Tree Compression and Indexing
(2005; Ferragina, Luccio, Manzini, Muthukrishnan)

Paolo Ferragina, University of Pisa, www.di.unipi.it/~ferragin
S. Srinivasa Rao, IT University of Copenhagen, www.itu.dk/people/ssrao

entry editor: Paolo Ferragina

INDEX TERMS: Trees, Indexing data structures, Data compression, Tree navigation and search.
SYNONYMS: XML compression and indexing.

1 PROBLEM DEFINITION

Trees are a fundamental structure in computing. They are used in almost every aspect of modeling and representation for explicit computation like searching for keys, maintaining directories, and representations of parsing or execution traces—to name just a few. One of the latest uses of trees is `XML`, the *de facto* format for data storage, integration, and exchange over the Internet (see `http://www.w3.org/XML/`). Explicit storage of trees, with one pointer per child as well as other auxiliary information (e.g. label), is often taken as given but can account for the dominant storage cost. Just to have an idea, a simple tree encoding needs at least 16 bytes per tree node: one pointer to the auxiliary information (e.g. node label) plus three node pointers to the parent, the first child, and the next sibling. This large space occupancy may even prevent the processing of medium size trees, e.g. `XML` documents. This entry surveys the best known storage solutions for unlabeled and labeled trees that are space efficient and support fast navigational and search operations over the tree structure. In the literature, they are referred to as *succinct/compressed tree indexing* solutions.

Notation and basic facts. Consider a rooted tree $\mathcal{T}$ of arbitrary degree and shape. In what follows it will be detailed three main classes of trees and their information-theoretic storage costs. The storage lower bound is derived via a simple *counting argument*: at least $\log |U|$ bits are needed to distinguish any two objects of a universe U.[1]

Ordinal Trees. $\mathcal{T}$ is unlabeled and its children are left-to-right *ordered*. The number of ordinal trees on t nodes is $C_t = \binom{2t}{t}/(t+1)$ which induces a lower bound of $2t - \Theta(\log t)$ bits.

Cardinal k-ary Trees. $\mathcal{T}$ is labeled on its *edges* with symbols drawn from alphabet Σ, where $k = |\Sigma|$. Any node has degree at most k because the edges outgoing from each node have *distinct* labels. Typical examples of cardinal trees are the binary tree ($k = 2$), the (uncompacted) trie and the *Patricia tree*. The number of k-ary cardinal trees on t nodes is $C_t^k = \binom{kt+1}{t}/(kt+1)$ which induces a lower bound of $t(\log k + \log e)$ bits, when k is a slowly-growing function of t.

(Multi-)Labeled Trees. $\mathcal{T}$ is an ordinal tree, labeled on its *nodes* with symbols drawn from alphabet Σ. In the case of multi-labeled trees, every node has *at least* one symbol as its label. The *same* symbols may repeat among sibling nodes, so that the degree of each node is *unbounded*, and the same labeled-subpath may occur many times in $\mathcal{T}$, anchored anywhere.

[1] Throughout the entry, all logarithms are taken to the base 2, and it is assumed $0 \log 0 = 0$.

The information-theoretic lower bound on the storage complexity of this class of trees on t nodes comes easily from the decoupling of the tree structure and the storage of tree labels. For labeled trees it is $\log C_t + t \log |\Sigma| = t(\log |\Sigma| + 2) - \Theta(\log t)$ bits.

The following query operations should be supported over $\mathcal{T}$:

Basic navigational queries. They ask for the parent of node u, the ith child of u, the degree of u. These operations may be restricted to some label $c \in \Sigma$, if $\mathcal{T}$ is labeled.

Sophisticated navigational queries. They ask for the jth level-ancestor of u, the depth of u, the subtree size of u, the lowest common ancestor of a pair of nodes, the ith node according to some node ordering over $\mathcal{T}$, possibly restricted to some label $c \in \Sigma$ (if $\mathcal{T}$ is labeled). For even more operations see [2, 10].

