Succinct Representations of Ordinal Trees

Rajeev Raman[1] and S. Srinivasa Rao[2⋆]

[1] Department of Computer Science, University of Leicester, UK.
r.raman@mcs.le.ac.uk
[2] School of Computer Science and Engineering, Seoul National University, S. Korea.
ssrao@cse.snu.ac.kr

Abstract. We survey *succinct* representations of ordinal, or rooted, ordered trees. Succinct representations use space that is close to the appropriate information-theoretic minimum, but support operations on the tree rapidly, usually in $O(1)$ time.

1 Introduction

An increasing number of applications such as information retrieval, XML processing, data mining etc. require large amounts of tree-structured data to be represented in main memory. Unfortunately, the memory consumption of classical ways of representing such data is often prohibitively large: for example, the standard in-memory representation of XML documents in a number of common programming languages such as C++, Java or Python requires memory an order of magnitude more than the size of the XML document on disk [40]. A similar situation arises when attempting to build a *suffix tree* data structure to index a collection of documents [31]. The large memory usage of standard tree representations severely affects the scalability of such applications.

This problem has led to the intensive study of *succinct*, or highly space-efficient, tree representations. This survey focusses on succinct *ordinal* trees, i.e. rooted, ordered trees. The standard way to represent an ordinal tree is to store pointers from each node to its first child and its next sibling; this represents the structure of the tree using $2n$ pointers, or $2n\lceil \lg n \rceil$ bits[3]. However, this is significantly more space than necessary. As there are $C_{n-1} = \frac{1}{n}\binom{2n-2}{n-1}$ ordinal trees on n nodes, there is an information-theoretic worst-case lower bound of $\lg C_{n-1} = 2n - O(\lg n)$ bits on any ordinal tree representation, and there is a trivial encoding of an n-node ordinal tree as an integer of $\lceil \lg C_{n-1} \rceil$ bits: the tree is encoded by its position in any systematic enumeration of all n-node ordinal trees. It has been known for several years (see e.g. [30] and references therein)

⋆ Work partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant number 2012-0008241).

[3] We use lg to denote the logarithm base 2.

that there are many—less brute-force—ways of encoding the structure of an n-node ordinal tree in $2n + O(1)$ bits. Thus, a succinct tree representation can potentially offer a $\Theta(\lg n)$ factor space savings in the worst case.

At first sight it appears that it would be difficult to perform operations quickly on such a compact representation: in particular, converting a compact representation to a standard one to perform operations defeats the purpose of considering the compact representation. However, standard representations also have limitations: for instance, in the basic first-child next-sibling representation described above, navigating to the parent of a node takes $O(n)$ time, unless a parent pointer is added, increasing the space usage to $3n\lceil \lg n \rceil$ bits from $2n\lceil \lg n \rceil$ bits. Determining the size of the subtree rooted at a node in $O(1)$ time would require storing an additional $\Theta(n \lg n)$ bits, and adding more complex functionality such as level-ancestor [6] or lowest common ancestor [1] would add a further $\Theta(n \lg n)$ bits each. In short, while standard representations can support a number of navigational and other queries in 'linear' space ($\Theta(n)$ words or $\Theta(n \lg n)$ bits), as the set of operations increases, so does the constant within the $\Theta(n)$. To mitigate this space blow-up, real-world software libraries can sometimes attempt to minimize the constant factor in the space usage by omitting some pointers, with disastrous results [14, p144]. On the other hand, as we will see, a number of succinct representations have been developed that use only $2n + o(n)$ bits of space, and support a wide variety of navigational and other operations in $O(1)$ time on the word RAM model. A key advantage of succinct representations is that as their functionality increases, usually only the constant in the $o(n)$ increases. Thus, adding new functionality has very little additional cost.

Furthermore, as a number of implementations and empirical evaluations have demonstrated (see Section 5), the practical performance of succinct ordinal tree representations is excellent, both in terms of memory usage and speed. They have also only been integrated successfully into a range of applications [13, 3, 45]. Furthermore, they are currently available in well-documented open-source libraries such as SDSL [22] and succinct [37].

The paper is organised as follows. After some preliminaries, we discuss the main succinct representations of ordinal trees. We then cover specialized topics including dynamization and the redundancy of tree representations, and conclude with implementations, empirical evaluations and practical issues.

2 Preliminaries

2.1 Terminology

Succinct ordinal trees can be *dynamic*, i.e. allow changes to the tree, or *static*. Our focus will be more on static trees, though we address dynamization issues in Section4.1. For static trees, we assume that the input tree is pre-processed to obtain a succinct representation: we do not normally focus on the time and space requirements for the pre-processing (which is, however, an important issue in practice). The space used by the succinct representation of an n-node tree can

be expressed as $I(n) + R(n)$ bits where $I(n) = \lceil \lg C_{n-1} \rceil = 2n - O(\lg n)$ is the information-theoretic lower bound and $R(n) \geq 0$ is called the *redundancy*. The redundancy is a term of great practical significance, and studying the trade-off between redundancy and query time is of fundamental importance.

