Adaptive Searching in Succinctly Encoded Binary Relations and Tree-Structured Documents *Extended Abstract*

Jérémy Barbay[1], Alexander Golynski[1], J. Ian Munro[1], and S. Srinivasa Rao[2]

[1] School of Computer Science
University of Waterloo, Canada.
[2] Dept. of Theoretical Computer Science
IT University of Copenhagen, Denmark.

Abstract. This paper deals with succinct representations of data types motivated by applications in posting lists for search engines, in querying XML documents, and in the more general setting (which extends XML) of multi-labeled trees, where several labels can be assigned to each node of a tree.

To find the set of references corresponding to a set of keywords, one typically intersects the list of references associated with each keyword. We view this instead as having a single list of objects $[n] = \{1, \ldots, n\}$ (the references), each of which has a subset of the labels $[\sigma] = \{1, \ldots, \sigma\}$ (the keywords) associated with it. We are able to find the objects associated with an arbitrary set of keywords in time $O(\delta k \lg \lg \sigma)$ using a data structure requiring only $t(\lg \sigma + o(\lg \sigma))$ bits, where δ is the number of steps required by a non-deterministic algorithm to check the answer, k is the number of keywords in the query, σ is the size of the set from which the keywords are chosen, and t is the number of associations between references and keywords. The data structure is succinct in that it differs from the space needed to write down all t occurrences of keywords by only a lower order term.

An XML document is, for our purpose, a labeled rooted tree. We deal primarily with "non-recursive labeled trees", where no label occurs more than once on any root to leaf path. We find the set of nodes which path from the root include a set of keywords in the same time, $O(\delta k \lg \lg \sigma)$, on a representation of the tree using essentially minimum space, $2n + n(\lg \sigma + o(\lg \sigma))$ bits, where n is the number of nodes in the tree. If we permit nodes to have multiple labels, this space bound becomes $2n + t(\lg \sigma + o(\lg \sigma))$ bits, that is the information theoretic lower bound for an ordinal tree (a node can have an arbitrary number of children ordered from left to right) plus that for the multiple labeling, where t is the total number of labels assigned.

In proving those results, we consider two data-structures if independant interest: we give an encoding for σ by n boolean matrices, using optimal space and supporting in time $O(\lg \lg \sigma)$ the operators access (the value at the intersection of a row and a column) rank (how many matches occur in this row to the left of this entry, or how many are in this column and above), and select (find the r-th match in this row, or in this column); and we give an encoding for labeled trees of n nodes and σ labels, using optimal space and supporting in time $O(\lg \lg \sigma)$ the labeled based operator `labeltree_desc`(a, x), which finds the first descendant of x labeled a.

Keywords: succinct data-structures, labeled trees, multi-labeled trees, conjunctive queries, intersection problem, opt-threshold set.

[1] TECH REPORT CS-2005-35 ($Date: 2005/11/16 14:21:01 Revision: 1.90 $)

1 Introduction

We consider succinct data structures motivated by applications in posting lists for search engines, in querying XML documents, and in querying multi-labeled trees (such as file systems), which generalize XML documents in that several labels can be assigned to each node of the tree. To solve those queries, we consider adaptive algorithms whose performance is expressed as a function of the non-deterministic complexity of the instance.

Succinct data structures were introduced by Jacobson [22], to encode bit vectors, (unlabeled) trees and planar graphs in space essentially equal to the information-theoretic lower bound, while supporting operators on it efficiently. For *bit vectors*, he defined the following useful operators:

- $\texttt{bin\_rank}(c, p)$: the number of occurences of character $c \in \{0, 1\}$ before position p.
- $\texttt{bin\_select}(c, i)$: the position of the ith occurence of character $c \in \{0, 1\}$ in the sequence.

Clark and Munro [10] showed later that those operators could be supported in constant time on a bit vector of lenght n using only $n + o(n)$ bits. Golynski *et al.* [19] generalized this problem to alphabets of arbitrary size σ, extending the operators to $\texttt{string\_rank}(a, x)$, the number of occurences of a before position x, $\texttt{string\_select}(a, r)$, the position of the r-th occurence of a in the sequence, and added an operator $\texttt{string\_access}(x)$, which corresponds to the character at position x of the sequence. They give two different encodings which both use $n(\lg \sigma + o(\lg \sigma))$ bits, and support the operators in the following times:

	`select`	`access`
$\texttt{string\_rank}(a, x)$	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
$\texttt{string\_select}(a, r)$	$O(1)$	$O(\lg \lg \sigma)$
$\texttt{string\_access}(x)$	$O(\lg \lg \sigma)$	$O(1)$

We extend the problem to the encoding of sequences of n objects where each object can be associated with several labels, this association being defined by a binary relation of t pairs from $[n]\times[\sigma]$. We give two representations (Th. 1), which use $t(\lg \sigma + o(\lg \sigma))$ bits each and support the operators extended to binary relations in the same time than G. *et al.*, and support the orthogonal operators in sligthly larger times.

