribution of the shape of the Cartesian tree (and hence random BSTs) gives an information-theoretic lower bound for the space required by any RMQ data structure (in the encoding model studied here). Kieffer, Yang and Szpankowski [31] show[8]

[7] A technical report containing preliminary results for random RMQ, but including more details on the data structure aspects of our solution, can be found on arXiv [33].

[8] Hwang and Neininger [26] showed already in 2002 that the quicksort recurrence can be solved explicitly for arbitrary toll functions. H_n satisfies this recurrence with toll function $\lg n$, hence they implicitly proved Equation (1).

that the entropy of random BSTs $H_n = \mathbb{E}_T\big[\lg(1/\mathbb{P}[T])\big] = \mathbb{E}_T\big[\sum_{v\in T}\lg(|T[v]|)\big]$ is

$$H_n \;=\; \lg(n) + 2(n+1)\sum_{i=2}^{n-1}\frac{\lg i}{(i+2)(i+1)} \;\sim\; 2n\sum_{i=2}^{\infty}\frac{\lg i}{(i+2)(i+1)} \;\approx\; 1.7363771n. \tag{1}$$

With these preparations, we are ready to prove our first result on range-minimum queries.

▶ **Corollary 5** (Average-case optimal succinct RMQ). *There is a data structure that supports (static) range-minimum queries on an array A of n (distinct) numbers in $O(1)$ worst-case time and which occupies $H_n + o(n) \approx 1.736n + o(n)$ bits of space on average over all possible permutations of the elements in A. The worst case space usage is $2n + o(n)$ bits.*

Proof. We construct a hypersuccinct tree on the Cartesian tree for A. It supports $\texttt{node\_rank}_{\texttt{IN}}$, $\texttt{node\_select}_{\texttt{IN}}$, and `LCA` in $O(1)$ time and thus `RMQ` in constant time without access to A. By Corollary 2, the space usage of hypersuccinct trees is at most $\min\{2n, \lg(1/\mathbb{P}[t])\} + o(n)$ for $\mathbb{P}[t] = \prod_{v\in t}\frac{1}{|t[v]|}$. By the above observations, this is the probability to obtain t as the Cartesian trees of a random permutation, so we store t with maximal pointwise redundancy of $o(n)$, hence also $o(n)$ expected redundancy over the entropy $H_n \sim 1.736n$. ◀

5.3 RMQ with Runs

A second example of compressible RMQ instances results from partially sorted arrays. Suppose that $A[1..n]$ can be split into r runs, i.e., maximal contiguous ranges $[j_i, j_{i+1}-1]$, $(i = 1,\ldots,r$ with $j_1 = 1$ and $j_{r+1} = n+1)$, so that $A[j_i] \le A[j_i+1] \le \cdots \le A[j_{i+1}-1]$.

▶ **Theorem 6** (Lower bound for RMQ with runs). *Any range-minimum data structure in the encoding model for an array of length n that contains r runs must occupy at least $\lg N_{n,r} \ge 2\lg\binom{n}{r} - O(\log n)$ bits of space where $N_{n,r} = \frac{1}{n}\binom{n}{r}\binom{n}{r-1}$ are the* Narayana numbers.

The proof follows from a bijection between Cartesian trees on sequences of length n with exactly r runs and mountain-valley diagrams (a.k.a. Dyck paths) of length $2n$ with exactly r “peaks”; the latter is known to be counted by the *Narayana numbers* [27]. Details are given in Appendix A.

▶ **Corollary 7** (Optimal succinct RMQ with runs). *There is a data structure that supports (static) range-minimum queries on an array A of n numbers that consists of r runs in $O(1)$ worst-case time and which occupies $2\lg\binom{n}{r} + o(n) \le 2n + o(n)$ bits of space.*

This follows from the observation that a node’s type in the Cartesian tree, i.e., whether or not its left resp. right child is empty, closely reflects the runs in A: A binary node marks the beginning of a non-singleton run, a leaf node marks the last position in a non-singleton run, a right-unary node (i.e., left child empty, right child nonempty) is a middle node of a run, and a left-unary node corresponds to a singleton run. With $s \in [0..r]$ the number of singleton runs, we can bound the space for a hypersuccinct tree in terms of its empirical node-type entropy by $H_0^{\text{type}}(T) + o(n) = nH\big(\frac{r-s}{n}, \frac{s}{n}, \frac{n-2r+s}{n}, \frac{r-s}{n}\big) + o(n)$, which can be shown to be no more than $2\lg\binom{n}{r} + o(n)$ for any value of s; again, details are deferred to Section A.

