A Uniform Approach Towards Succinct Representation of Trees

Arash Farzan and J. Ian Munro
{afarzan,imunro}@cs.uwaterloo.ca

School of Computer Science,
University of Waterloo,
Waterloo, Ontario, Canada.

Technical Report CS-2008-11

Abstract. We propose a uniform approach for succinct representation of various families of trees. The method is based on two recursive decomposition of trees into subtrees. Recursive decomposition of a structure into substructures is a common technique in succinct data structures and has been shown fruitful in succinct representation of ordinal trees [7, 10] and dynamic binary trees [16, 21]. We take an approach that simplifies the existing representation of ordinal trees while allowing the full set of navigational operations. The approach applied to cardinal (*i.e.* k-ary) trees yields a space-optimal succinct representation allowing cardinal-type operations (*e.g.* determining child labeled i) as well as the full set of ordinal-type operations (*e.g.* reporting the number of siblings to the left of a node). Existing space-optimal succinct representations had not been able to support both types of operations [2, 19].
We demonstrate how the approach can be applied to obtain a space-optimal succinct representation for the family of free trees where the order of children is insignificant. Furthermore, we show that our approach can be used to obtain entropy-based succinct representations. We show that our approach matches the degree-distribution entropy suggested by Jansson *et al.* [13]. We discuss that our approach can be made adaptive to various other entropy measures.

1 Introduction

With the ever increasing size of data sets, an important aspect in handling information is their storage requirement. A succinct representation of a combinatorial object is an encoding which supports a reasonable set of operations on the object in constant time and has a storage requirement matching the information theoretic lower bound, to within lower order terms. Succinct data structures perform under the uniform-time word RAM-model with $\Theta(\lg n)^1$ word size.

Trees are fundamental data structures in computer science and as a result a great deal of research has been done on their succinct representation. Succinct representation of two major families of trees have been well studied: *ordinal* and *cardinal*. In ordinal trees the order of children of nodes is significant and preserved. However, in cardinal trees (also known as k-ary trees), each node has k-slots for edges to children which can be independently occupied or not. Binary trees are a subclass of cardinal trees for value $k = 2$. Here we also study the succinct encoding of another family of trees: *free trees*, in which the order of children of a node is not significant.

Certain subfamilies of these major families of trees have been studied in the context of succinct representations. Binary trees ($k = 2$) or DNA trees ($k = 4$) form two of the best known subfamilies of cardinal trees. Ordered trees with a given degree distribution where a list of numbers n_i ($i \geq 0$) is given and the tree is guaranteed to have exactly n_i nodes with i children form a subfamily of ordinal trees studied recently by Jansson *et al.* [13]. We also investigate *free binary trees* which are free trees with maximum two children per node.

[1] $\lg n$ denotes $\log_2 n$.

Tree family	Space lower bound (Highest order term)	Succinct representation
Ordinal trees	$2n$ [14]	[11, 5, 15, 2, 7, 10]
Ordinal trees with a given degree distribution	$\sum_i n_i \lg \frac{n}{n_i}$ [20]	[13]
Cardinal trees	$(k \lg k - (k-1)\lg(k-1))\, n$ [8]	[2, 19]
Binary trees	$2n$ [8]	[11, 12, 4, 5, 15]
Free trees	$1.56n$ [18]	this paper
Free binary trees	$1.31n$ [6, 22]	this paper

Table 1. Space lower bounds in bits to represent families of trees with n nodes and references to succinct representations.

Space lower bounds on the required number of bits to represent each class of trees is obtained via information theory by counting the number of trees in the class. Table 1 illustrates the space lower bounds for these classes along with existing references to succinct representations which achieve the optimal space within lower order terms and support a variety of operations.

1.1 Contribution

We propose a uniform approach for representing trees succinctly that encompasses the families of trees in table 1. The method is based on two-level decomposition of a tree into subtrees. The recursive decomposition method is a common technique in succinct representations of various data structures [5, 19, 1] and has been used to represent trees [16, 7, 10].