Succinct tree representations are of two kinds: *systematic* (also known as *succinct indices*) and *non-systematic*. In systematic tree representations, we separate the storage for the encoding of the tree (which usually takes $2n + O(1)$ bits) from the *succinct index*, an auxiliary data structure created during preprocessing to help answer queries rapidly. A query is answered by reading $O(1)$ words (each of $O(\lg n)$ consecutive bits) from the query and the succinct index. In a systematic representation, the redundancy is essentially the size of the succinct index. Separating the 'structure of the tree' from auxiliary data structures not only makes it easier to show lower bounds on the redundancy needed to achieve a particular query time [19, 24], it also has several algorithmic advantages [4]. Non-systematic representations do not have this conceptual separation, but it is known that the redundancy needed to achieve a particular query time is lower for non-systematic representations than for systematic representations.

2.2 Features of succinct tree representations

Succinct tree representations require some care in their use. Firstly, the numbering of nodes is based upon the position of the representation of the nodes in the encoding of the tree. This is particularly problematic in the dynamic case, since a single update may change the numbers of every node in the tree. Furthermore, certain operations can be implemented efficiently in one tree encoding, but are hard or impossible to implement in another. Since different encodings number nodes differently, it may not be possible to come up with a succinct representation that supports the union of the operations of two encodings. Finally, this way of numbering nodes sometimes means that the 'natural' numbering of nodes is a sequence of non-consecutive integers from the range $\{1, \ldots, 2n + O(1)\}$.

2.3 FIDs

Given a subset S from a universe U, we define a *fully indexable dictionary* (FID, from now on) on S to be any data structure that supports the following operations on S in constant time, for any $x \in U$, and $1 \leq i \leq |U|$:

- $rank(x)$: return the number of elements in S that are less than x,
- $select(i, S)$: return the i-th smallest element in S, and
- $select(i, \bar{S})$: return the i-th smallest element in $U \setminus S$.

Lemma 1. *[41] Given a subset of size n from the universe $[m]$, there is a FID that uses $\lg \binom{m}{n} + O(m \lg \lg m / \lg m)$ bits.*

Given a bitvector B, we define its FID to be the FID for the set S where B is the characteristic vector of set S.

3 Ordinal tree representations

In this section we describe the main succinct ordinal tree representations.

3.1 Operations on ordinal trees

We first introduce a set of useful operations that are supported by various ordinal tree representations. Given an ordinal tree, we consider the following operations:

- `parent`(x): returns the parent of node x;
- `child`(x, i): returns the i-th child of node x, for $i \geq 1$;
- `child_rank`(x): returns the number of left siblings of node x plus 1;
- `depth`(x): returns the depth of node x;
- `level_ancestor`(x, i): returns the ancestor of node x that is i levels above x, for $i \geq 0$ (if x is at depth d, it returns the ancestor of x at depth $d - i$);
- `desc`(x): returns the number of descendants of node x;
- `degree`(x): returns the degree of node x;
- `height`(x): returns the height of the subtree rooted at node x;
- `LCA`(x, y): returns the lowest common ancestor of the nodes x and y;
- `left_leaf`(x) (`right_leaf`(x)): returns the leftmost (rightmost) leaf of the subtree rooted at node x;
- `leaf_rank`(x): returns the number of leaves up to node x in preorder;
- `leaf_select`(i): returns the i-th leaf among all the leaves from left to right;
- `leaf_size`(x): returns the number of leaves in the subtree rooted at node x;
- `rank`$_{\text{PRE/POST/DFUDS/LEVEL}}(x)$: returns the position of node x in the preorder, postorder, DFUDS order or level order traversal of the tree;
- `select`$_{\text{PRE/POST/DFUDS/LEVEL}}(j)$: returns the j-th node in the preorder, postorder or DFUDS order or level order traversal of the tree;
- `level_left`(i) (`level_right`(i)): returns the first (last) node visited in a preorder traversal among all the nodes whose depths are i;
- `level_succ`(x) (`level_pred`(x)): the *level successor* (*predecessor)* of node x, i.e. the node visited immediately after (before) node x in a preorder traversal among all the nodes that are at the same level as node x.

