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

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

[1] David R. Cheriton School of Computer Science
University of Waterloo, Canada.
[2] Computational Logic and Algorithms group
IT University of Copenhagen, Denmark.

Abstract. The most heavily used methods to answer conjunctive queries on binary relations (such as the one associating keywords with web pages) are based on inverted lists stored in sorted arrays and use variants of binary search. We show that a succinct representation of the binary relation permits much better results, while using space within a lower order term of the optimal. We apply our results not only to conjunctive queries on binary relations, but also to queries on semi-structured documents such as XML documents or file-system indexes, using a variant of an adaptive algorithm used to solve conjunctive queries on binary relations.
Keywords: conjunctive queries, intersection problem, succinct data structures, labeled trees, multi-labeled trees.

1 Introduction

Consider the task of a search engine answering conjunctive queries: given a set of keywords, it must return a list of references to the objects relevant to all those keywords. These objects can be web-pages as in the case of a web search engine like Google, or documents as in a search engine on a file system, or any kind of data searched by keywords in general. Rather than roam the set of all objects (which is usually huge — think about the set of web-pages indexed by Google), a good search engine uses a *precomputed index*, which represents the binary relation between the space of objects $\{1, \ldots, n\} = [n]$ and the space of admissible keywords $\{1, \ldots, \sigma\} = [\sigma]$, so that it can be easily searched.

Usually, such an index is coded as a set of sorted arrays, so that the answer to conjunctive queries is the intersection of those arrays. This intersection can then be computed in time linear in the sum of the sizes of the array, but several adaptive algorithms have been studied for the easier case where a small number of comparisons permits to check the result, with much better results than linear [2, 3, 5, 6]. These intersection algorithms are all based on variants of the binary search algorithm: as the cost of a search is logarithmic in the size of the array, this impacts on their complexity, in particular on "easy" instances where the intersection is empty or where only a few comparisons are sufficient to check the result of the intersection.

Our results are threefold:

- First, observing that the use of inverted lists in sorted arrays is far from being a mandatory step to compute the intersection, we consider instead succinct data structures to encode the binary relation, which also permits much faster searches. We give two representations (Theorem 1) for binary relations associating n objects with σ labels in t pairs from $[n]\times[\sigma]$. Each of these representations uses $t\big(\lg\sigma + o(\lg\sigma)\big)$ bits, and supports queries in time $\mathcal{O}(\lg\lg\sigma)$ or better (depending on the operator and on the encoding), thus generalizing the results from Golynski *et al.* [9] on strings on large alphabets. These results can be directly applied to the intersection problem (Theorem 3), to improve the time complexity of the algorithm from Barbay and Kenyon [3], and thus to reduce the time required to answer a conjunctive query.
- Second, observing that a labeled tree is simply a tree in which each node is associated with a label through a binary relation, we give a representation for labeled trees (Theorem 2). This uses $n\big(\lg\sigma + o(\lg\sigma)\big)$ bits and supports both structure-based navigation operators in constant time and label-based search operators in time $\mathcal{O}(\lg\lg\sigma)$ or better, improving on the space used by the solutions from both Geary *et al.* [8] and Ferragina *et al.* [7] on labeled trees. These results can be immediately generalized to multi-labeled trees (such as XML documents or file-system indexes) where nodes are associated with zero or more labels in t pairs (rather than only n pairs in labeled trees), giving a representation (Corollary 1) which uses $t\big(\lg\sigma + o(\lg\sigma)\big)$ bits and supports the same operators in the same time.
- Third, observing the similarity between conjunctive queries and *unordered path-subset* queries on labeled and multi-labeled trees, we prove tight upper (Theorem 4) and lower (Theorem 5) bounds on the complexity of any randomized algorithm solving these queries, hence extending the results of Barbay and Kenyon on the intersection problem [3] to unordered path-subset queries on multi-labeled trees.

All our results concerning the running time of operators and algorithms are expressed in the RAM model, where words of size $\Theta(\lg(\max\{n,\sigma\}))$ can be accessed and processed in constant time.

The paper is organized as follows. In Section 2, 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 and the encoding of labeled trees are combined to encode multi-labeled trees. We describe in Section 3 the algorithms that search the objects efficiently using those data structures: the adaptive algorithm for the intersection using our encoding of binary relations in Section 3.1, and our new adaptive algorithm for searching multi-labeled trees in Section 3.2. We conclude in Section 4 with some perspectives on future work.