Subpath query. Given a labeled subpath Π, it asks for the (number *occ* of) nodes of $\mathcal{T}$ that immediately descend from Π. Every subpath occurrence may be anchored anywhere in the tree (i.e. not necessarily in its root).

The elementary solution to the tree indexing problem consists of encoding the tree $\mathcal{T}$ via a mixture of pointers and arrays, thus taking a total of $\Theta(t \log t)$ bits. This supports basic navigational operations in constant time, but is not space efficient and requires the whole visit of the tree to implement the subpath query or the more sophisticated navigational operations! Here the goal is to design tree storage schemes that are either *succinct*, namely "close to the information-theoretic lower bound" mentioned before, or *compressed* in that they achieve "entropy-bounded storage". Thus, succinct/compressed tree indexing solutions are distinct from simply compressing the input, and then uncompressing it later on at query time!

In this entry, it is assumed that $t \geq |\Sigma|$ and the Random Access Machine (RAM) with word size $\Theta(\lg t)$ is taken as model of computation. This way, one can perform various arithmetic and bit-wise boolean operations on single words in constant time.

2 KEY RESULTS

The notion of *succinct* data structures was introduced by Jacobson [9] in a seminal work over eighteen years ago. He presented a storage scheme for ordinal trees using $2t + o(t)$ bits and supporting basic navigational operations in $O(\log \log t)$ time (i.e. parent, first child and next sibling of a node). Later, Munro and Raman [12] closed the issue for ordinal trees on basic navigational queries and the subtree-size query by achieving constant query-time and $2t + o(t)$ bits of storage. Their storage scheme is called *Balanced Parenthesis* (shortly, `BP`).[2] Subsequently, Benoit *et al.* [3] proposed a storage scheme called *Depth-First Unary Degree Sequence* (shortly, `DFUDS`) that still uses $2t + o(t)$ bits but performs more navigational operations like ith child, child rank, and node degree in constant time. Geary *et al.* [8] gave another representation still taking asymptotically optimal space that extends `DFUDS`'s operations to the level-ancestor query.

Although these three representations achieve the optimal space occupancy, none of them supports every existing operation in constant time: e.g. `BP` does not support ith child and child rank, `DFUDS` and Geary *et al.*'s representation do not support `LCA`. Recently Jansson *et al.* [10] extended the `DFUDS` storage scheme toward two directions: (1) they showed how to implement in constant

[2] Some papers [Chiang *et al.*, ACM-SIAM SODA '01; Sadakane, ISAAC '01; Munro *et al.*, J.ALG '01; Munro and Rao, ICALP '04] have extended `BP` to support in constant time other sophisticated navigational queries like `LCA`, node degree, rank/select on leaves and number of leaves in a subtree, level-ancestor and level-successor.

time all navigational operations above; (2) they showed how to compress the new tree storage scheme up to $H^d(\mathcal{T})$, which denotes the entropy of the distribution of node degrees in $\mathcal{T}$.[3]

Theorem 1. [Jansson et al. 2007] *For any rooted tree $\mathcal{T}$ with t nodes, there exists a tree indexing scheme that uses $tH^d(\mathcal{T}) + O(t(\log\log t)^2/\log t)$ bits and supports all navigational operations in constant time.*

This improves the standard tree pointer-based representation, since it needs no more than $H^d(\mathcal{T})$ bits per node and does not compromise the performance of sophisticated navigational operations. In the case of highly regular trees, it is $H^d(\mathcal{T}) \leq 2$, so the improvement may be significant! This result can be extended to achieve the kth order entropy of the `DFUDS` sequence, by adopting any compressed-storage scheme for strings (see e.g. [7] and references therein).

Benoit *et al.* [3] extended the use of `DFUDS` to cardinal trees, and proposed a tree indexing scheme whose space occupancy is close to the information-theoretic lower bound and supports various navigational operations in constant time. Raman *et al.* [14] improved the space by using a different approach (based on storing the tree as a set of edges) thus proving the following:

Theorem 2. [Raman et al. 2002] *For any k-ary cardinal tree $\mathcal{T}$ with t nodes, there exists a tree indexing scheme that uses $\log C_t^k + o(t) + O(\log\log k)$ bits and supports in constant time the following operations: finding the parent, the degree, the ordinal position among its siblings, the child with label c, the ith child of a node.*

The subtree size operation cannot be supported efficiently using this representation, so [3] should be resorted to in case this operation is a primary concern.