* * *

We close by pointing out that hypersuccinct trees simultaneously achieve the optimal bounds for RMQ on random permutations and arrays with r runs without taking explicit precautions for either. The same is true for any other shape distributions of Cartesian trees that can be written as one of the sources from Table 4.

6 Conclusion

We presented the first succinct tree data structures with optimally adaptive space usage for a large variety of random tree sources, both for binary trees and for ordinal trees. This is an important step towards the goal of efficient computation over compressed structures, and has immediate applications, e.g., as illustrated above for the range-minimum problem.

A goal for future work is to reduce the redundancy of $o(n)$, which becomes dominant for sources with sublinear entropy. While this has been considered for tree covering in principle [41], many details remain to be thoroughly investigated.

For very compressible trees, the space savings in hypersuccinct trees are no longer competitive. On the other hand, with current methods for random access on dictionary-compressed sequences, constant-time queries are not possible in the regime of mildly compressible strings; the same applies to known approaches to represent trees. An interesting question is whether these opposing approaches can be combined in a way to complement each other's strengths. We leave this direction for future work.

References

1 Djamal Belazzougui, Veli Mäkinen, and Daniel Valenzuela. Compressed suffix array. In *Encyclopedia of Algorithms*, pages 1–6. Springer US, 2014. doi:10.1007/978-3-642-27848-8_82-2.

2 Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Information and Computation*, 243:166–177, 2015. doi:10.1016/j.ic.2014.12.012.

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

4 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Interscience, 2nd edition, 2006.

5 Pooya Davoodi, Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti. Encoding range minima and range top-2 queries. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130131–20130131, April 2014. doi:10.1098/rsta.2013.0131.

6 M. Effros, K. Visweswariah, S. R. Kulkarni, and S. Verdu. Universal lossless source coding with the burrows wheeler transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, May 2002. doi:10.1109/18.995542.

7 Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014. doi:10.1007/s00453-012-9664-0.

8 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Comput. Soc, 2000. doi:10.1109/sfcs.2000.892127.

9 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, January 2011. doi:10.1137/090779759.

10 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. (available on author's website: http://algo.inria.fr/flajolet/Publications/book.pdf).

11 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *STOC 1984*. ACM Press, 1984. doi:10.1145/800057.808675.

12 Moses Ganardi, Danny Hucke, Artur Jez, Markus Lohrey, and Eric Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86:136–158, June 2017. doi:10.1016/j.jcss.2016.12.007.

13 Moses Ganardi, Danny Hucke, Markus Lohrey, and Louisa Seelbach Benkner. Universal tree source coding using grammar-based compression. *IEEE Transactions on Information Theory*, 65(10):6399–6413, October 2019. doi:10.1109/tit.2019.2919829.

14 Moses Ganardi, Danny Hucke, Markus Lohrey, and Eric Noeth. Tree compression using string grammars. *Algorithmica*, 80(3):885–917, 2017. doi:10.1007/s00453-017-0279-3.

15 Michal Ganczorz. Entropy lower bounds for dictionary compression. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 11:1–11:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.11.

16 Michał Gańczorz. Using statistical encoding to achieve tree succinctness never seen before. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, LIPIcs, pages 22:1–22:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.22.

17 Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-based compression of unranked trees. *Theory of Computing Systems*, 64(1):141–176, 2020. doi:10.1007/s00224-019-09942-y.

18 Pawel Gawrychowski and Artur Jez. LZ77 Factorisation of Trees. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, volume 65 of *LIPIcs*, pages 35:1–35:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2016.35.

19 Paweł Gawrychowski and Patrick K. Nicholson. Optimal encodings for range top-k, selection, and min-max. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 593–604, 2015. doi:10.1007/978-3-662-47672-7_48.

20 Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, October 2006. doi:10.1145/1198513.1198516.

21 Mordecai Golin, John Iacono, Danny Krizanc, Rajeev Raman, Srinivasa Rao Satti, and Sunil Shende. Encoding 2d range maximum queries. *Theoretical Computer Science*, 609:316–327, January 2016. doi:10.1016/j.tcs.2015.10.012.