In the case of ordinal trees, our approach supports a wide range of operations proposed by He *et al.* [10] and simplifies implementation of the supported operations. In the case of cardinal trees, there is no known succinct representation that supports a wide range of navigational operations. Raman *et al.* [19] state that their succinct representation for cardinal trees cannot support subtree size. Our succinct representation of cardinal trees can support all ordinal-tree-type operations listed by He *et al.* [10] (such as subtree size) as well as cardinal-tree-type operations suggested by Raman *et al.* [19] (such as following the edge labeled i from a node where $1 \leq i \leq k$).

To show the power of our method, we consider free trees which are trees with no order imposed on children of nodes and show that we can have a succinct representation taking the optimal $(1.56\ldots)n$ bits supporting all navigational operations. Similarly, free binary trees, which are free trees with maximum two children per node, can be represented in the optimal $(1.31\ldots)n$ number of bits.

Existing succinct encodings of trees assume a uniform distribution over the family of trees and therefore give worst case space guarantees. In practice however, there might be many reasons to have trees with certain property that are more likely than others, and therefore an entropy-based succinct representation is necessary. Jansson *et al.* [13] considered this case when the distribution is based on degrees of nodes and gave a representation that matches the degree-distribution entropy. Our succinct tree representation not only can match the degree-distribution entropy, but can be made adaptive to a variety of other entropy measures: *e.g.* trees with a particular probability distribution of number of children (a node has i children with probability p_i).

2 Tree Decomposition

At the heart of our method is the tree decomposition technique. Vaguely speaking, we aim to decompose the tree into subtrees of roughly the same size. Geary *et al.* [7] and He *et al.* [10] use the same decomposition algorithm to match the decomposition algorithm of Munro *et al.* [16] in the binary tree case. Given the subtree size L, the algorithm decomposes a tree into subtrees with size between L and $3L$ (with possible exception of the root subtree). Furthermore, these subtrees are disjoint other than their roots.

The drawback with their algorithms is that the number of child subtrees of a component can grow arbitrarily large (roughly as large as the size of components). With our decomposition technique, the number of child subtrees of a subtree does not exceed the original node degrees. We guarantee this by allowing (a small number of) undersized subtrees.

Theorem 1. *A tree with n nodes can be decomposed into $\Theta(n/L)$ subtrees of size at most $2L$. These are pairwise disjoint aside from the subtree roots. Furthermore, aside from edges stemming from the component root nodes, there is at most one edge per component leaving a node of a component to its child in another component.*

Figure 1 depicts the result of our decomposition algorithm. We start the proof by considering the nodes that have many descendants:

Definition 1. *For a fixed parameter L, a node is* **heavy** *if it has at least L descendants (including itself). Ancestors of a heavy node are heavy by the definition. Therefore, heavy nodes form a subtree on the original tree. We call this tree as the* **heavy-subtree***. A* **branching node** *is a node which has at least two heavy children.* **Branching edges** *are the edges between a branching node and its heavy children.*

For instance, in the tree of figure 1, heavy nodes are a, b, d, n, o, p. Branching nodes are a, n and branching edges are ab, ad, no, np. A crucial observation is that the number of branching edges is bounded (we omit the proof):

Lemma 1. *The number of branching nodes and edges in a tree with n nodes and parameter L is $O(n/L)$.* □

As with previous decomposition methods [5, 16, 7], we use a recursive bottom-up approach. To decompose a tree rooted at a node v, we first recursively decompose the trees rooted at its children $u_1, \ldots, u_k$. Each recursive call decomposes a tree rooted at a node and returns the component subtrees. Component subtrees that

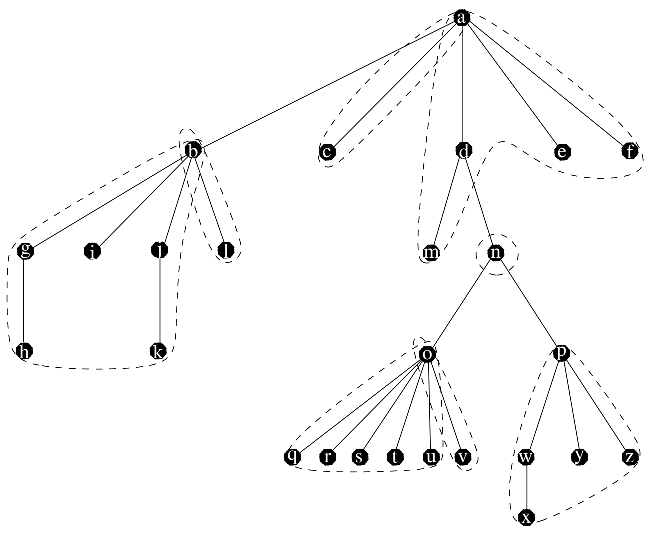

Fig. 1. A tree decomposed into component subtrees for value $L = 5$.