3.2 Jacobson's representation

Jacobson [26] proposed the first succinct tree representation for ordinal trees. His representation is based on the *level order unary degree sequence* (LOUDS) of a tree, which visits the nodes in a level-order traversal of the tree and encodes their degrees in unary. (In a level-order traversal of a tree, the root is visited first, followed by all the children of the root, from left to right, followed by all the nodes at each successive level.) See Figure 1 for an example. This representation uses $2n+O(1)$ bits to encode an ordinal tree on n nodes, and an additional $o(n)$ bits to support `rank` and `select` operations on the encoding. Jacobson considered these operations in the bit probe model, and showed how to support them in $O(\lg n)$

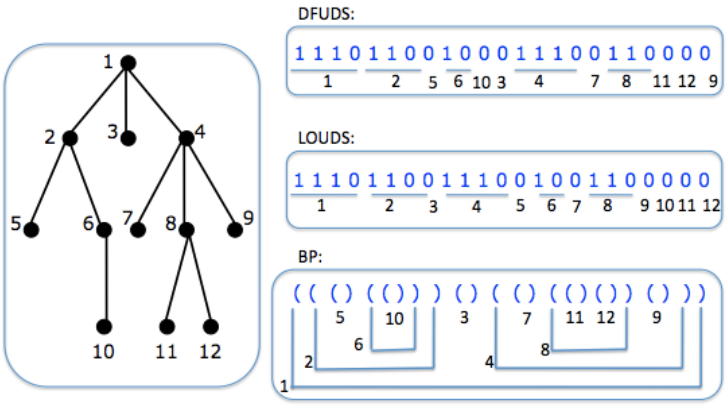


Fig. 1. An example of the DFUDS, LOUDS and BP sequences of a given ordinal tree.

probes. Clark and Munro [10] further showed how to support the `rank` and `select` operations in $O(1)$ time under the word RAM model with $\Theta(\lg n)$ word size. Thus the LOUDSrepresentation supports `parent`, `child` and `child_rank` operations in $O(1)$ time. In addition to these operations, it is straightforward to support $\texttt{rank}_{\text{LEVEL}}$, $\texttt{select}_{\text{LEVEL}}$, `level_succ` and `level_pred` operations in constant time.

3.3 Balanced Parenthesis representation

Munro and Raman [33] proposed another type of succinct representation of trees based on the isomorphism between *balanced parenthesis sequences* (BP) and ordinal trees. The BP sequence of a given tree is obtained by performing a depth-first traversal, and writing an opening parenthesis each time a node is visited, and a closing parenthesis immediately after all its descendants are visited. This gives a $2n$-bit encoding of an n-node tree as a sequence of balanced parentheses, and a node can be identified e.g. by the position of the opening parenthesis of the pair corresponding to it. See Figure 1 for an example. Extending the ideas from Jacobson [26], Munro and Raman showed how to support a few basic operations in constant time on a balanced parenthesis sequence of length n using an auxiliary structure of size $o(n)$ bits. By showing how to translate the operations on the tree to the basic operation on the parenthesis sequence representing it, Munro and Raman [33] presented a succinct representation of an ordinal tree on n nodes in $2n + o(n)$ bits based on BP, which supports `parent`, `desc`, `depth`, $\texttt{rank}_{\text{PRE/POST}}$ and $\texttt{select}_{\text{PRE/POST}}$ in constant time, and $\texttt{child}(x, i)$ in $O(i)$ time. Munro et al. [34] provided constant-time support for `leaf_rank`, `leaf_select`, `left_leaf`, `right_leaf` and `leaf_size` on the BP representation using $o(n)$ additional bits, which were used to design space-efficient suffix trees. Chiang et al. [8] showed how to support `degree` in constant time. The support for `level_ancestor`, `level_succ` and `level_pred` in constant time was further

provided by Munro and Rao [36], which has applications in the succinct representations of functions. Lu and Yeh [32] showed how to support `child`, `child_rank`, `height` and `LCA` operations in constant time.

Using a different approach to support the operations, Sadakane and Navarro [43] augmented the BP sequence with $o(n)$-bit auxiliary structure to obtain a "fully functional" representation as described in Section 3.7.

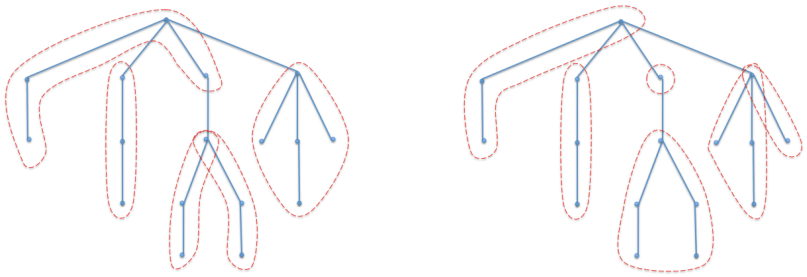
3.4 Depth-First Unary Degree Sequence representation