Despite this flurry of activity, the fundamental problem of indexing *labeled* trees succinctly has remained mostly unsolved. In fact, the succinct encoding for ordered trees mentioned above might be replicated $|\Sigma|$ times (one per possible symbol of Σ), and then the divide-and-conquer approach of [8] might be applied to reduce the final space occupancy. However, the final space bound would be $2t + t\log|\Sigma| + O(t|\Sigma|\frac{\log\log\log t}{\log\log t})$ bits, which is nonetheless far from the information-theoretic storage bound even for moderately large Σ. On the other hand, if subpath queries are of primary concern (e.g. `XML`), one can use the approach of [11] which consists of a variant of the suffix-tree data structure properly designed to index all $\mathcal{T}$'s labeled paths. Subpath queries can be supported in $O(|\Pi|\log|\Sigma| + occ)$ time, but the required space would be still $\Theta(t\log t)$ bits (with large hidden constants due to the use of suffix trees). Recently, some papers [1, 2, 5] addressed this problem in its whole generality by either dealing simultaneously with subpath queries and basic navigational operations [5], or by considering *multi*-labeled trees and a larger set of navigational operations [1, 2].

The tree-indexing scheme of [5] is based on a *transform* of the labeled tree $\mathcal{T}$, denoted $\mathsf{xbw}[\mathcal{T}]$, which linearizes it into *two* coordinated arrays $\langle \mathcal{S}_{\mathsf{last}}, \mathcal{S}_\alpha \rangle$: the former capturing the tree structure and the latter keeping a permutation of the labels of $\mathcal{T}$. $\mathsf{xbw}[\mathcal{T}]$ has the optimal (up to lower-order terms) size of $2t + t\log|\Sigma|$ bits and can be built and inverted in optimal linear time. In designing the `XBW`-transform, the authors were inspired by the elegant Burrows-Wheeler transform for strings [4]. The power of $\mathsf{xbw}[\mathcal{T}]$ relies in the fact that it allows to transform compression and indexing problems on labeled trees into easier problems over strings. Namely, the following two string-search primitives are key tools for indexing $\mathsf{xbw}[\mathcal{T}]$: $\mathsf{rank}_c(S, i)$ returns the number of occurrences of the symbol c in the string prefix $S[1, i]$, and $\mathsf{select}_c(S, j)$ returns the position of the j-th occurrence of the symbol c in string S. The literature offers many time/space efficient solutions for these primitives that could be used as a *black-box* for the compressed indexing of $\mathsf{xbw}[\mathcal{T}]$ (see e.g. [2, 13] and references therein).

[3] The `BP` representation and the one of Geary *et al.* [8] have been recently extended to support further operations— like depth/height of a node, next node in the same level, rank/select over various node orders— still in constant time and $2t + o(t)$ bits [Personal communication by Rao, 2006].

Theorem 3. [Ferragina et al. 2005] *Consider a tree $\mathcal{T}$ consisting of t nodes labeled with symbols drawn from alphabet Σ. There exists a tree indexing scheme that has the following performance:*

- *if $|\Sigma| = O(\text{polylog}(t))$, the compressed index takes at most $tH_0(\mathcal{S}_\alpha) + 2t + o(t)$ bits, supports basic navigational operations in constant time and subpath searches in $O(|\Pi|)$ time.*
- *if $|\Sigma| = \Omega(\text{polylog}(t))$, the compressed index takes at most $t(H_k(\mathcal{S}_\alpha) + H_k(\mathcal{S}_{\mathsf{last}}) + o(\log|\Sigma|))$ bits, but navigational operations and queries are slowed down by a factor of $o(\log\log^3|\Sigma|)$.*

where $H_k(s)$ is the kth order empirical entropy of string s, with $H_k(s) \leq H_{k-1}(s)$ for any $k > 0$.