2 Succinct Indexes

2.1 Binary relations

Consider a binary relation R between an ordered set of n objects and an ordered set of σ labels. Let t denote the cardinality of R, i.e. the number of pairs (object, label) that are in R. In the context in which objects are references to web-pages, and labels are keywords associated with the web-pages, such relations are used to answer conjunctive queries, i.e. for a given set of keywords, to return all pages that are associated with all the keywords in the set. Typically, such a relation is encoded as a collection of *postings lists*, in which each list associates a sorted list of web pages (objects) to a keyword (label), which can be intersected [2, 3, 5, 6] to answer conjunctive queries.

Let α be a label from $[\sigma]$, x be an object from $[n]$, and r be an integer. We propose a succinct encoding of the relation R that takes asymptotically minimal space and supports the following operators:

– `label_rank`(α, x), the number of objects labeled α preceding x;
– `label_select`(α, r), the r-th object labeled α, if any, or ∞ otherwise;
– `label_nb`(α), the number of objects labeled α;
– `object_rank`(x, α), the number of labels associated with object x preceding label α;
– `object_select`(x, r), the r-th label associated with object x, if any, or ∞ otherwise;
– `object_nb`(x), the number of labels associated with object x;
– `table_access`(x, α), checks whether object x is associated with label α.

The naive encoding of such lists as sorted arrays uses $t \lg n + \sigma \lg t$ bits of space and supports `label_select`(α, r) in constant time, but `label_rank`(α, x) requires time logarithmic in the number of objects associated with label α. It is not clear how to support `object_rank`(x, α) and `object_select`(x, r) with such an encoding. Each posting list can also be represented by a binary string of length n, and encoded using Clark and Munro's [4] encoding to support the operators `label_rank` and `label_select` in constant time. However, this representation uses a total of $\sigma n + o(\sigma n)$ bits, which is too much in practice, especially when the number of pairs t is much smaller than σn.

The operators `label_rank` and `label_select` are extensions of the operators `string_rank` and `string_select` defined by Golynski *et al.* [9], who only considered the case of strings, or in other words, the case where each object (i.e. position in a string) is associated with exactly one label (i.e. a character from an alphabet of size σ, that occurs at the given position in the string). We support the `label_rank` and `label_select` operators in the same time as theirs. The operators `object_rank`, `object_select` are extensions of `string_access`: `string_access`(x) gives the label associated with x (i.e., the character at position x), the operators `object_rank` and `object_select` are used to navigate in the set of labels that are associated with a given object. The techniques from Golynski *et al.* are not directly applicable to the case of binary relations, however we use similar ideas and obtain an efficient implementation of the new operators `object_rank`, `object_select`, `label_nb`, `object_nb` and `table_access`. In what follows, we use two encodings described by Golynski *et al.*: `select` encoding and `access` encoding, and extend them to binary relations.

Theorem 1. *Consider a binary relation on $[n] \times [\sigma]$ of cardinality t. Assume that each object is associated with at least one label and each label is associated with at last one object. Then there are two encodings (named* `label` *encoding and* `object` *encoding), each using $t\big(\lg\sigma + o(\lg\sigma)\big)$ bits, that support the defined operators with the following run-times:*

	`label`	`object`
`label_rank`(α, x)	$\mathcal{O}(\lg\lg\sigma)$	$\mathcal{O}(\lg\lg\sigma \lg\lg\lg\sigma)$
`label_select`(α, r)	$\mathcal{O}(1)$	$\mathcal{O}(\lg\lg\sigma)$
`label_nb`(α)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`object_rank`(x, α)	$\mathcal{O}\big((\lg\lg\sigma)^2\big)$	$\mathcal{O}(\lg\lg\sigma)$
`object_select`(x, r)	$\mathcal{O}(\lg\lg\sigma)$	$\mathcal{O}(1)$
`object_nb`(x)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`table_access`(α, x)	$\mathcal{O}(\lg\lg\sigma)$	$\mathcal{O}(\lg\lg\sigma)$

where $x \in [n]$, $\alpha \in [\sigma]$, and r is a positive integer.