Benoit et al. [5] observed that by visiting the nodes in depth-first order (i.e., preorder) and encoding their degrees in unary (instead of visiting them in level order to produce the LOUDS encoding), one can support other useful operations such as `desc`. The resulting encoding of the tree is called the *depth first unary degree sequence* (DFUDS). More specifically, the DFUDS sequence represents a node of degree d by d opening parentheses followed by a closing parenthesis. All the nodes are listed in preorder (an extra opening parenthesis is added to the beginning of the sequence). See Figure 1 for an example. The DFUDS number of a node is defined to be the rank of the opening parenthesis in its parent's description that corresponds to this node. Benoit et al. [5] presented a succinct tree representation based on DFUDS that occupies $2n + o(n)$ bits and supports `child`, `parent`, `degree` and `desc` in constant time. In their representation, each node is referred to by the position of the first parenthesis in the representation of the node. Jansson et al. [27] extended this representation using $o(n)$ additional bits to provide constant-time support for `child_rank`, `depth`, `level_ancestor`, `LCA`, `left_leaf`, `right_leaf`, `leaf_rank`, `leaf_select`, `leaf_size`, $\texttt{rank}_{\text{PRE}}$ and $\texttt{select}_{\text{PRE}}$. Barbay et al. [4] further showed how to support $\texttt{rank}_{\text{DFUDS}}$ and $\texttt{select}_{\text{DFUDS}}$. The operations $\texttt{rank}_{\text{PRE}}$, $\texttt{select}_{\text{PRE}}$, $\texttt{rank}_{\text{DFUDS}}$ and $\texttt{select}_{\text{DFUDS}}$ support the constant-time conversions between the preorder number and DFUDS number of the same node, which is used to support various queries on labeled trees by Barbay et al. [4].

3.5 Representations based on tree covering

Another approach to represent ordinal trees is based on a *tree covering* algorithm (TC). This approach, first proposed by Geary et al. [20], is based on an algorithm to cover an ordinal tree with a set of *mini-trees*, each of which is further covered by a set of *micro-trees*.

Geary et al. [20] proposed an algorithm to cover a given ordinal tree on n nodes into $O(n/M)$ mini-trees, each of size $O(M)$, for a given parameter M. This decomposition guarantees that any two mini-trees computed by this algorithm are either disjoint, or only joined at their common root, and in addition, every mini-tree, except possibly the one containing the root, has size $\Theta(M)$. See Figure 3.5(a) for an example. The tree structure representing how these $O(n/M)$ mini-trees are connected is then stored using $O(n \lg n/M)$ bits. Similarly, each mini-tree is further decomposed into $O(M/M')$ micro-trees, each of size $O(M')$,



(a) (b)

Fig. 2. An example of covering an ordinal tree using the algorithm of (a) Geary et al. [20], and (b) Farzan and Munro [15], with parameter $M = 3$.

for a parameter M', using the same algorithm, and the tree structures of all the mini-trees (representing their micro-tree decomposition) is stored using a total of $O(n(\lg M)/M')$ bits. Each of the micro-trees is represented as a pointer to a table of size at most $2^{2M'}$. This table stores the representations of all possible micro-trees of size M'.

By choosing $M = \lceil \lg^4 n \rceil$ and $M' = \lceil (\lg n)/24 \rceil$, Geary et al. [20] obtain a representation that takes $2n + o(n)$ bits. They further show how to support various operations (namely, `child`, `child_rank`, `depth`, `level_ancestor`, `desc`, `degree`, $\texttt{rank}_{\text{PRE/POST}}$ and $\texttt{select}_{\text{PRE/POST}}$) on the ordinal tree in constant time by storing various auxiliary structures using $o(n)$ bits. He et al. [25] extended the representation by supporting several additional operations, namely `LCA`, `height`, `left_leaf`, `right_leaf`, `leaf_rank`, `leaf_select`, `leaf_size`, $\texttt{rank}_{\text{DFUDS}}$, $\texttt{select}_{\text{DFUDS}}$, `level_left`, `level_right`, `level_succ` and `level_pred`.

Farzan and Munro [15] modified the tree covering algorithm so that each mini-tree has at most one node, other than the root of the mini tree, that is connected to the root of another mini-tree. See Figure 3.5(b) for an example. This simplifies the representation of the tree, as well as the auxiliary structures needed to support various operations on the tree. They call this approach as the *uniform approach*, and justify the name by demonstrating that this approach can be applied to obtain succinct representations for various other families of trees, including cardinal trees.