do not contain any of $u_1, \ldots, u_k$ are *permanent* and remain intact. The root components that contain one of $u_1, \ldots u_k$ are exception: they can be declared *temporary*. The temporary components and the parent v can possibly be merged together. The merging of the temporary components depends on the number of heavy children of v:

1. if v has no heavy children (*e.g.* node b in figure 1), entire children subtrees are temporary components to be merged together. We create a new component initially containing only v. We scan the list of children $u_1, \ldots, u_k$ from left to right adding the entire tree rooted at the current child to the component. If the component size exceeds L, we finalize that component and create a new component containing v only and continue in this manner. Since none of the children is heavy, the size of components does not exceed $2L$. The last such component can have size less than L. If we had created at least another component aside from the last component, we finalize the last component (charging its small size to its neighbor component which has the right size). Otherwise if there is only one component, we have put all descendants of v together in a component which we declare as temporary and send up to the parent of v .
2. if v has only one heavy child u_i (*e.g.* node d in figure 1 as n is heavy), we put children of v into components analogously to the previous case. The only difference occurs where the component containing u_i has been declared permanent as opposed to temporary. In this case, we simply ignore u_i, skipping from u_{i-1} to u_{i+1} during the scan.
3. If v is a branching node–*i.e.* with two or more heavy children (*e.g.* nodes a, n in figure 1)– then among children $u_1, \ldots, u_k$, there are $h \geq 2$ heavy

nodes $u_{i_1}, \ldots, u_{i_h}$. We first declare permanent the components containing these heavy nodes. If the component containing u_{i_j} for some j is undersized, we charge it to the branching edge vu_{i_j}.
If all left children are heavy, v by itself is a permanent single-node component (we charge this undersized component to branching node v itself). Otherwise, the remaining children of v are broken by the heavy nodes into intervals of consecutive non-heavy nodes. We consider the intervals separately, treating each as in the first case. The difference is we charge the possible undersized component at the end of each such interval l to one of the interval's end edges $vu_{i_{l-1}}$ or vu_{i_l} (note that $vu_{i_{l-1}}, vu_{i_l}$ are branching edges) .

One can verify that the manner the components are constructed guarantees that the number of nodes within a subtree does not exceed $2L$ and moreover, aside from the components' root nodes there is at most one edge stemming out of a node of a component to a child in another component. Furthermore, to bound the number of components, one only has to account for the undersized components. One can charge undersized components to branching edges and nodes which we know by lemma 1, there are $\Theta(n/L)$ of them. Therefore, the number of undersized components is $O(n/L)$ and thus the total number of components is $\Theta(n/L)$.

3 Ordinal trees

In this section, we outline our succinct representation for ordinal trees. The representation is analogous to Munro *et al.* [16] and the simple representation of Geary *et al.* [7] in that it is a two-level recursive decomposition of a given tree. In the first level of recursion, the tree with n nodes is first decomposed into subtrees using value $L = \lceil \lg^2 n \rceil$, and subsequently these subtrees are, in turn, decomposed into yet smaller subtrees using value $L = \lceil \lg n/4 \rceil$ to obtain the subtrees on the second level of recursion. Using the terminology of Geary *et al.* [7], we refer to as the subtrees on the first level by *mini-trees* and the second level by *micro-trees.*

Micro-trees which have size less than $\lceil \lg n/2 \rceil$ are small enough to be represented by a look-up table. The table requires $o(n)$ bits and stores encodings of all trees with sizes up to $\lceil \lg n/2 \rceil$ along with answers to variety of types of queries for each of those trees. The representation of a micro-tree with k nodes consists of two fields: the first field simply is the size of the micro-tree ($O(\log k) = O(\lg \lg n)$ bits) and the second field is an index to the look-up table ($2k$ bits). These indices sum up to $2n$ bits over all micro trees and are the dominant space term; other auxiliary data amounts to $o(n)$ bits.