22 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation For Computer Science*. Addison-Wesley, 1994.

23 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2000. doi:10.1145/335305.335351.

24 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, January 2005. doi:10.1137/s0097539702402354.

25 Meng He, J. Ian Munro, and Srinivasa Satti Rao. Succinct ordinal trees based on tree covering. *ACM Transactions on Algorithms*, 8(4):1–32, 2012. doi:10.1145/2344422.2344432.

26 Hsien-Kuei Hwang and Ralph Neininger. Phase change of limit laws in the quicksort recurrence under varying toll functions. *SIAM Journal on Computing*, 31(6):1687–1722, January 2002. doi:10.1137/s009753970138390x.

27 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, A001263, 2021. URL: https://oeis.org/A001263.

28 G. Jacobson. Space-efficient static trees and graphs. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1989. doi:10.1109/sfcs.1989.63533.

29 Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences*, 78(2):619–631, March 2012. doi:10.1016/j.jcss.2011.09.002.

30 J.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, May 2000. `doi:10.1109/18.841160`.

31 John C. Kieffer, En-Hui Yang, and Wojciech Szpankowski. Structural complexity of random binary trees. In *2009 IEEE International Symposium on Information Theory*. IEEE, June 2009. `doi:10.1109/isit.2009.5205704`.

32 Markus Lohrey. Grammar-based tree compression. In *International Conference on Developments in Language Theory*, pages 46–57. Springer, 2015.

33 J. Ian Munro and Sebastian Wild. Entropy trees and range-minimum queries in optimal average-case space, 2019. `arXiv:1903.02533`.

34 Gonzalo Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016.

35 Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Computing Surveys*, 54(2):29:1–29:36, February 2021.

36 Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Computing Surveys*, 54(2):26:1–26:38, February 2021. `doi:10.1145/3432999`.

37 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2, April 2007. `doi:10.1145/1216370.1216372`.

38 Jürg Nievergelt and Edward M. Reingold. Binary search trees of bounded balance. *SIAM J. Comput.*, 2(1):33–43, 1973. `doi:10.1137/0202005`.

39 Nicola Prezza. Optimal Rank and Select Queries on Dictionary-Compressed Text. In Nadia Pisanti and Solon P. Pissis, editors, *Symposium on Combinatorial Pattern Matching (CPM)*, volume 128 of *LIPIcs*, pages 4:1–4:12, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CPM.2019.4`.

40 Louisa Seelbach Benkner and Markus Lohrey. Average Case Analysis of Leaf-Centric Binary Tree Sources. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *LIPIcs*, pages 16:1–16:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl. `doi:10.4230/LIPIcs.MFCS.2018.16`.

41 Dekel Tsur. Representation of ordered trees with a given degree distribution, 2018. `arXiv:1807.00371`.

42 Sebastian Wild. *Dual-Pivot Quicksort and Beyond: Analysis of Multiway Partitioning and Its Practical Potential*. Dissertation (Ph. D. thesis), University of Kaiserslautern, 2016. URL: `https://www.wild-inter.net/publications/wild-2016`.

43 Jie Zhang, En-Hui Yang, and John C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Transactions on Information Theory*, 60(3):1373–1386, March 2014. `doi:10.1109/tit.2013.2295392`.

A Range-Minimum Queries With Runs

In this appendix, we give the proofs of the results from Section 5.3.

A.1 Lower Bound

In this section, we proof Theorem 6.

We refer to a run of length one as a singleton run. The types of nodes in the Cartesian tree (whether or not their left and right children exist) directly reflect their role in runs: A binary node is a run head of a non-singleton run, a leaf is the last node of non-singleton run, a right-unary node (i.e., unary node with a right child) is a middle node of run and a left-unary node is a singleton run. The leftmost node, i.e., the node with smallest inorder rank, is the only exception to this rule: if the leftmost run is a singleton run, the leftmost node is a leaf; otherwise it is right-unary.

In any case, a Cartesian tree for an array with r runs that has b binary nodes and u_ℓ left-unary nodes thus satisfies $r = b + u_\ell + 1$: every binary node represents the non-singleton run that begins with it, every left-unary node represents the singleton run at that position, and the leftmost run is counted separately. (Note that we do not double count the latter because the leftmost node is by definition neither binary nor left-unary.) We therefore obtain a lower bound for the number of equivalence classes among length-n arrays with r runs under range-minimum queries by counting binary trees with a given number of nodes n and a given number of nodes of certain types.