3.6 "Universal" representation

The succinct tree representations described so far are systematic encodings: they encode the *structure* of the tree as a bit-string of length $2n + o(n)$ bits, together with an index of $o(n)$ bits, where the index depends upon the choice of structure bit-string and the operations to be supported. Operations are supported in $O(1)$ time by reading $O(1)$ words from the structure bit-string and/or the index.

Since the index depends on the choice of structure bit-string, which also determines the node numbering, this approach leads to certain difficulties [17]. For example, certain operations can be implemented efficiently using one encoding, but are hard or impossible to implement in another. As noted in the introduction, it is not possible in general to create a representation that supports the union of the sets of operations of two representations without losing optimality. Even if we represent the given tree as two separate copies, each using the respective structure bit-strings and index data structures (thus losing optimality) we would still face the problem that a series of linked operations using both representations would require us to be able to map nodes of one numbering to the other, which is not always easy to achieve.

Farzan et al. [17] proposed an optimal-space succinct encoding for ordinal trees that can return $b = O(w)$ consecutive bits from the structure bit-strings of other (BP, DFUDS or TC) encodings, where w is the word-size, in $O(1)$ time. Since we can emulate access to the structure bit-strings of other encodings, by adding the appropriate index of $o(n)$ bits, one can directly support any operations supported by those encodings, with only a constant factor slowdown and negligible additional space cost. This representation is called the *universal representation* (UNIVERSAL).

3.7 "Fully-functional" representation

Sadakane and Navarro [43] proposed an ordinal tree representation that is based on storing the structure bit-string of BP, and constructing auxiliary structures to support the operations. But it differs from the BP representation by significantly simplifying the auxiliary structures, and also improving the lower-order term in space. In particular, they base navigation on the key primitive of *excess search*. The *excess* of a position in a parentheses sequence is the difference between the numbers of open and closed parentheses from the start of the sequence to the given position (the depth of a node equals the excess of its open parenthesis in the BP sequence). An excess search operation such as `fwd_excess`(i, d) returns the closest position $j \geq i$ whose where the excess equals the excess at i plus d. They propose a simple and flexible data structure, called the *range min-max tree* for supporting excess search, and then reduce a wide range of operations on the ordinal trees to simple combinations excess search and other primitive operations. The resulting representation, which they call as a *fully-functional* (FF) representation, (i) is conceptually simpler than most of the earlier representations, (ii) has smaller redundancy than the earlier ones, (iii) can be easily implemented, and (iv) can be efficiently dynamized.

Table 1 shows the comparison between the functionalities of various succinct tree representations that we described in this section.

Operations	LOUDS	BP	DFUDS	TC/ UNIVERSAL	FF
`parent`, `child`, `child_rank`	✓	✓	✓	✓	✓
`depth`, `level_ancestor`		✓	✓	✓	✓
`degree`	✓	✓	✓	✓	✓
`height`		✓		✓	✓
`LCA`		✓	✓	✓	✓
`desc`, `leaf_size`		✓	✓	✓	✓
`left_leaf`, `right_leaf`		✓	✓	✓	✓
`leaf_rank`, `leaf_select`		✓	✓	✓	✓
$\texttt{rank}_{\text{PRE}}$, $\texttt{select}_{\text{PRE}}$		✓	✓	✓	✓
$\texttt{rank}_{\text{POST}}$, $\texttt{select}_{\text{POST}}$		✓		✓	✓
$\texttt{rank}_{\text{DFUDS}}$, $\texttt{select}_{\text{DFUDS}}$			✓	✓	
$\texttt{rank}_{\text{LEVEL}}$, $\texttt{select}_{\text{LEVEL}}$	✓				
`level_left`, `level_right`				✓	✓
`level_succ`, `level_pred`	✓	✓		✓	✓

Table 1. Navigational operations supported in $O(1)$ time on various succinct ordinal tree representations. All these representations use $2n + o(n)$ bits.

4 Additional Topics

4.1 Dynamization

Following upon dynamization of binary trees [35, 42], dynamization of succinct ordinal trees was considered by Sadakane and Navarro [43] and Farzan and Munro [16]. In the former, the update operations are taken to be edits on the BP sequence representing the tree, e.g. adding or deleting a parenthesis pair, while Farzan and Munro consider the slightly less general operations of adding a leaf or a new node breaking an existing edge. In both cases, navigation and other operations should continue to be supported once the update is complete.

A fundamental difference is in the way the operations are specified in the two approaches. Sadakane and Navarro follow the static API of [33], and perform many navigation operations via excess search. A key feature is that operations on a node are specified by the position of that node in the BP bit-string. As noted by Joannou and Raman [28], when using this approach, any navigation operation has several steps that are known require $\Omega(\lg n / \lg \lg n)$ time (unless updates are allowed to take more than poly-log time). However, the resulting dynamization does support the full range of navigational operations supported by the FF representation, together with the updates, in $O(\lg n / \lg \lg n)$ time.