Mini-trees consist of micro-trees and links between them. Links between different micro-trees can be either in form of a common root node or an edge from a non-root node from a micro-tree to the root of another micro-tree. We represent such an edge vr by introducing a *dummy node* on it. we introduce a dummy node d on the edge between the micro-tree root r and node v and replace vr by vd and dr. Edge vd is accounted for the micro-tree representation, and edge dr is

Operations	Definition
child(v,i), child_rank(v)	i^{th} child of node v, Number of left siblings of node v
degree(v), subtree_size(v)	number of children of v, Number of descendants of v
depth(v), height(v)	the depth/height of node v
leftmost(rightmost)_leaf(v), leaf_size(v)	v's leftmost/rightmost descendant leaf , number of descendant leaves
leaf_rank(v), leaf_select(v)	number of leaves before v in preorder, i^{th} leaf of the tree in preorder
node_rank$_{\text{pre}}(i)$, node_select$_{\text{pre}}(v)$	position of v in preorder, i^{th} node in pre order
node_rank$_{\text{post}}(i)$, node_select$_{\text{post}}(v)$	position of v in post order, i^{th} node in post order
level_anc(v, i), LCA(x, y), distance(x, y)	ancestor of v at level i, lowest common ancestor and distance of x, y
level_left/rightmost(i), level_succ/pred(v)	left/right most node at level i, successor or predecessor of v on its level

Table 2. Comprehensive list of operations on an ordinal tree suggested by [10]

explicitly stored as a $O(\log\log n)$-bit pointer. We refer to edges with a dummy parent such as dr as *dummy edges.* It is easy to see that these pointers require $o(n)$ space overall mini-trees.

To represent the common roots among micro-trees, we use the fully indexable dictionary (FID) of Raman *et al.* [19]. They showed that given a set S a subset of a universe U, there is a FID on S that requires $\lg\binom{|U|}{|S|}+O(|U|\log\log|U|/\log|U|)$ bits and supports rank/select on elements and non-elements of S in constant time. Given a root node v with children $u_1,\ldots,u_k$ in p different micro-trees, if $i_1,\ldots,i_p$ are the indices of children that belong to a different micro-tree than their immediate left siblings, we form set $I=\{i_1,i_2,\ldots,i_p\}$ over the universe of $[k]$. We represent this set as a FID to navigate on children of v. The required space for this FID is $\lg\binom{k}{p}+O(k\log\log k/\log k)$ bits. We omit the details of the proof that the collective space of these structures contributes only to $o(n)$.

The tree consists of mini-trees and links between them. The tree over mini-trees is represented analogously to the manner a mini-tree is represented over micro-trees: *i.e.* explicit pointers for edges coming out of mini-trees from non-root nodes and a FID to represent edges out of a common mini-tree root. One can assess the space analogously to $o(n)$.

Operations. We demonstrate that various operations on ordinal trees can be implemented trivially using our representation. Table 2 defines a comprehensive list of operations suggested by He *et al.* [10] for ordinal trees. We show an implementation for the subtree size operation as an example of how straightforward the support of operations becomes in our representation.

To compute the subtree size of node v, we explicitly store the subtree size at mini-tree roots and we store the subtree size at micro-tree roots within mini-trees, and the look-up table contains the subtree size within a micro-tree for each node of the micro-tree.

If v is a mini-tree root then the value is explicitly stored. Otherwise, if v is a micro-tree root, we have the subtree size within the mini-tree stored. If v

is not a micro-tree root then we determine the subtree size within the micro-tree from the look-up table to get the first value. Thus far, we have counted the descendants within the mini-tree; We determine if v is an ancestor of the dummy node of the mini-tree (the ancestor query is easy to support) and if so we add the subtree size value of the mini-tree stemming off the dummy node.