Basic navigational operations mean the parent of a node u, the ith child of u with some label c, or the degree of u. Note that $H_k(\mathcal{S}_\alpha) \leq H_0(\mathcal{S}_\alpha) \leq \log|\Sigma|$, hence the indexing of $\mathsf{xbw}[\mathcal{T}]$ takes the same space of its plain representation, up to lower order terms, but now one can additionally navigate and search $\mathcal{T}$ efficiently. This is indeed a sort of *pointerless representation* of the labeled tree $\mathcal{T}$ with additional search functionalities!

If sophisticated navigational operations over labeled trees are a primary concern, and subpath queries are not necessary, then the approach of Barbay *et al.* [1, 2] should be followed. They proposed the novel concept of *succinct index*, which is different from the concept of *succinct/compressed encoding* implemented by all the above solutions. A succinct index does not *touch* the data to be indexed, it just accesses the data via basic operations offered by the underlying abstract data type (ADT), and requires asymptotically less space than the information-theoretic lower bound on the storage of the data itself. The authors reduce the problem of indexing labeled trees to the one of indexing ordinal trees and strings; and the problem of indexing multi-labeled trees to the one of indexing ordinal trees and binary relations. Then, they provide succinct indexes for strings and binary relations. In order to present their result, the following definitions are needed. Let t_c be the number of nodes labeled c in $\mathcal{T}$, and let ρ_c be the maximum number of labels c in any rooted path of $\mathcal{T}$ (called the *recursivity* of c). Define ρ as the average recursivity of $\mathcal{T}$, namely $\rho = (1/m)\sum_{c\in\Sigma}(t_c\rho_c)$.

Theorem 4. [Barbay et al. 2007] *Consider a tree $\mathcal{T}$ consisting of t nodes (multi-)labeled with possibly many symbols drawn from alphabet Σ. Let m be the total number of symbols in $\mathcal{T}$, and assume that the underlying ADT for $\mathcal{T}$ offers basic navigational operations in constant time and retrieves the ith label of a node in time f. There is a succinct index for $\mathcal{T}$ using $m(\log\rho + o(\log(|\Sigma|\rho)))$ bits that supports for a given node u the following operations (where $L = \log\log|\Sigma|\ \log\log\log|\Sigma|$):*

- *every c-descendant or c-child of u can be retrieved in $O(L\,(f + \log\log|\Sigma|))$ time.*
- *the set A of c-ancestors of u can be retrieved in $O(L\,(f + \log\log|\Sigma|) + |A|\,(\log\log\rho_c + \log\log\log|\Sigma|\,(f + \log\log|\Sigma|)))$ time.*

3 APPLICATIONS

As trees are ubiquitous in many applications, this section concentrates just on two examples that, in their simplicity, highlight the flexibility and power of succinct/compressed tree indexes.

The first example regards suffix trees, which are a crucial algorithmic block of many string processing applications— ranging from bioinformatics to data mining, from data compression to search engines. Standard implementations of suffix trees take at least 80 bits per node. The compressed suffix tree of a string $S[1, s]$ consists of three components: the tree topology, the string depths stored into the internal suffix-tree nodes, and the suffix pointers stored in the suffix-tree leaves (also called *suffix array* of S). The succinct tree representation of [10] can be used to encode the suffix-tree topology and the string depths taking $4s + o(s)$ bits (assuming w.l.o.g. that $|\Sigma| = 2$). The suffix array can be compressed up to the kth order entropy of S via any solution surveyed in

[13]. The overall result is never worse than 80 bits per node, but can be significantly better for highly compressible strings!

The second example refers to the XML format which is often modeled as a labeled tree. The succinct/compressed indexes in [1, 2, 5] are theoretical in flavor but turn out to be relevant for practical XML processing systems. As an example, [6] has published some initial encouraging experimental results that highlight the impact of the XBW-Transform on real XML datasets. The authors show that a proper adaptation of the XBW-Transform allows to compress XML data up to state-of-the-art XML-conscious compressors, and to provide access to its content, navigate up and down the XML tree structure, and search for simple path expressions and substrings in few milliseconds over MBs of XML data, by uncompressing only a tiny fraction of them at each operation. Previous solutions took several seconds per operation!