Proof (sketch). Without loss of generality, we assume that $\sigma \leq n$: the construction is similar in the symmetric case. We reduce the problem of encoding a binary matrix of size $\sigma \times n$ to the encoding of n/σ matrices of size $\sigma \times \sigma$ each, using the same technique as Golynski *et al* [9]: we call this step a *domain reduction*. Let t_M denote the number of ones in one of the smaller matrix M, and let the operators `row_rank`, `row_select`, `column_rank` and `column_select` have the same functionalities as the operators `label_rank`, `label_select`, `object_rank` and `object_select` respectively, but restricted to the smaller matrices, e.g. `row_rank`(i, j) is defined only for $j \leq \sigma$. This reduction allows the implementation of the operators `label_rank`, `label_select`, `object_rank`, `object_select` using the operators `row_rank`, `row_select`, `column_rank`, `column_select` with an acceptable space and time overhead.

We represent a boolean matrix M of size $\sigma \times \sigma$ by two strings: `COLUMNS`, on alphabet $[\sigma]$ and of length t_M, such that the k-th symbol of `COLUMNS` corresponds to the column index of the k-th pair in the row-major order[3] traversal of M; and `ROWS`, a binary string of length $t_M + \sigma$, such that the number of zeros between the i-th and the $i+1$-st one indicates how many ones are in the i-th row of M. We say that `COLUMNS` is divided into σ *parts* by `ROWS`. See the following example:

$$M = \begin{pmatrix} 0\,1\,0\,0 \\ 1\,1\,1\,0 \\ 1\,0\,0\,1 \\ 0\,1\,0\,0 \end{pmatrix} \qquad \begin{array}{ll} \texttt{COLUMNS} & = 2,\quad 1, 2, 3,\quad 1, 4,\quad 2 \\ \texttt{ROWS} & = 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1 \end{array}$$

[3] *row-major* order lists the elements from the first row, then from the second row, and so on.

We encode `COLUMNS` using one of the two encodings from Golynski *et al* [9] depending on the preferred time tradeoffs between different operators as mentioned in the statement of the theorem. These encodings use $t_M(\lg\sigma + o(\lg\sigma))$ bits of space with the following time tradeoffs:

	`select` encoding	`access` encoding
`string_access`	$\mathcal{O}(\lg\lg\sigma)$	$\mathcal{O}(1)$
`string_select`	$\mathcal{O}(1)$	$\mathcal{O}(\lg\lg\sigma)$
`string_rank`	$\mathcal{O}(\lg\lg\sigma)$	$\mathcal{O}(\lg\lg\sigma\lg\lg\lg\sigma)$

The vector `ROWS` can be encoded using any succinct fully indexable dictionary that supports in constant time the operators `bin_rank` and `bin_select`, the rank and select operators on binary strings introduced by Jacobson [10] and improved by Clark and Munro [4]. The operators `column_select`(i, j) and `column_rank`(i, j) are based on searching for occurrences of symbol j in the string `COLUMNS`, which is done through the `string_rank` and `string_select` operators on `COLUMNS` and `bin_rank` and `bin_select` operators on `ROWS`. The operator `row_select`(i, j) corresponds to a call to the `string_select` operator on the i-th part of `COLUMNS`.

A naive implementation of the operator `row_rank`(i, j) using a binary search on the i-th part of `COLUMNS` takes $\mathcal{O}(\lg x \cdot$ complexity of `string_access`$)$ time, where $x \leq \sigma$ is the length of the i-th part of `COLUMNS`, which is not good enough. We use a sparsification idea similar to the one introduced by Golynski *et al* [9], fixing the parameter $z = \lg\sigma$ and encoding every z-th character of the i-th part of `COLUMNS` using a y-fast trie (as defined by Willard [13]). This structure supports the rank operator in the "sparsified" string Y in time $\mathcal{O}(\lg\lg\sigma)$ using $\mathcal{O}(x/z\sigma) = \mathcal{O}(x)$ bits (which is $\mathcal{O}(t)$ for all values of i together). Note that `row_rank`$(i, j) \in [z\,\mathtt{rank}_Y(j), z\,(\mathtt{rank}_Y(j) + 1)]$, where $\mathtt{rank}_Y$ is the *set rank*, which denotes how many elements in Y are smaller than j. The result of `row_rank`(i, j) can be computed using a binary search in an interval of size $\lg\sigma$ in time $\mathcal{O}(\lg\lg\sigma \cdot$ complexity of `string_access`$)$.