4 Cardinal trees

In this section, we show the uniform approach can be applied to represent cardinal (k-ary) trees. This representation is the first succinct structure that supports, in constant time, cardinal-type queries such as "find the child labeled j" as well as all ordinal-type queries such as subtree size, degree, or the i-th child.

The number of k-ary trees is $C(n,k) = \frac{1}{kn+1}\binom{kn+1}{n}$ [8] which suggests that a space-optimal representation requires $\lg C(n,k) = (k\lg k - (k-1)\lg(k-1))\,n - O(\lg(kn))$ bits. We assume a RAM model with word size $w = \max\{\lg n, \lg k\}$.

The representation is a two-level recursive decomposition of the tree analogous to the representation for ordinal trees in section 3. We decompose the tree with value $L = \lg^2 w$ into mini-trees and then recursively decompose each mini-tree into micro-trees with value $L = \max\left\{\frac{\lg w}{4\lg k}, 1\right\}$. Without loss of generality, we assume $n \geq k$ and thus $w = \lg n$. All the arguments go through analogously where $k > n$ which causes $w = \lg k$ and mini-trees with $L = \lg^2 k$ and micro-trees with $L = 1$. Hence, we can assume $L = \lg^2 n$ for mini-trees and $L = \max\left\{\frac{\lg n}{4\lg k}, 1\right\}$ for micro-trees.

The representation only differs from that of ordinal trees in how we form the look-up table and represent the roots of mini/micro-trees. We single out nodes that are roots of a micro or a mini tree and represent them separately. The representation we use is the indexable dictionary (ID) of Raman *et al.* [19] (as opposed to their fully indexable dictionary (FID)). They showed that given a set S a subset of a universe U, there is an ID on S that requires $\lg\binom{|U|}{|S|} + o(|S|) + O(\log\log|U|)$ bits and supports rank/select on elements S (In contrast to a FID, we cannot perform rank on non-elements). Here, the universe is k-slots $U = \{1, 2, \ldots, k\}$ and our subset S is the set of present edges. In contrast to ordinal trees, in a root of a micro-tree, we do not confine ourselves within the framework of the containing mini-tree and use the ID on all edges of the root. We note that all ordinal-tree structures are included in our representation such as the FID on roots of micro-trees built over the universe of present edges (confined to the containing mini-tree). The ID and FID will help us answer the cardinal-type queries as well as ordinal-type queries on a root node.

The look-up table must contain all possible micro-trees. Since we keep root nodes' information separately, the trees in the look-up table are such that all nodes are k-ary except for the roots whose children are only ordered. We refer to such trees as *root-relaxed* cardinal trees. We enumerate all root-relaxed tree of size less than $\frac{\lg n}{4\lg k}$ and list them in the lookup table.

The rest of the representation is the same as the ordinal representation: for instance, dummy nodes and edges are introduced and represented in the same manner. Now we argue that the representation is space optimal within lower order terms.

Space optimality: All auxiliary data pertinent to the ordinal tree can be proved to sum to $o(n \lg k)$ bits analogously to the proof in section 3. Thus, we only have to account for the new structures we have introduced: IDs on the root nodes and the sum of indices to the look-up table. An ID on a root node v with d_v children requires $\lg \binom{k}{d_v} + o(d_v) + O(\log \log k)$ from which the first term is dominant. Hence, the contribution of IDs to the space over the entire tree is $\sum_{v:\, root} \lg \binom{k}{d_v}$.

The space required to represent a micro-tree is the size of the index to the look-up table. Consider a root-relaxed tree T with root r and root children $r_1, \ldots, r_d$. We define T_i as the subtree rooted at child r_i and refer to its size as $n_i = |T_i|$. We use enumeration to encode root-relaxed trees and thus we obtain the shortest code. Thus the encoding requires fewer than $\sum_{i=1}^{d} (\lg n_i + \lg C(n_i, k))$ bits which sum as follows:

$$\sum_{i=1}^{d} (\lg n_i + \lg C(n_i, k)) = \lg \prod_{i=1}^{d} n_i C(n_i, k) = \lg \prod_{i=1}^{d} \binom{kn_i}{n_i - 1} \leq \lg \binom{k(|T| - 1)}{|T| - 1 - d_v}.$$