For *unlabeled trees*, Jacobson proposed a succinct data structure to store an tree of n nodes, which supports the standard navigational operations in time $O(\lg n)$. Munro and Raman [26] gave the first representation that supports in constant time the operators $\texttt{tree\_parent}(x)$, the parent of x; $\texttt{tree\_degree}(x)$, the number of children of x; and $\texttt{tree\_nbdesc}(x)$, the number of descendants of x. Benoit *et al.* gave another tree representation that supports in constant time the operator $\texttt{tree\_child}(i)$, which finds the i-child, in addition to the other operators. Geary et al. [18] proposed a structure supporting in constant time all the operators cited above, and added to it several operators supported in constant time: $\texttt{tree\_leveled\_ancestor}(x, i)$, which provides the ith ancestor of a node x; $\texttt{tree\_rank}(x)$ and $\texttt{tree\_select}(r)$ which provide a bijection between the nodes x of the tree and their pre-order rank r; $\texttt{tree\_childrank}(x)$ which provides the rank of a node among the children of its parent; and $\texttt{tree\_depth}(x)$ which provides the number of edges separating the root from a node x. All these structures take $2n + o(n)$ bits, which is asymptotically equivalent, when n is large, to the lower bound suggested by information theory.

Geary *et al.* extended those operators to consider labels associated to the nodes of the tree and to support them in constant time, but their data-structure for label-based operators on labeled trees uses $2n + n\big(\lg \sigma + O(\sigma n \lg \lg \lg n / \lg \lg n)\big)$ bits, which is much more than the asymptotic lower bound of $2n + n(\lg \sigma - o(\lg \sigma))$ suggested by information theory when σ is large. Ferragina *et al.* [17] proposed another structure which supports in constant time the operator $\texttt{labeltree\_child}(a, x)$, which finds the first child[3] labeled

[3] Ferragina *et al.*'s encoding also supports finding all the children of x labeled a in constant time.

a of x , but this structure does not support the operators `labeltree_anc` or `labeltree_desc`, and it uses $2n \lg \sigma + O(n)$ bits, which is almost twice the minimum space required to encode the tree.

We give (see Cor. 1) an encoding for labeled trees using $2n + n(\lg \sigma + o(\lg \sigma))$ bits, which is asymptotically the minimum space required to encode the labeled tree. It supports operators `labeltree_parent`, `labeltree_nbdesc`, `labeltree_desc` and `labeltree_anc` in $O(\lg \lg \sigma)$ on non-recursive labeled trees, i.e. labeled trees where no label occurs more than once on any root to leaf path. We also give (see Th. 3) a second encoding, using the same space but which supports operators `labeltree_parent`, `labeltree_nbdesc`, `labeltree_desc` and `labeltree_child` in $O(\lg \lg \sigma)$ on any labeled tree.

We extend the concept to multi-labeled trees, where each node can have more than one label. We combine our results on multi-labeled sequences and labeled trees to represent and access multi-labeled trees: the two resulting representations use $2n + t(\lg \sigma + o(\lg \sigma))$ bits (see Cor. 2 and 3) and support the label-based operators in the same time as in labeled trees, where t is the total number of labels assigned.

Adaptive algorithms are algorithms which, among instances of the same size, perform better on "easier" instances, where the easiness has to be defined for each problem. Kirkpatrik and Seidel [23] were the first to point the interest of such algorithms, by giving an algorithm to compute the convex-hull of a set of points, whose complexity is expressed as a function of the size of the convex-hull rather than of the size of the set. Adaptive algorithms for sorting were studied as functions of many distinct measures of difficulty, which Estivill and Castro summarized in a survey [16].

Closer to our applications, Demaine *et al.* [14] studied adaptive algorithms for the union, intersection and difference of sets represented by sorted arrays, motivated by applications in posting lists for search engines. Their measure of difficulty is defined as the cost of encoding a certificate of the result of the instance, and they proved that their algorithms are optimal in the comparison model with respect to this measure of difficulty. Barbay and Kenyon [5] defined another measure of difficulty for the intersection problem, denoted δ, based on the number of steps required by a non-deterministic algorithm to check the answer. They proved the optimality of their deterministic algorithm with respect to this measure of difficulty, as compared to randomized algorithms in the comparison model. Barbay [4] proved that randomized algorithms perform better than deterministic ones on the intersection problem, by providing a finer measure of difficulty, denoted ρ, such that any randomized algorithm optimal for ρ is optimal for δ, but no deterministic algorithm could be optimal for δ.

We show that the algorithm from Barbay and Kenyon [5] can be adapted to use our data-structure for binary relations to answer a query composed of k labels from $[\sigma]$ in time $O(\delta k \lg \lg \sigma)$, with the same definition of δ (Th. 4). Denoting by t the number of relations associated with any label of the query, this corresponds to an improvement in complexity from $O(\delta k \lg(t/\delta k))$ to $O(\delta k \lg \lg \sigma)$, which is tremendous on most instances.

We also adapt those results to the search in multi-labeled trees. We define the answer to a *Path* query Q, constituted of k labels, as the set of nodes x for which the trace of labels in a rooted path contains the k labels of Q. For this type of queries, only the first occurence of a label on a branch matters, hence a multi-labeled tree can be encoded as a non-recursive multi-labeled tree without changing the result of the query, by removing the redundant labels and removing the nodes left without labels. We extend the adaptive algorithm from Barbay and Kenyon [5] to solve a Path queries on non-recursive multi-labeled trees with the same time performance (Th. 5) than queries on binary relations.