An alternative is the *finger* model [16, 42, 35], where updates are done using a finger, which is effectively a "pointer" to a node in the tree. A finger may be moved using some navigational operations, and crucially, updates can only happen in the vicinity of a finger. In the finger model, the non-constant lower bounds above do not apply and both updates and navigation operations can in fact be performed in $O(1)$ time [16] (the time for updates is amortized).

However, the set of operations supported by Farzan and Munro is smaller than that of Sadakane and Navarro: in addition to the basic navigation operations, only `child`, `child_rank` and `desc` are supported.

4.2 Compressibility

When considering the information-theoretic lower bound of $2n - O(\lg n)$ bits, it must be noted that this is a *worst-case* bound. The same information-theoretic lower bound applies if the tree is a random tree or highly regular (e.g. complete k-ary tree). On the other hand, trees we find in practice are usually compressible. Although there has been work on compressing labelled trees [18], the work on representing compressible ordinal trees is far less mature. Jansson et al. [27] provided a definition of tree compressibility based on degree sequences (and gave a combinatorial justification thereof). They showed that it is possible to compress a given ordinal tree in the minimum possible space (according to their measure of compressibility), plus lower-order terms, and still support operations in $O(1)$ time. Bille et al. [7] considered ordinal trees that are compressed by sharing subtrees, which is a form of grammar-based compression. They showed that by generalizing excess search to such grammar-compressed BP strings, it is possible to support many of the operations of the representation of [43] in $O(\lg n)$ time. The space used is $O(m \lg n)$ bits, where m is the size of the grammar that generates the given tree.

4.3 Redundancy

The redundancy of a succinct representation is a quantity of both practical and fundamental importance, and papers have increasingly focussed on obtaining the best possible redundancy while still obtaining the best running times for operations (typically $O(1)$ time). As noted in the introduction, the redundancy is viewed slightly differently for systematic and non-systematic representations. For systematic representations, it is known that for the BP encoding, an index of size $\Theta(n \lg \lg n / \lg n)$ bits is needed to perform a full set of navigational operations [24]. However, using a non-systematic encoding, it is possible to obtain a redundancy of $O(n/(\lg n)^c)$ for any constant $c > 0$ [39, 43].

We close on a lighter note. We note that even without the redundancy caused by requiring $O(1)$-time operations, the basic representations such as BP, DFUDS etc. are at least $2n - O(1)$ bits long, while the lower bound is $2n - O(\lg n)$ bits. We consider how to eliminate this $O(\lg n)$-bit additive overhead. As noted in the introduction, a trivial encoding—encoding a given tree by its position in an enumeration—achieves the lower bound, albeit at the cost of making operations quite difficult. However, Golin et al. [23] note that binary trees can be encoded as follows: write down the size of the left subtree of the root (which is a number from $\{0, \ldots, n-1\}$), and then recurse on the left and right subtrees; finally encode this sequence as a mixed-radix number. This approach gives an optimal-space (zero redundancy) encoding of binary trees such that operations can be performed in

polynomial time; via the standard equivalence between binary and ordinal trees, we also obtain an optimal succinct ordinal tree representation where navigation can be performed in polynomial time.

5 Implementations and Experimental Evaluation

Succinct representations of trees have been applied in a number of practical contexts. In the context of dictionary indexing, Clark [11] implemented succinct binary trees, and an implementation of a compact trie, the *Bonsai* tree, was used for text prediction and compression [12].

Ordinal tree implementations were apparently first studied in detail by Geary et al. [21] who implemented a simplified version of the BP representation of Munro and Raman (although [33] mentions a previous BP implementation by Chupa [9], details appear not to be in the public domain). Delpratt et al. [14] considered engineering the LOUDS representation and suggested a useful practical trick called *double numbering*. This addresses the issue that while the 'natural' numbering of notes in the succinct representation of an n-node tree is often as non-consecutive integers from $\{1, \ldots, 2n + O(1)\}$, for a variety of applications, a 'compact' numbering from $\{1, \ldots, n\}$ is desirable, particularly in order to associate information with the nodes. In theory, this can often be achieved in $O(1)$ time by means of `select`, to convert a user-provided node number (in a compact numbering) to the representation's natural numbering, followed by the appropriate tree operation, finally followed by a `rank` operation to convert the answer back to a node number in a compact numbering. However, this is a significant overhead in practice, and Delpratt et al. note that it is often better to number a node as a pair $\langle x, y \rangle$, where x and y number the node according to the compact and natural numberings, as updating the *pair* during operations is often trivial.