Over all micro-trees these terms together with space for IDs which is $\lg \binom{k}{d_v}$ for each root v sum to:

$$\sum_{T_i} \left(\lg \binom{k(|T_i| - 1)}{|T_i| - 1 - d_{root}} + \lg \binom{k}{d_{root}} \right) = lg \left(\prod_{T_i} \binom{k(|T_i| - 1)}{|T_i| - 1 - d_{root}} \binom{k}{d_{root}} \right),$$

which is less than $\lg \binom{kn}{n}$. Thus, the space requirement of our representation matches the lower bound, to within lower order terms: $\lg C(n, k) + o(n \log k)$.

Operations in constant time: We can support all ordinal-type operations listed in table 2 analogously to ordinal trees. To determine the child labeled i of a node v, if v is not a micro-tree root, then the answer is looked-up from the table. If v is a root node, then we use its ID to see if there is a child at that label. If it exists, we perform rank(i) to know how many siblings to the left there are. Then we can use select on the FID to actually find the mini-tree and then the micro-tree and finally the child labeled i.

5 Free trees

A *free tree* as a tree with no particular order among children of nodes. We are interested in succinct encodings of such trees allowing navigation in the tree in constant time. A *free binary tree* is a free tree such that nodes have at most two children, or alternatively a binary tree in which ignore the distinction between left and right branches. To show the power of the uniform approach we explain how these families of trees can be encoded succinctly.

Lower bounds. The lower bounds for binary and general free trees come directly from counting by information theory. Define $FB(n)$ and $F(n)$ as the number of free binary trees, and general free trees with n nodes, respectively.

There is no known explicit closed form formula for $FB(n)$ or $F(n)$. Nevertheless, asymptotic behavior of either series is well-studied [9, 18, 6, 22]. The sequence $(FB(n)), n = 1, 2, \ldots$ is known as Etherington-Wedderburn [6, 22] sequence and from its asymptotic behavior one can infer that asymptotically $\lg FB(n) = (1.3122\ldots)n + o(n)$. Otter [18] described the asymptotic behavior for $F(n)$ from which one obtains that asymptotically $\lg F(n) = (1.5639\ldots n) + o(n)$. This implies that free trees can potentially be represented more space-efficiently than ordinal trees which require $2n + o(n)$ bits.

Theorem 2. *The information-theoretic lower bound on the number of bits required to represent free binary trees and free general trees with n nodes is $(1.3122\ldots)n + o(n)$ and $(1.5639\ldots)n + o(n)$ respectively.* □

Upper bounds. The representation differs from that of ordinal trees in section 3 in the look-up table. In the case of free binary trees, all free binary trees of size up to $\frac{1}{4}\lg n$ are enumerated modulo isomorphisms and listed in the look-up table in increasing order of their sizes. To represent a micro-tree we use a pair (k, i) index to the table. k is the size of the micro-tree and i is the index to the look-up table which is an offset from the start location of trees with size k. All auxiliary data are carried forward from ordinal trees as they only take $o(n)$ space. One can easily argue that the total bits required by the first fields of pairs (k) is also $o(n)$. Therefore, the dominant field is the sum of the bits of the second fields of pairs (i). Theorem 2 suggest that the size of this field for a tree of size t is $(1.31\ldots)t + o(t)$ bits. The second term $o(t)$ term adds up to $o(n)$ over the entire tree. The first term is the dominant term which adds up to $(1.31\ldots)n + o(n)$ over the entire tree when n is the number of nodes. The $(1.56\ldots)n + o(n)$ bit representation for free general trees is analogous; the look-up table lists free general trees as opposed to free binary trees:

Theorem 3. *The succinct representation for free binary trees and free general trees with n nodes requires $(1.3122\ldots)n + o(n)$ and $(1.5639\ldots)n + o(n)$ bits respectively and supports all navigational operations listed in table 2 in constant time.* □

6 Entropy-based succinct encodings

Thus far, we have assumed a uniform distribution among trees belonging to a certain family of trees. However, there might be many applications so that some trees are biased against other trees within the tree family, and therefore the distribution is non-uniform. Thus, entropy-based succinct encodings are necessary. Jansson *et al.* [13] were the first to give entropy-based succinct encodings for the degree-distribution entropy. In this section, we show how our method can be used to match the degree-distribution entropy as well as a variety of other entropy measures.