4 OPEN PROBLEMS

For a complete set of open problems and further directions of research, the interested reader is referred to the recommended readings. Here two main problems, which naturally derive from the discussion above, are commented.

Motivated by XML applications, one could like to extend the subpath search operation to the *efficient* search for all leaves of $\mathcal{T}$ whose labels *contain* a substring β and that *descend from* a given subpath Π. The term "efficient" here means in time proportional to $|\Pi|$ and to the number of retrieved occurrences, but independent as much as possible of $\mathcal{T}$'s size in the worst case. Currently, this search operation is possible only for the leaves which are immediate descendant of Π, and even for this setting, the solution proposed in [6] is not optimal.

There are two main encodings for trees which lead to the results above: ordinal tree representation (BP, DFUDS or the representation of Geary *et al.*) and XBW. The former is at the base of solutions for sophisticated navigational operations, and the latter is at the base of solutions for sophisticated subpath searches. Is it possible to devise *one unique* transform for the labeled tree $\mathcal{T}$ which combines the best of the two worlds and is still compressible?

5 EXPERIMENTAL RESULTS

Look at http://cs.fit.edu/~mmahoney/compression/text.html and at the paper [6] for numerous experiments on XML datasets.

6 DATA SETS

Look at http://cs.fit.edu/~mmahoney/compression/text.html and the references in [6].

7 URL to CODE

Paper [6] contains a list of software tools for compression and indexing of XML data.

8 CROSS REFERENCES

Compressed Text Indexing; Rank and Select over Binary Strings; Succinct Encoding of Permutations and its Applications to Text Indexing; Table Compression; Text Indexing.

9 RECOMMENDED READING

[1] J. Barbay, A. Golynski, I. Munro, and S. S. Rao, *Adaptive searching in succinctly encoded binary relations and tree-structured documents*, in Proc. 17th Combinatorial Pattern Matching conference (CPM), 2006, pp. 24–35.

[2] J. Barbay, M. He, J. Munro, and S. S. Rao, *Succinct indexes for string, bynary relations and multi-labeled trees*, in Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

[3] D. Benoit, E. Demaine, I. Munro, R. Raman, V. Raman, and S. S. Rao, *Representing trees of higher degree*, Algorithmica, 43 (2005), pp. 275–292.

[4] M. Burrows and D. Wheeler, *A block sorting lossless data compression algorithm*, Tech. Report 124, Digital Equipment Corporation, 1994.

[5] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan, *Structuring labeled trees for optimal succinctness, and beyond*, in Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS), 2005, pp. 184–193.

[6] ——, *Compressing and searching XML data via two zips*, in Proc. 15th International World Wide Web Conference (WWW), 2006, pp. 751–760.

[7] P. Ferragina and R. Venturini, *A simple storage scheme for strings achieving entropy bounds*, in Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

[8] R. Geary, R. Raman, and V. Raman, *Succinct ordinal trees with level-ancestor queries*, in Proc. 15th ACM-SIAM symposium on Discrete Algorithms (SODA), 2004, pp. 1–10.

[9] G. Jacobson, *Space-efficient static trees and graphs*, in Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), 1989, pp. 549–554.

[10] J. Jansson, K. Sadakane, and W. Sung, *Ultra-succinct representation of ordered trees*, in Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

[11] S. R. Kosaraju, *Efficient tree pattern matching*, in Proc. 20th IEEE Foundations of Computer Science (FOCS), 1989, pp. 178–183.

[12] I. Munro and V. Raman, *Succinct representation of balanced parentheses and static trees*, SIAM Journal on Computing, 31:3 (2001), pp. 762–776.

[13] G. Navarro and V. Mäkinen, *Compressed full text indexes*, ACM Computing Surveys, 2007 (To appear).

[14] R. Raman, V. Raman, and S. S. Rao, *Succinct indexable dictionaries with applications to encoding k-ary trees and multisets*, in Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 233–242.