The paper is organized as follows. In the next Section, we present our succinct data structures for the three objects considered: binary relations in Section 2.1, labeled trees in Section 2.2 and multi-labeled trees in Section 2.3. The encoding of binary relations is independant from the encoding of labeled trees, and both are combined to encode multi-labeled trees. We describe in Section 3 the algorithms using those data structures to search the objects efficiently: the adaptive algorithm for the intersection using our encoding of binary relations in Section 3.1, and our new adaptive algorithm for searching non-recursive multi-labeled trees in Section 3.2. We conclude in Section 4 with some perspectives for future work.

2 Succinct Indexes

2.1 Binary relations

Consider a sequence of n objects, an ordered set of σ labels and r pairs from $[n] \times [\sigma]$, forming a binary relation R.

In the context where objects are references to web-pages, and labels are keywords associated to the web-pages, such relations are used as an index to answer conjunctive queries through the intersection of sets of references [4, 5, 14, 15]. Such indexes support the operators `string_rank`(a, x), `string_select`(a, r) and `string_access`(x)[4], and are traditionally encoded as a set of *postings lists*, which associates a sorted list of objects to each label. This encoding takes $t \lg n + \sigma \lg t$ bits, and supports `string_select` and `string_access` in constant time, but `string_rank` requires a time logarithmic in the number of objects associated to label a.

Each posting list can also be represented by a binary string of lenght n, and encoded using Clark and Munro's encoding to support the operators `string_rank` and `string_select` on it in constant time, but this representation uses $\sigma n + o(\sigma n)$ bits, which is too much in practice, especially when the number of pairs t is much smaller than σn.

We propose a succinct encoding that takes asymptotic minimal space and supports operators on both dimension of the binary relation. Let a be a label from $[\sigma]$, x be an object from $[n]$, and r be an integer. We defi ne the following operators:
- `label_rank`(a, x), the number of objects labeled a preceding x;
- `label_select`(a, r), the r-th object labeled a if any, or ∞ otherwise;
- `object_rank`(x, a), the number of labels associated with object x preceding label a;
- `object_select`(a, r), the r-th label associate with object x if any, or ∞ otherwise;
- `table_access`(x, a), check whether object x is associated with label a.

The operators `label_rank` and `label_select` are mere extenstions of the operators `string_rank` and `string_select`, and are supported in the same time, the only difference with the work of G. *et al.* being that each object can be associated to several labels. The new operators are `object_rank`, `object_select`, and `table_access`: their meaning is specifi c to binary relations, and their implementation require different techniques than `label_rank` and `label_select`.

Theorem 1. *Consider a set of objects $[n]$ and an object x, a set of labels $[\sigma]$ and a label a, an integer r and a binary relation formed by t pairs from $[n] \times [\sigma]$. There are two encodings, each using $t(\lg \sigma + o(\lg \sigma))$ bits that support the defined operators with the following run-times:*

	`select`	`access`
`label_rank`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`label_select`(a, r)	$O(1)$	$O(\lg \lg \sigma)$
`object_rank`(x, a)	$O\big((\lg \lg \sigma)^2\big)$	$O(\lg \lg \sigma)$
`object_select`(x, r)	$O(\lg \lg \sigma)$	$O(1)$
`table_access`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma)$

Proof (sketch). First, we reduce the problem to the encoding of matrices of size $\sigma \times \sigma$ using the same technique than G. *et al* [19]. This reduce the problem to support operators `bin_rank` and `bin_select` on both dimensions of the binary relation, which we call `row_select`(i, j), `column_select`(i, j), `row_rank`(i, j) and `column_rank`(i, j).

[4] As defined in the Introduction.

We represent a boolean matrix M of size $\sigma \times \sigma$ by two strings: COLUMNS, of alphabet $[\sigma]$ and of lenght t, such that the k-th symbol of COLUMNS corresponds to the column index of the k-th pair in the row-major order[5] traversal of M; and ROWS, a binary string of lenght $\sigma + t$, such that the k-th one of ROWS signals which symbol of COLUMNS corresponds to the last pair of its row. For instance, the strings COLUMNS= "$1, 3, 2, 3, 1, 2, 3$" and ROWS= "$0, 0, 1, 0, 0, 1, 0, 0, 1$" would correspond to the binary relation defined by the pairs $\{(1,1),(1,3),(2,2),(2,3),(3,1),(3,2),(3,3)\}$.