The operator `label_nb`(α) can be done in constant time using `ROWS`. The operator `object_nb`(x) can also be done in constant time by maintaining an additional bit vector similar to `ROWS` that counts the numbers of occurrences for each column. The operator `table_access`(i, j) can be computed either as the difference between `row_rank`$(i, j + 1)$ and `row_rank`(i, j), or equivalently as the difference between `column_rank`$(i + 1, j)$ and `column_rank`(i, j).

The encoding of `COLUMNS` uses $t(\lg\sigma + o(\lg\sigma))$ bits (summed over all M). The encodings of y-fast tries and `ROWS` vectors use $\mathcal{O}(t + n)$ bits in total, hence the total space of $t\big(\lg\sigma + o(\lg\sigma)\big)$ bits for each encoding. □

Note, that the operators described above are "symmetrical" with respect to interchanging roles of objects and labels, so that we can assume that $n \geq \sigma$. The space used by the above data structure is almost optimal (equal to the information-theoretical minimum plus a lower order term) under the assumption that the average number of ones per column is small, namely if $t/n = \sigma^{o(1)}$. In this case the lower bound suggested by information theory, equal to $\lg\binom{n\sigma}{t}$, is roughly $t\big(\lg(n\sigma) - \lg t + \mathcal{O}(1)\big) = t\big(\lg\sigma - o(\lg\sigma)\big)$, which is close to the space used by our encodings, $t(\lg\sigma + o(\lg\sigma))$.

2.2 Labeled Trees

An *ordinal tree* is a rooted tree in which the children of a node are ordered and specified by their rank. Geary *et al.* [8] proposed an encoding for ordinal trees which supports in constant time the following operators, called *navigation operators*:

– `tree_ancestor`(x, i), the i-th ancestor of node x for $i \geq 0$;
– `tree_rank`$_{pre/post}(x)$, the position of node x in the *pre* or *post* order traversal of the tree;
– `tree_select`$_{pre/post}(r)$, the r-th node in the *pre* or *post* order traversal of the tree;
– `tree_child`(x, i), the i-th child of node x for $i \geq 1$;
– `tree_child_rank`(x), the number of left siblings of node x;
– `tree_depth`(x), the depth of x (number of edges in the path from root to x);
– `tree_nbdesc`(x), the number of descendants of x;
– `tree_deg`(x), the degree of x, i.e. its number of children.

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* [7,8]. Let α be a label from $[\sigma]$ and x be a node from $[n]$. We define the following operators on labeled trees, for the pre-order traversal of the tree:

– `labeltree_desc`(α, x), the first descendant of x which is labeled α, or ∞ if there is none;
– `labeltree_nbdesc`(α, x), the number of descendants of x that are labeled α;
– `labeltree_anc`(α, x), the ancestor of x which is labeled α and closest to the root, or ∞ if there is none;

In a manner similar to Ferragina *et al.* [7], 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 [8] encoding for unlabeled trees.

Theorem 2. *Consider a labeled tree of n nodes and σ labels. There is an encoding using $n\big(\lg\sigma + o(\lg\sigma)\big)$ bits that supports in constant time the navigation operators on the structure of the tree and in time $\mathcal{O}(\lg\lg\sigma)$ the operators* `labeltree_anc`, `labeltree_desc` *and* `labeltree_nbdesc` *along with the operators* `string_rank`, `string_select` *and* `string_access` *on the pre-order traversal of the labels of the tree.*

Proof (sketch). Represent the structure of the tree as an ordinal tree encoded using the encoding for unlabeled ordinal trees defined by Geary *et al.* [8]: this takes $2n+o(n)$ bits, and supports the navigation operators on the tree structure in constant time.

The labels are extended by one bit (i.e. to an alphabet of size 2σ) such that any node x originally labeled α is now labeled:

- α_* if x has no ancestor labeled α (but eventually some descendants);
- α_*^+ if x has at least one ancestor labeled α.

The sequence of extended labels is encoded in pre-order, using the representation of Golynski *et al.* [9] which uses $n(\lg(2\sigma)+o(\lg(2\sigma))) = n(\lg\sigma+o(\lg\sigma))$ bits and supports the operators `string_access`, `string_select` and `string_rank` on the pre-order traversal of the labels of the tree in the times claimed.