Subsequently, Arroyuelo et al. [2] implemented a simplified $O(\lg n)$-time excess search algorithm based on [43] for navigating in the BP representation; the resulting implementation appears to have similar speed performance, greater functionality and better space usage to that of Geary et al. [21]. Joannou and Raman [28] observe that the reason that the simple $O(\lg n)$-time excess search of Arroyuelo et al. performs comparably to the $O(1)$-time implementation of Geary et al. for navigation is that navigation in ordinal trees (i.e. a sequence of steps moving from a node to an adjacent one) induces a kind of locality of access in the BP sequence; consequently, the $O(\lg n)$-time worst-case cost may not be paid frequently. They advocate the use of *splay* trees [44] as a data structure for excess search to exploit this locality, and show good empirical performance. However, they do not perform a theoretical worst-case analysis. They also present the first empirical study of dynamic ordinal trees. Grossi and Ottaviano [38] present a highly engineered version of excess search.

In general, the experimental results show that succinct ordinal tree representations are competitive with naive representations, even when both the succinct and naive representations are too large to fit in cache, and small enough to fit

in main memory. Succinct representations are usually much faster in case the succinct representation fits into faster memory (cache/main memory) while the naive representation does not. Admittedly, an $O(1)$-time operation on a succinct ordinal tree can be more complex (in terms of the number of instructions performed) than the same operation in a classical ordinal tree representation, where an operation may be as simple as just following a pointer. However, when storing relatively large data, following a pointer can incur a cache miss or a TLB miss (which is even worse [29]). On the other hand, the best practical implementations of a succinct data structure will spend many instructions sequentially accessing memory locations, which is usually very fast in practice. Furthermore, since more information is packed into the faster levels of the memory hierarchy, succinct data structures show better locality as well.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: On finding lowest common ancestors in trees. In: Aho, A.V., Borodin, A., Constable, R.L., Floyd, R.W., Harrison, M.A., Karp, R.M., Strong, H.R. (eds.) STOC. pp. 253–265. ACM (1973)
2. Arroyuelo, D., Cánovas, R., Navarro, G., Sadakane, K.: Succinct trees in practice. In: Blelloch, G.E., Halperin, D. (eds.) ALENEX. pp. 84–97. SIAM (2010)
3. Arroyuelo, D., Claude, F., Maneth, S., Mäkinen, V., Navarro, G., Nguyen, K., Sirén, J., Välimäki, N.: Fast in-memory XPath search using compressed indexes. In: Li, F., Moro, M.M., Ghandeharizadeh, S., Haritsa, J.R., Weikum, G., Carey, M.J., Casati, F., Chang, E.Y., Manolescu, I., Mehrotra, S., Dayal, U., Tsotras, V.J. (eds.) ICDE. pp. 417–428. IEEE (2010)
4. Barbay, J., He, M., Munro, J.I., Satti, S.R.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms 7(4), 52 (2011)
5. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica 43(4), 275–292 (2005)
6. Berkman, O., Vishkin, U.: Finding level-ancestors in trees. J. Comput. Syst. Sci. 48(2), 214–230 (1994)
7. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Satti, S.R., Weimann, O.: Random access to grammar-compressed strings. In: Randall, D. (ed.) SODA. pp. 373–389. SIAM (2011)
8. Chiang, Y.T., Lin, C.C., Lu, H.I.: Orderly spanning trees with applications. SIAM J. Comput. 34(4), 924–945 (2005)
9. Chupa, K.: Efficient Representation of Binary Search Trees. Master's thesis, University of Waterloo (1997), mMath Thesis
10. Clark, D.R., Munro, J.I.: Efficient suffix trees on secondary storage (extended abstract). In: ACM-SIAM SODA. pp. 383–391 (1996)
11. Clark, D.R.: Compact pat trees. Ph.D. thesis, University of Waterloo, Waterloo, Ontario., Canada. (1998)
12. Darragh, J.J., Cleary, J.G., Witten, I.H.: Bonsai: a compact representation of trees. Softw., Pract. Exper. 23(3), 277–291 (1993)
13. Delpratt, O., Raman, R., Rahman, N.: Engineering succinct DOM. In: EDBT. ACM Intl. Conference Proceeding Series, vol. 261, pp. 49–60. ACM (2008)
14. Delpratt, O., Rahman, N., Raman, R.: Engineering the LOUDS succinct tree representation. In: Àlvarez, C., Serna, M.J. (eds.) WEA. Lecture Notes in Computer Science, vol. 4007, pp. 134–145. Springer (2006)