The operators $\texttt{column\_select}(i,j)$ and $\texttt{column\_rank}(i,j)$ are based on searching for occurences of symbol j in the string COLUMNS, which is done through the string_rank and string_select operators defined by G. *et al* [19]. The operator $\texttt{row\_select}(i,j)$ corresponds to a string_select operator on a substring of COLUMNS. The operator $\texttt{row\_rank}(i,j)$ is the most complicated to support. It is reduced to a binary search over $\lg \sigma$ elements, where each comparison costs $\lg \lg \sigma$. As such a binary search performs $\lg \lg \sigma$ comparisons, the operator is supported in time $O\big((\lg \lg \sigma)^2\big)$. The operator $\texttt{table\_access}(i,j)$ is reduced to the difference between $\texttt{row\_rank}(i,j)$ and $\texttt{row\_rank}(i,j+1)$, or equivalently to the difference between $\texttt{column\_rank}(i,j)$ and $\texttt{column\_rank}(i+1,j)$, hence its uniform cost in both encodings. □

The operators $\texttt{label\_nb}(a)$, the number of objects with a label a; and $\texttt{object\_nb}(x)$, the number of labels associated to object x; correspond to particular cases of the data-structure from G. *et al*: they are supported in constant time.

The space used by the above data structure is optimal (asymptotically equal to the information theoretical minimum) under the assumption that the average number of ones per column is quite small, namely if $r/n = \sigma^{o(1)}$: It uses $t(\lg \sigma + o(\lg \sigma))$ bits when the lower bound suggested by information theory is

$$\lg \binom{n\sigma}{t} = t\big(\lg(n\sigma) - \lg t + O(1)\big) = t\big(\lg \sigma - o(\lg \sigma)\big).$$

2.2 Labeled Trees

Consider a set of σ labels, and an ordinal tree of n nodes such that each node is assigned a label: this is a *labeled tree* [17, 18]. Let a be a label from $[\sigma]$, and x be a node from $[n]$. We define the following operators on labeled trees:

- $\texttt{labeltree\_desc}(a,x)$, the first descendant of x labeled a in some order, or ∞ if there is none;
- $\texttt{labeltree\_nbdesc}(a,x)$, the number of descendants of x associated to label a;
- $\texttt{labeltree\_anc}(a,x)$, the first ancestor of x labeled a in some order, or ∞ if there is none;
- $\texttt{labeltree\_child}(a,x)$, the first child of x labeled a in the natural order.
- $\texttt{labeltree\_parent}(a,x)$, the parent of x labeled a, or ∞ if there is none;

The structure proposed by Geary *et al.* [18] partitions the tree in smaller trees, and supports both the navigation operators and the label-based operators in constant time. But as small trees on large alphabets are taking a lot of space, the whole structure requires $2n + n\big(\lg \sigma + O(\sigma \lg \lg n / \lg n)\big)$ bits, which is much more than the lower bound of $2n + n(\lg \sigma - o(\lg \sigma))$ suggested by information theory.

The structure proposed by Ferragina *et al.* [17] is based on a different concept: it codes the structure of the tree and of the labels separately. Their structure supports in constant time the operators labeltree_parent, and labeltree_child, but it does not support the operators labeltree_anc or labeltree_desc, and it uses $2n \lg \sigma + O(n)$ bits, which is twice more than what is necessary.

As Ferragina *et al.*, we encode the structure of the tree separately from the labels, but we encode it as the trace of the pre-order traversal of the tree, and we encode the structure of the tree using Geary *et al.*'s encoding for unlabeled trees.

[5] The *Row-Major* order lists the elements from the first row, then from the second one, and so on.

Theorem 2. *Consider a labeled tree of n nodes and σ labels. Let a be a label from $[\sigma]$, x be a node from $[n]$, d be the depth of x, n_a be the number of descendants of x which are labeled a, and r be the number of pre-order predecessors of x which are labeled a. There are two encodings, each using $2n+n(\lg\sigma+o(\lg\sigma))$ bits, which supports:*
- the non-labeled operators supported by Geary et al.*'s data-structure in constant time;*
- the labeled operators of Theorem 1 on the pre-order traversal of the labels of the tree;
- the following operators, in the time indicated below, for the pre-order rank:

	`select`	`access`
`labeltree_parent`(a,x)	$O(\lg\lg\sigma)$	$O(1)$
`labeltree_desc`(a,x)	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma\lg\lg\lg\sigma)$
`labeltree_nbdesc`(a,x)	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma\lg\lg\lg\sigma)$
`labeltree_child`(a,x)	$O(n_a\lg\lg\sigma)$	$O(\lg\lg\sigma\lg\lg\lg\sigma+n_a)$
`labeltree_anc`(a,x)	$O(\min(\lg\lg\sigma+r, d\lg\lg\sigma))$	$O(\min(r\lg\lg\sigma, \lg\lg\sigma\lg\lg\lg\sigma+d))$

Proof (sketch). We represent a labeled tree of n nodes and σ labels in two parts: a string `LABELS` encoding the labels of the noded listed in pre-order (using our structure from Theorem 1 or the encoding of G. *et al.* [19]), and an encoding `TREE` encoding the structure of the tree (using the encoding from Geary *et al.* [18]). In addition to the usual navigation operators, the operators on `TREE` provide a bijection between its nodes and their pre-order rank, i.e. the position of their label in `LABELS`.