That r is the sum of two quantities is inconvenient, hence we instead consider the following sequence of bijections (see Figure 2). First, we map Cartesian trees t of n nodes bijectively to balanced-parenthesis (BP) strings of n pairs of parentheses as follows: The empty tree corresponds to the empty string. For a nonempty tree, we recursively compute the BP strings of the (potentially empty) left resp. right subtrees of the root; let these be denoted by L and R. Then the BP string for the entire tree is obtained as `L(R)`. (This is a variation of the canonical BP representation.)

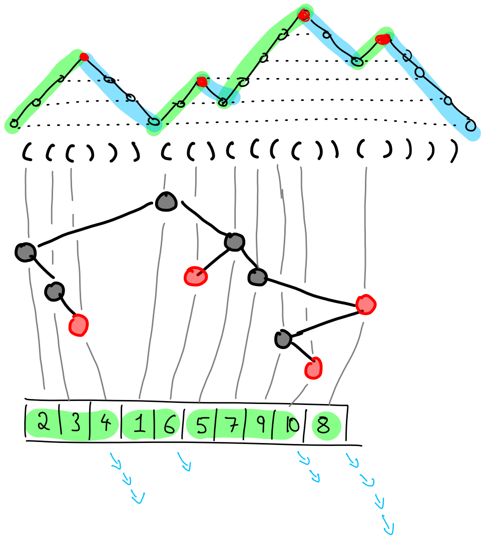

Figure 2 An example illustrating the bijections: The input array is $A = (2, 3, 4, 1, 6, 5, 7, 9, 10, 8)$, the min-oriented Cartesian tree is shown above with run ends highlighted in red. The BP string for the Cartesian tree is shown above the tree, with tree nodes connected to the corresponding opening parenthesis (note that nodes appear in inorder in the BP string). The maintain-valley (excess, Dyck path) representation of the BP string is on top; run ends correspond to peaks there.

It is easy to check that the resulting sequence is indeed the push/pop sequence of a max-stack [9, 19] where '`(`' means push and '`)`' means pop. We map this sequence to a lattice path by replacing '`(`' by step vector $(1, +1)$ and '`)`' by $(1, -1)$; the resulting lattice path is a mountain-valley diagram (Dyck paths).

The important property of the above bijections is that they *preserve runs*: A *run end* is an index where the next number is smaller (or nonexistent). In the Cartesian tree, these are the leaves *and* left-unary nodes, in the BP string, these are the occurrences of '`()`' and in the mountain-valley representations, these are the *peaks*. The latter is known to be counted by the *Narayana numbers*: There are

$$N_{n,r} \quad = \quad \frac{1}{n}\binom{n}{r}\binom{n}{r-1} \quad = \quad \frac{r}{n(n+1-r)}\binom{n}{r}^2 \tag{2}$$

mountain-valley diagrams of length $2n$ with exactly r peaks [27]. This concludes the proof of Theorem 6; the asymptotic approximation for $\lg N_{n,r}$ immediately follows from the above closed form.

A.2 Hypersuccinct RMQ with Runs

In this section, we prove Corollary 7. To this end, we show that using a hypersuccinct tree to represent the Cartesian tree of an array $A[1..n]$ with r increasing runs has a space usage that is bounded by $\lg N_{n,r} + o(n)$ bits. By Theorem 6, this space usage is optimal up to the $o(n)$ term.

As noted in Section A.1, the correspondence between runs and node types in the Cartesian tree can be made more specific by also specifying the number $s \in [r]$ of singleton runs: Singleton runs correspond to the left-unary nodes in the (min-oriented) Cartesian tree t, except possibly for a leftmost singleton run (which corresponds to a leaf). In either case, we will have $u_\ell = s \pm 1$ left-unary nodes and $\ell = r - s \pm 1$ leaves. That implies a number of binary nodes of $b = \ell - 1 \pm 1$; the remaining $u_r = n - b - \ell - u_\ell = n - 2r + s \pm 1$ nodes are right-unary nodes.