6.1 Succinct encoding based on degree-distribution entropy

The degree-distribution of an ordinal tree with n nodes is a series of numbers $(n_0, n_1, \ldots)$ such that the tree has n_i nodes with exactly i children ($\sum_i n_i = n$ and $\sum_i i\, n_i = n - 1$). Rote [20] showed that the number of trees with a given degree-distribution is $\frac{1}{n}\binom{n}{n_0,\ldots,n_{n-1}}$, the logarithm of which is $L(T) = \sum_i n_i \lg \frac{n}{n_i}$ to within lower order terms. $L(T)$ is therefore a lower bound on the required number of bits to represent the tree.

Jansson *et al.* [13] gave a representation that requires $L(T) + O\left(\frac{n(\log\log n)^2}{\log n}\right)$ number of bits and supports a variety of operations in constant time. Using our approach we obtain another space-optimal succinct representation with $L(T) + O\left(n \log\log\log n / \log\log n\right)$ number of bits supporting all operations in table 2 in constant time. They did not assume that the degree-distribution is explicitly given. We observe that we can make explicit the assumption that the degree-distribution is given as it takes negligible space to encode the sequence and have it explicitly stored. Thus we can accompany it with the succinct representation.

Our succinct representation sensitive to degree-distribution entropy is the same as that of the ordinal trees with the difference in the look-up table. The look-up table contains all trees with less than $\frac{1}{4}\lg n$ as in ordinal trees; However, the trees are ordered based on their degree-distribution sequence in the lexicographical order and listed in the table accordingly. In order to encode a micro-tree T, we use an index to the table. The index to the table is a pair $(\mathcal{N}, k)$ where $\mathcal{N}$ is the degree-distribution encoding of the tree and k is an offset in the table from where the trees with degree-distribution $\mathcal{N}$ start to the actual position of tree T we reference to.

One can easily verify that the total number of bits required by the first fields of indices (*i.e.* $\mathcal{N}$) is negligible. The second field k is the dominant term. The size of this field for a micro-tree T_t, by a counting argument, is $\lceil L(T_t) \rceil = \left\lceil \lg\left(\frac{1}{|T_t|}\binom{|T_t|}{n_{t,0}\ldots n_{t,|T_t|}}\right)\right\rceil$ where $n_{t,i}$ is the number of nodes with i children in tree T_t. A node with more than $\log n$ children in the original tree is a micro-tree root and its children are split among different micro-trees. Since there are $\Theta\left(n \log n\right)$ of these roots and each micro-tree root contributes at most $\log\log n$ bits to the space, the sum of these terms is $O\left(n \log\log n / \log n\right)$ and within our space bound. The sum over all micro-trees $T_1, \ldots, T_m$ modulo their roots can be assessed as follows:

$$\begin{aligned}
\sum_{t=1}^{m} \lg\left(\frac{1}{|T_t|}\binom{|T_t| - 1}{n_{t,0}, \ldots, n_{t,|T_t|}}\right) &= \lg \prod_t \frac{1}{|T_t|}\binom{|T_t| - 1}{n_{t,0} \ldots n_{t,|T_t|}} \\
&\le \lg\left(\frac{1}{n}\binom{n}{n_0, \ldots, n_{n-1}}\right) \approx L(T).
\end{aligned}$$

Hence, the representation has the optimal space within lower order terms and clearly we can perform all operations listed in table 2 in constant time as in an ordinal tree.

6.2 Other entropy measures

Similar to the manner in which we represented trees adaptive to their degree-distribution entropy, we can use the approach to obtain succinct representations adaptive to various other combinatorial properties and entropy measures. For instance, consider the family of ordinal trees such that internal nodes have at least two children. The number of such trees with n nodes is known as Riordan number [3]. The logarithm based two of Riordan numbers is asymptotically $\lg(3)n + o(n) \approx 1.58n + o(n)$. One can use our approach to encode this family of trees. Similarly, The family of ordinal trees with a fixed constant upper bound d on the number children of a node can be represented in the same manner. More generally, where there is a probability distribution for the number of children of a node, our representation can match the entropy bound.