The operator `labeltree_parent`(a,x) corresponds trivially to a test of the label of the parent. The operators `labeltree_desc`(a,x) and `labeltree_nbdesc`(a,x) are just using the fact that the labels of all descendants of x are consecutive in the string encoding the labels of the tree. The first descendant of x corresponds to the first symbol a in `LABELS` after the position corresponding to x; and the number of such descendants corresponds to the number of symbols a in a substring of `LABELS`, both easily obtained using the operators `string_rank` and `string_select` on `LABELS`. The operators `labeltree_child`(a,x) and `labeltree_anc`(a,x) both correspond to some naive algorithms listing all children or all descendants of label a or all ancestors of x labeled a: they are not supported efficiently. □

The operator `labeltree_anc` is efficiently supported in the particular case where for each label a, each node has at most one ancestor associated with a, as then the labeled ancestor of a node is unique and can be found efficiently.

Definition 1. *A* Non-Recursive Labeled Tree *is a labeled tree where for each label a and each leaf x of the tree, x has at most one ancestor of label a.*

Corollary 1. *Consider a non-recursive labeled tree of n nodes and σ labels. Let be a a label from $[\sigma]$, and x a node from $[n]$. The encoding described in Theorem 2 supports* `labeltree_anc`(a,x) *in time $O(\lg\lg\sigma)$ for the encoding* `select` *and in time $O(\lg\lg\sigma\lg\lg\lg\sigma)$ for the encoding* `access`*.*

Proof (sketch). The last node y labeled a preceding x in the pre-order is the only possile candidate for ancestor of x that is labeled a: any node that comes in between y and x in the pre-order is a descendant of y, and can be labeled a if and only only x and y have no common ancestor labeled a.

We can find y in time $O(\lg\lg\sigma)$ using the operator `string_rank`, and check in constant time whether it is an ancestor of x, using the `tree_depth` and `tree_leveled_ancestor` operators from Geary *et al.*. □

This restriction to non-recursive labeled trees is more practical than it looks: Zhang *et al.* [31] measured the number of nodes sharing a label and a branch in several XML benchmarks and found that it was very low. In particular, for any label a, the documents from DBLP, XMark10 and XMark100 have at most two nodes labeled a in any rooted path. All based-operators are supported in the times described in Theorem 2 and

Corollary 1 on those documents, even though they are not strictly non-recursive labeled trees: it is sufficient to double the size of the alphabet and the time needed by the operators.

In some application it might be necessary to support both `labeltree_child` and `labeltree_desc` operators. The order in which the labels are stored, which keeps consecutive the descendants of a node and allows efficient `labeltree_desc` operators in the structure of Theorem 2. Ferragina *et al.* store the labels in an order which keeps consecutive the children of a node, to allow efficient labeled `labeltree_child` operators, but as this order does not keep the descendants consecutive, they can not support `labeltree_desc` operators.

Listing the labels of the tree in DFUDS[6] [6] order, the labels of a subtree of root x are encoded as one isolated character, corresponding to the label of x, and one substring which lists the children of x followed by its other descendants. Replacing the pre-order by the DFUDS order in the structure of the Theorem 2 yields better results for the complexity of the operator `labeltree_child`:

Theorem 3. *Consider a labeled tree of n nodes and σ labels. Let a be a label from $[\sigma]$, x be a node from $[n]$, and d be the depth of x. There are two encodings, each using $2n + n(\lg \sigma + o(\lg \sigma))$ bits, which supports:*
- the non-labeled operators supported by Benoit et al. *[6]'s data-structure in constant time;*
- the labeled operators of Theorem 1 on the DFUDS traversal of the labels of the tree;
- the following operators in the time indicated below, for the DFUDS order:

	`select`	`access`
`labeltree_parent`(a, x)	$O(\lg \lg \sigma)$	$O(1)$
`labeltree_nbchildren`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_child`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_nbdesc`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_desc`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_anc`(a, x)	$O(d \lg \lg \sigma)$	$O(d)$

Proof (sketch). As before, the operator `labeltree_parent`(a, x) is trivial. The operators `labeltree_nbdesc`(a, x) and `labeltree_desc`(a, x) are solved as before, because the labels of all the descendants are consecutive in the string encoding the labels. The operator `labeltree_child`(a, x) is solved in a similar way to `labeltree_desc`(a, x) because the labels of the children are also consecutive in the string. But the operator `labeltree_anc`(a, x) cannot be computed efficiently, even in the particular case of non-recursive labeled trees: it is computed by scanning all the ancestors of x. □

The information theoretic lower bound for storing a labeled tree on n nodes with σ labels is asymptotically $2n + n(\lg \sigma - o(\lg \sigma))$, hence our encodings, which use $2n + n(\lg \sigma + o(\lg \sigma))$ bits, are asymptotically optimal when σ is large.

2.3 Multi-Labeled Trees

File systems can be seen as tree-structured documents, but the XML [30] model is too restrictive to represent them, as several keywords can be associated to each folder and file. We consider an extension of labeled trees to represent and search efficiently the indexes of file systems.

Definition 2 (multi-labeled trees and non-recursive multi-labeled trees). *Consider an ordinal tree of n nodes, a set of σ labels, and a set of t pairs from $[n] \times [\sigma]$: we call it a* multi-labeled tree, *and we extend to it the operators on labeled trees. As for labeled trees, we study the particular case where for each label a and each leaf x of the tree, x has at most one ancestor of label a: we call such a multi-labeled tree a* non-recursive multi-labeled tree.