By Corollary 2, the hypersuccinct representation of the Cartesian tree t for $A[1..n]$ supports LCA-queries on t in $O(1)$ time and uses

$$|\mathsf{H}(t)| + o(n) \quad \le \quad H_0^{\text{type}}(t) \;+\; o(n)$$

bits of space. We show that $H_0^{\text{type}}(t) \le \lg N_{n,r} + o(n)$. Let

$$p \quad = \quad \left(\frac{b}{n}, \frac{u_l}{n}, \frac{u_r}{n}, \frac{\ell}{n}\right)$$

denote the empirical distribution of node types in t, and let $H(p)$ denote the entropy of this distribution. (For probability distribution $d = (d_1, d_2, \ldots, d_k)$, its entropy is defined by

$$H(d) \quad = \quad \sum_{i=1}^{k} d_i \lg\left(\frac{1}{d_i}\right),$$

as usual.)

By definition of the type-entropy H_0^{type}, we find $H_0^{\text{type}}(t) = nH(p)$. By our previous observations, p differs from

$$p' \quad = \quad \left(\frac{r-s}{n}, \frac{s}{n}, \frac{n-2r+s}{n}, \frac{r-s}{n}\right)$$

only by $\|p - p'\|_\infty \le \frac{2}{n}$. Using [42, Prop. 2.42], we thus find $H(p) \le H(p') + O(n^{-0.9})$ (this follows from Hölder-continuity of $x \mapsto x \ln x$). It thus remains to show that $n\,H(p') \le \lg N_{n,r} + o(n)$. By the grouping property of H, we have

$$n\,H(p') \quad = \quad n\left(H\left(\frac{r}{n}, \frac{n-r}{n}\right) + \frac{r}{n} H\left(\frac{s}{r}, \frac{r-s}{r}\right) + \frac{n-r}{n} H\left(\frac{r-s}{n-r}, \frac{(n-r)-(r-s)}{n-r}\right)\right).$$

In order to estimate the right-hand side, observe that it follows from [22, Eq. (5.22)] that $\sum_{s=0}^{r} \binom{r}{s}\binom{n-r}{r-s} = \binom{n}{r}$. Since all summands are positive, we have $\binom{r}{s}\binom{n-r}{r-s} \le \binom{n}{r}$ and hence

$$\lg\binom{r}{s} + \lg\binom{n-r}{r-s} \quad \le \quad \lg\binom{n}{r}, \qquad \text{for all } s \in [r]. \tag{3}$$

For a number $q \in [0,1]$, we set $h(q) = q\lg(1/q) + (1-q)\lg(1/(1-q))$. Using the standard inequality

$$\frac{2^{nh(q)}}{n+1} \;\le\; \binom{n}{qn} \;\le\; 2^{nh(q)}, \qquad nq \in [0..n], \tag{4}$$

we find

$$\begin{aligned} rh\Big(\frac{s}{r}\Big) \;+\; (n-r)h\Big(\frac{r-s}{n-r}\Big) \;&\underset{(4)}{\le}\; \lg\binom{r}{s} + \lg\binom{n-r}{r-s} \;+\; \lg(r+1) + \lg(n-r+1) \\ &\underset{(3)}{\le}\; \lg\binom{n}{r} \;+\; \lg(r+1) + \lg(n-r+1) \\ &\underset{(4)}{\le}\; nh\Big(\frac{r}{n}\Big) \;+\; \lg(r+1) + \lg(n-r+1). \end{aligned}$$

We thus have

$$\begin{aligned} nH(p') \;&=\; n\bigg(H\Big(\frac{r}{n}, \frac{n-r}{n}\Big) + \frac{r}{n}H\Big(\frac{s}{r}, \frac{r-s}{r}\Big) + \frac{n-r}{n}H\Big(\frac{r-s}{n-r}, \frac{(n-r)-(r-s)}{n-r}\Big)\bigg) \\ &=\; n\bigg(h\Big(\frac{r}{n}\Big) + \frac{r}{n}h\Big(\frac{s}{r}\Big) + \frac{n-r}{n}h\Big(\frac{r-s}{n-r}\Big)\bigg) \\ &\le\; 2nh\Big(\frac{r}{n}\Big) \;+\; O(\log n) \\ &\underset{(4)}{\le}\; 2\lg\binom{n}{r} \;+\; O(\log n) \\ &\underset{(2)}{\le}\; \lg N_{n,r} \;+\; O(\log n). \end{aligned}$$

So in total, we have shown that

$$|\mathsf{H}(t)| \;\le\; \lg N_{n,r} \;+\; o(n),$$

which implies Corollary 7.