Another interesting family of trees is AVL trees which consists of binary trees such that the height of left and right subtrees differ by at most one. Odlyzko [17] showed that if a_n is the number of AVL trees with n nodes, $\lg a(n) \approx (0.9381\ldots)n$, our representation matches this entropy bound and thus can represent AVL trees in optimal number of bits to within lower order terms.

7 Conclusion

In this paper, we proposed a uniform approach towards succinct representation of trees. We showed that all families of trees with an existing succinct representation can be represented using our framework. Our representation improves on the existing ones on cardinal trees as we are able to answer ordinal-type queries such as subtree size as well as cardinal-type queries. We introduce a new family of trees: *free trees*. We demonstrated how easily our approach can represent these trees succinctly We argued that our approach can represent trees succinctly adaptive to the degree-distribution entropy. We discussed that a variety of other entropy measures can be dealt with similarly.

References

1. Barbay, J., He, M., Munro, J. I., and Rao, S. S. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2007), ACM-SIAM, pp. 680–689.
2. Benoit, D., Demaine, E. D., Munro, J. I., Raman, R., Raman, V., and Rao, S. S. Representing trees of higher degree. *Algorithmica 43*, 4 (2005), 275–292.
3. Bernhart, F. Catalan, motzkin, and riordan numbers. *Discrete Mathematics 204* (1999), 72–112.
4. Clark, D. R. *Compact pat trees.* PhD thesis, Waterloo, Ontario., Canada., 1998.
5. Clark, D. R., and Munro, J. I. Efficient suffix trees on secondary storage (extended abstract). In *SODA* (1996), pp. 383–391.
6. Etherington, I. M. H. Non-associate powers and a functional equation. *The Mathematical Gazette 21*, 242 (1937), 36–39.

7. Geary, R. F., Raman, R., and Raman, V. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms 2*, 4 (2006), 510–534.
8. Graham, R. L., Knuth, D. E., and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.
9. Harary, F., and Palmer, E. M. *Graphical Enuemration*. Academic Press, New York, 1973.
10. He, M., Munro, J. I., and Rao, S. S. Succinct ordinal trees based on tree covering. In *ICALP* (2007), vol. 4596 of *Lecture Notes in Computer Science*, Springer, pp. 509–520.
11. Jacobson, G. Space-efficient static trees and graphs. *Foundations of Computer Science, 1989., 30th Annual Symposium on* (30 Oct-1 Nov 1989), 549–554.
12. Jacobson, G. J. *Succinct static data structures*. PhD thesis, Pittsburgh, PA, USA, 1988.
13. Jansson, J., Sadakane, K., and Sung, W.-K. Ultra-succinct representation of ordered trees. In *SODA* (2007), N. Bansal, K. Pruhs, and C. Stein, Eds., SIAM, pp. 575–584.
14. Knuth, D. E. *The Art of Computer Programming*, third ed., vol. 1. Addison-Wesley, 1997.
15. Munro, J. I., and Raman, V. Succinct representation of balanced parentheses, static trees and planar graphs. In *IEEE Symposium on Foundations of Computer Science* (1997), pp. 118–126.
16. Munro, J. I., Raman, V., and Storm, A. J. Representing dynamic binary trees succinctly. In *SODA* (2001), pp. 529–536.
17. Odlyzko, A. M. Some new methods and results in tree enumeration, May 04 1984.
18. Otter, R. The number of trees. *The Annals of Mathematics, 2nd Ser. 49*, 3 (1948), 583–599.
19. Raman, R., Raman, V., and Rao, S. S. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *SODA* (2002), pp. 233–242.
20. Rote, G. Binary trees having a given number of nodes with 0,1, and 2 children. *Sminaire Lotharingien de Combinatoire 38* (1997).
21. Storm, A. J. Representing dynamic binary trees succinctly. Master's thesis, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2000.
22. Wedderburn, J. H. M. The functional equation $g(x^2) = 2ax + [g(x)]^2$. *The Annals of Mathematics, 2nd Ser. 24*, 2 (1922), 121–140.