15. Farzan, A., Munro, J.I.: A uniform approach towards succinct representation of trees. In: Gudmundsson, J. (ed.) SWAT. Lecture Notes in Computer Science, vol. 5124, pp. 173–184. Springer (2008)
16. Farzan, A., Munro, J.I.: Succinct representation of dynamic trees. Theor. Comput. Sci. 412(24), 2668–2678 (2011)
17. Farzan, A., Raman, R., Rao, S.S.: Universal succinct representations of trees? In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S.E., Thomas, W. (eds.) ICALP (1). Lecture Notes in Computer Science, vol. 5555, pp. 451–462. Springer (2009)
18. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. J. ACM 57(1) (2009)
19. Gál, A., Miltersen, P.B.: The cell probe complexity of succinct data structures. Theor. Comput. Sci. 379(3), 405–417 (2007)
20. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms 2(4), 510–534 (2006)
21. Geary, R.F., Rahman, N., Raman, R., Raman, V.: A simple optimal representation for balanced parentheses. Theor. Comput. Sci. 368(3), 231–246 (2006)
22. Gog, S.: Succinct data structure library (SDSL) (2013), https://github.com/simongog/sdsl, https://github.com/simongog/sdsl
23. Golin, M.J., Iacono, J., Krizanc, D., Raman, R., Rao, S.S.: Encoding 2-d range maximum queries. CoRR abs/1109.2885 (2011)
24. Golynski, A., Grossi, R., Gupta, A., Raman, R., Rao, S.S.: On the size of succinct indices. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA. Lecture Notes in Computer Science, vol. 4698, pp. 371–382. Springer (2007)
25. He, M., Munro, J.I., Rao, S.S.: Succinct ordinal trees based on tree covering. ACM Transactions on Algorithms 8(4), 42 (2012)
26. Jacobson, G.J.: Space-efficient static trees and graphs. IEEE Symposium on Foundations of Computer Science, 1989 pp. 549–554 (1989)
27. Jansson, J., Sadakane, K., Sung, W.K.: Ultra-succinct representation of ordered trees with applications. Journal of Computer and System Sciences 78(2), 619 – 631 (2012), http://www.sciencedirect.com/science/article/pii/S0022000011001012
28. Joannou, S., Raman, R.: Dynamizing succinct tree representations. In: Klasing, R. (ed.) SEA. Lecture Notes in Computer Science, vol. 7276, pp. 224–235. Springer (2012)
29. Jurkiewicz, T., Mehlhorn, K.: The cost of address translation. In: Sanders, P., Zeh, N. (eds.) ALENEX. pp. 148–162. SIAM (2010)
30. Katajainen, J., Mäkinen, E.: Tree compression and optimization with applications. Int. J. Found. Comput. Sci. 1(4), 425–448 (1990)
31. Kurtz, S.: Reducing the space requirement of suffix trees. Softw., Pract. Exper. 29(13), 1149–1171 (1999)
32. Lu, H., Yeh, C.: Balanced parentheses strike back. ACM Trans. Algorithms 4(3), 1–13 (2008)
33. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. 31(3), 762–776 (2001)
34. Munro, J.I., Raman, V., Rao, S.S.: Space efficient suffix trees. J. Algorithms 39(2), 205–222 (2001)
35. Munro, J.I., Raman, V., Storm, A.J.: Representing dynamic binary trees succinctly. In: Kosaraju, S.R. (ed.) SODA. pp. 529–536. ACM/SIAM (2001)
36. Munro, J.I., Rao, S.S.: Succinct representations of functions. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP. Lecture Notes in Computer Science, vol. 3142, pp. 1006–1015. Springer (2004)

37. Ottaviano, G.: The succinct library (2013), https://github.com/ot/succinct, https://github.com/ot/succinct
38. Ottaviano, G., Grossi, R.: Design of practical succinct data structures for large data collections, invited talk in *SEA'13*
39. Patrascu, M.: Succincter. In: FOCS. pp. 305–313. IEEE Computer Society (2008)
40. Poyias, A.: XXML: Handling extra-large XML documents (February 2013), https://lra.le.ac.uk/bitstream/2381/27744/1/SiXDOMWhitepaper.pdf, siXML technology whitepaper
41. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3(4), 43 (2007 Preliminary version in SODA 2002)
42. Raman, R., Rao, S.S.: Succinct dynamic dictionaries and trees. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP. Lecture Notes in Computer Science, vol. 2719, pp. 357–368. Springer (2003)
43. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: Charikar, M. (ed.) SODA. pp. 134–149. SIAM (2010)
44. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM 32(3), 652–686 (1985)
45. Tabei, Y.: Succinct multibit tree: Compact representation of multibit trees by using succinct data structures in chemical fingerprint searches. In: Raphael, B.J., Tang, J. (eds.) WABI. Lecture Notes in Computer Science, vol. 7534, pp. 201–213. Springer (2012)