[6] For Depth First Unary Degree Sequence. Siblings are stored consecutively, followed recursively by their subtrees in left to right order.

We will see in Section 3.2 that the restriction from multi-labeled trees to non-recursive multi-labeled trees is not a problem for pratical queries on file-systems. The results on binary relations from Section 2.1 combine very easily with the results on labeled trees from Section 2.2 to give efficient encodings and operators on non-recursive multi-labeled trees:

Corollary 2. *Consider a non-recursive multi-labeled tree of n nodes and σ labels, associated in t pairs. Let a be a label from $[\sigma]$, x be a node from $[n]$, and n_a be the number of descendants of x which are labeled a. There are two encodings, each using $2n + t(\lg \sigma + o(\lg \sigma))$ bits, which supports the same operators than the encodings of Theorem 2 and Corrolary 1, and in the same time:*
- the non-labeled operators supported by Geary et al.*'s data-structure in constant time;*
- the labeled operators of Theorem 1 on the pre-order traversal of the labels of the tree;
- the following operators in the time indicated below:

	`select`	`access`
`labeltree_parent`(a, x)	$O(\lg \lg \sigma)$	$O(1)$
`labeltree_desc`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_nbdesc`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_anc`(a, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`labeltree_child`(a, x)	$O(n_a \lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma + n_a)$

Proof. The encoding for labeled trees described in Theorem 2 and Corrolary 1 supports the same operators using only the operators `string_rank`, `string_select` and `string_access`, which are directly extended to the operators `label_rank`, `label_select` and `table_access`, and supported in the same times. The operators supported on labeled trees are extended to multi-labeled trees by replacing each operator defined on strings by its equivalent on binary relations. □

Corollary 3. *Consider a multi-labeled tree of n nodes and σ labels, associated in t pairs. Let a be a label from $[\sigma]$, x and y be some nodes from $[n]$, and d be the depth of x. There are two encodings, each using $2n+t(\lg \sigma+o(\lg \sigma))$ bits, which supports the same operators, in the same time, than described in Theorem 3.*

As in Section 2.1, the space used by our structure is optimal under the assumption $r/n = \sigma^{o(1)}$.

3 Applications

3.1 Efficient Posting Lists

Several algorithms have been proposed for computing the union [9, 12, 13, 14, 20, 21, 24], intersection [3, 4, 5, 14, 15] or difference [14] of ordered sets. These algorithms assume that the sets are represented as sorted arrays and perform searches on arrays, such as binary search, doubling search, or interpolation search, with a worst case complexity logarithmic in the size of the array. Using our structure for binary relations described in Section 2.1 improves the performance of those algorithms:

Theorem 4. *Consider a set of objects $[n]$, a set of labels $[\sigma]$, associated in r pairs from $[n] \times [\sigma]$, and a set Q of k from $[\sigma]$. There is a deterministic algorithm wolving the conjunctive query formed by Q in time $O(\delta k \lg \lg \sigma)$, where δ is the minimum number of operations performed by any non-deterministic algorithm to check the result of Q.*

Proof (sketch). Barbay and Kenyon proposed a deterministic algorithm for the conjunctive query that uses $O(\delta\sigma)$ doubling searches [5, Th. 3.3]. We replace the doubling search by `label_rank` operator, and the result follows. □

As the algorithm using doubling searches runs in time $O(k\delta \lg(t/\delta k))$, the improvement factor is $\lg(n/k\delta)/\lg \lg \sigma$, which is very large when n is much larger than σ.

3.2 File System Search

We already argued that the XML model was ill-adapted for indexing file-systems, and we introduced multi-labeled trees as a better alternative. The search in file-systems (and hence in multi-labeled trees) is an important application, and the tools [1, 2, 8, 25] used to search in XML documents can be extended to multi-labeled trees. But the structural queries [7, 11] on XML documents are not adequate for the search in file systems, as their structure is too heavy for the user and as some natural queries are very hard to express.

We introduce a new type of query to search in labeled and multi-labeled trees, which corresponds to one of the most natural search query that one can perform in a file-system.

Definition 3 (Path Query). *Given a non-recursive multi-labeled tree and a set Q of k labels, find the set of nodes x, such that:*

1. *the rooted path to x contains nodes matching all the labels from Q;*
2. *and this path contains no node satisfying (1) other than x.*

The query has a simple syntax, however its description in XPath is very long, as it must describe all possible orders of labels. For this type of queries, only the first occurence of a label on a branch matters: a multi-labeled tree can be encoded as a non-recursive multi-labeled tree without changing the result of the query, by removing the redundant labels and removing the nodes left without labels.

Such queries are motivated by the search in file systems, where the result corresponds to folders or files whose path match the set of keywords. Multi-labeled trees associate several keywords to each folder or file (such as the words and extension composing its name) in an index of the file-system. Using techniques similar to those used for the intersection problem, we prove the following results:

Theorem 5. *Consider a non-recursive multi-labeled tree of n objects and σ labels, associated in t pairs. Given a path query composed of k labels, there is an algorithm solving it which performs $O(\delta k)$ operations in time $O(\delta k \lg\lg \sigma)$, where δ is the minimum number of operation performed by a non-deterministic algorithm to solve the query.*

Proof (sketch). Consider the following algorithm:

Set x to the first node labeled a_1; Initialize `R` to the empty set, `YES` and s to one;
while x belongs to the tree **do**
  Set s to the next label in cyclic order;
  if x has an ancestor labeled a **then**
    Increment `YES`;
  else if x has a descendant labeled a **then**
    Set x to this descendant, and increment `YES`;
  else
    Set x to the next node labeld a in pre-order, and reset `YES` to one;
  end if
  if `YES` $= k$ **then**
    Add x to `R`;
    Set x to the next node labeld a in pre-order, and reset `YES` to one;
  end if
end while
return `R`;

This algorithm cycles through the labels indexed by s, maintains in x the lowest node of the current potential match, counts in `YES` how many labels are currently matched, and eventually adds x to the list `R` when `YES` $= k$.

The pre-order rank of successive nodes pointed to by x strictly increases at each update, so that

– at any time, all pre-order predecessors of x have been considered and added to L if necessary;
– every k iterations of the loop the algorithm considered at least as many nodes than a non-deterministic algorithm would have in a single operation.

When the pre-order rank of x reaches its final value, all nodes have been considered (hence the correctness), and the algorithm performed $2\delta k$ operations where a non-deterministic algorithm would have performed at least δ (hence the complexity result). □

Unless the operators defined in Section 2.1 can be encoded more efficiently, we prove that this result is optimal for deterministic algorithms:

Lemma 1. *Consider any deterministic algorithm Alg solving path queries, and $\delta \geq 1$, $k \geq 2$, $n \geq \delta(2k+1)+1$, $\sigma \geq 2k+1$, and $t \geq n$. There is a random distribution $\mathcal{D}$ on non-recursive multi-labeled trees of $O(n)$ objects and $O(\sigma)$ labels, associated in $O(t)$ pairs, and a path query composed of k labels which can be solved by a non-deterministic algorithm in at most $O(\delta)$ operations on any non-recursive multi-labeled tree from $\mathcal{D}$, such that Alg performs $O(\delta k \lg\lg\sigma)$ operations on average on $\mathcal{D}$.*

Proof (sketch). We define a distribution $\mathcal{D}$ on non-recursive multi-labeled trees such as each tree has $O(n)$, $O(\sigma)$, associated in $O(t)$ pairs in δ branches of $2k+1$ nodes; and such that any non-deterministic algorithm can show in δ operations that the query composed of labels $\{1,\ldots,k\}$ has no match. As no deterministic algorithm can check that this query has no match in less than δk operations on average, this proves our lower bound. □

The result on deterministic algorithms from Lemma 1 is combines trivially with the Yao-von Neumann principle [27, 28, 29] to prove a lower bound on the complexity of any randomized algorithms:

Theorem 6. *Consider any randomized algorithm $RandAlg$ solving path queries, and $\delta \geq 1$, $n \geq \delta(2k+1)+1$, $k \geq 2$, $\sigma \geq 2k+1$, and $t \geq n$. There is a non-recursive multi-labeled tree of $O(n)$ objects and $O(\sigma)$ labels, associated in $O(t)$ pairs, and a path query composed of k labels which can be solved by a non-deterministic algorithm in at most $O(\delta)$ operations, such that $RandAlg$ performs on average $O(\delta k \lg\lg\sigma)$ operations.*

Proof. This is a simple application of Lemma 1 and of the Yao-von Neumann principle [27, 28, 29]:
– Lemma 1 gives a distribution on which any deterministic algorithm performs poorly on average.
– The Yao-von Neumann principle permits to deduce from this distribution a lower bound on the worst case complexity of randomized algorithms. □

The proof of those results is similar to their counterpart on the intersection problem [5]. In particular, Theorems 5 and 6 show that a deterministic algorithm performs as well as any randomized algorithm for Path queries.

4 Conclusion

In this paper, we consider succinct data steructures for binary relations, labeled trees and multi-labeled trees. We propose two encodings for each, supporting efficiently the basic operators on it, and we use those operators to efficiently solve conjunctive queries on binary relations and Path queries on labeled and multi-labeled treees.

Given a sequence of n objects, a set of σ labels, associated f t pairs from $[n] \times [\sigma]$, we give two representations using asymptotically optimal space and efficiently supporting in different time trade-offs the operators `label_rank`(a,x), `label_select`(a,r), `label_access`(x), `label_nb`(x), `object_rank`(x,a),

`object_select`(x, r), `object_access`(a), and `object_nb`(a), where x is an object, a is a label, and r is an integer.

We give two new representations for labeled trees, which we combine with multi-labeled sequences to represent multi-labeled trees, each being defined as an ordinal tree of n nodes, a set of σ labels, and t pairs from $[n] \times [\sigma]$: a labeled tree is a particular case of multi-labeled tree and our results apply to it with $t{=}n$. Both our representations for multi-labeled trees use $2n + t(\lg \sigma + o(\lg \sigma))$ bits and support efficiently the operators `labeltree_parent`(a, x), `labeltree_nbdesc`(a, x), `labeltree_desc`(a, x). As they do not support the operators `labeltree_anc`(a, x) and `labeltree_child`(a, x) efficiently, we show that the operator `labeltree_anc`(a, x) is supported in $O(\lg \lg \sigma)$ in the special case of "Non-Recursive Multi-Labeled Trees", where no node has two ancestors with the same label. We also define two encodings based on a different technique, which use the same space, support efficiently `labeltree_child`(a, x) in addition to the other operators, but can not support `labeltree_anc`(a, x) efficiently, even on non-recursive multi-labeled trees.

We describe how the complexity of some intersection algorithm can be improved by a huge factor using our data-structure to replace traditional sorted arrays. We describe how the concept of non-recursive multi-labeled tree fits the indexing and searching in file systems through unstructured queries, and we give an adaptive algorithm solving those queries in time $O(\delta k \lg \lg \sigma)$ using our first encoding for a non-recursive multi-labeled tree of n nodes and σ labels, associated in t pairs.

The structures defined in [19] allowed us to efficiently extend succinct encodings for unlabeled trees to the multi-labeled case, so it is only natural to consider applying the same techniques to extend other unlabeled structures, such as graphs or relational data of dimension more than two, to labeled versions. The bounds can, undoubtly, be extended to be phrased in terms of entropy. While our structures use an amount of space asymptotically optimal, the time improvements they provided are significant, and allow us to solve queries in time much closer to a constant factor of the time needed by a non-deterministic algorithm. It would be nice to consider the lower bounds associated with queries for structures working on limited space.

Bibliography

[1] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proceedings of the Twelfth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 547–556, 2001.

[2] S. Alstrup and T. Rauhe. Improved labeling scheme for ancestor queries. In *Proceedings of the Thirteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–953, 2002.

[3] R. A. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusöz, editors, *CPM*, volume 3109 of *Lecture Notes in Computer Science*, pages 400–408. Springer, 2004.

[4] J. Barbay. Optimality of randomized algorithms for the intersection problem. In A. Albrecht, editor, *Proceedings of the Symposium on Stochastic Algorithms, Foundations and Applications (SAGA 2003), in Lecture Notes in Computer Science*, volume 2827 / 2003, pages 26–38. Springer-Verlag Heidelberg, November 2003.

[5] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the thirteenth ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 390–399, 2002.

[6] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. *Algorithmica*, to appear.

[7] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simon. Xml path language (xpath) 2.0. Technical report, W3C Working Draft, November 2003. http://www.w3.org/TR/xpath20/.

[8] P. Buneman, M. Grohe, and C. Koch. Path queries on compressed xml. In *VLDB*, 2003.

[9] C. Christen. Improving the bound on optimal merging. In *Proceedings of 19th FOCS*, pages 259–266, 1978.

[10] D. R. Clark and J. I. Munro. Effi cient suffi x trees on secondary storage. In *SODA '96: Proceedings of the Seventh annual ACM-SIAM Symposium on Discrete Algorithms*, pages 383–391, Philadelphia, PA, USA, 1996.

[11] J. Clark and S. DeRose. Xml path language (xpath). Technical report, W3C Recommendation, November 1999. http://www.w3.org/TR/xpath/.

[12] W. F. de la Vega, A. M. Frieze, and M. Santha. Average-case analysis of the merging algorithm of hwang and lin. *Algorithmica*, 22(4):483–489, 1998.

[13] W. F. de la Vega, S. Kannan, and M. Santha. Two probabilistic results on merging. *SIAM J. Comput.*, 22(2):261–271, 1993.

[14] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the* 11^{th} *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[15] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Experiments on adaptive set intersections for text retrieval systems. In *Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments, Lecture Notes in Computer Science*, pages 5–6, Washington DC, January 2001.

[16] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[17] 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 '05)*, Oct. 2005.

[18] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *SODA '04: Proceedings of the Fifteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10. Society for Industrial and Applied Mathematics, 2004.

[19] A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of SODA'06*, 2006.

[20] F. K. Hwang and S. Lin. Optimal merging of 2 elements with n elements. *Acta Informatica*, pages 145–158, 1971.

[21] F. K. Hwang and S. Lin. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM Journal of Computing*, 1(1):31–39, 1972.

[22] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 549–554, 1989.

[23] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 1986. 15(1):287–299.

[24] G. K. Manacher. Significant improvements to the hwang-ling merging algorithm. *JACM*, 26(3):434–440, 1979.

[25] J.-K. Min, M.-J. Park, and C.-W. Chung. Xpress: a queriable compression for xml data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 122–133. ACM Press, 2003.

[26] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001.

[27] J. V. Neumann and O. Morgenstern. *Theory of games and economic behavior*. 1st ed. Princeton University Press, 1944.

[28] M. Sion. On general minimax theorems. *Pacic Journal of Mathematics*, pages 171–176, 1958.

[29] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

[30] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (xml) 1.0 (third edition). Technical report, W3C Recommendation, February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.

[31] N. Zhang, M. T. Özsu, A. Aboulnaga, and I. F. Ilyas. XSEED: Accurate and Fast Cardinality Estimation for XPath Queries. In *to appear Proc. 22nd Int. Conf. on Data Engineering (ICDE) 2006*, 2006.