The operator `labeltree_anc`(α, x) is supported by checking for the last node y labeled α_* in pre-order before x, which takes time $\mathcal{O}(\lg\lg\sigma)$, and checking that y is an ancestor of x, which takes constant time. The symmetric operator `labeltree_desc`(α, x) is supported by checking for the first node y labeled α_* or α_*^+ in pre-order after x, which takes time $\mathcal{O}(\lg\lg\sigma)$, and checking that y is a descendant of x, which takes constant time. The operator `labeltree_nbdesc`(α, x) is easily supported via a combination of calls to the navigation operators, and two calls to the operator `string_rank`. Overall, the encoding uses $2n + o(n) + n(\lg\sigma + o(\lg\sigma)) = n\big(\lg\sigma + o(\lg\sigma)\big)$ bits. $\square$

The information-theoretic lower bound for storing a labeled tree on n nodes with σ labels is asymptotically $n\big(\lg\sigma - o(\lg\sigma)\big)$. Hence our encoding, which uses $n\big(\lg\sigma+o(\lg\sigma)\big)$ bits, differ from this bound by a lower order term in σ. Note that other encodings with similar results can be obtained using the other encodings proposed by Golynski *et al.*; we developed here only the most appropriate for our specific application.

2.3 Multi-Labeled Trees

XML documents and file systems can be seen as tree-structured documents, but the labeled tree model described in the previous section is too restrictive to represent them, as several labels are associated with each leaf in XML documents, and several labels are associated with each internal node (folder) or leaf (file) in a file system. We consider an extension of labeled trees where any number of labels can be associated with each node.

Definition 1. *A* **multi-labeled tree** *is an ordinal tree on n nodes together with a set of σ labels, and a set of t pairs from $[n] \times [\sigma]$. The operators are the same as those on labeled trees: structure-based navigation operators (as defined by Geary* et al. *[8]) and label-based operators (as defined in Theorem 2).*

The results on binary relations from Section 2.1 combine very easily with the results on labeled trees from Section 2.2 to give an encoding supporting efficiently the operators on multi-labeled trees:

Corollary 1. *Consider a multi-labeled tree on n nodes and σ labels, associated in t pairs. There is an encoding using $t\big(\lg\sigma + o(\lg\sigma)\big)$ bits and supporting the same operators as the encoding of Theorem 2 and in the same time.*

Proof. The operators supported on labeled trees are extended to multi-labeled trees by replacing each operator defined on strings [9] by its equivalent on binary relations as defined in Theorem 1, in the first encoding (named `label`), which supports all operators in time $\mathcal{O}(\lg\lg\sigma)$ or better. □

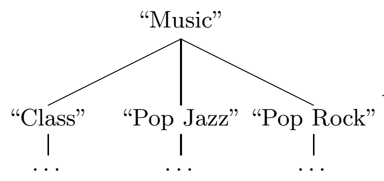

Fig. 1. A simplistic example of File System.

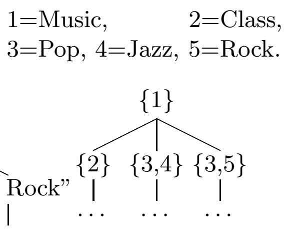

Fig. 2. The corresponding Multi-Labeled Tree.

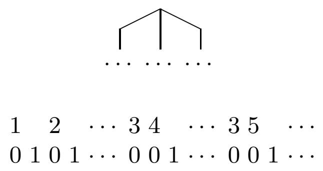

Fig. 3. The corresponding succinct encoding.

Figure 1 represents a simplistic view of a personal file system organizing music files. Figure 2 shows its representation as a multi-labeled tree, where the text associated with each node is replaced by numbers from the range $[1, \sigma]$. Figure 3 shows the succinct encoding of this multi-labeled tree: the structure of the ordinal tree, the string representing the labels in pre-order, and a binary string where ones separate sequences of zeroes encoding the number of labels associated to a node. As in Section 2.1, the space used by our structure is optimal under the assumption that $t/n = \sigma^{o(1)}$.

3 Applications

3.1 Efficient Posting Lists

Several algorithms have been proposed for computing the answer to conjunctive queries on a binary relation, through the intersection of inverted lists in sorted arrays. The intersection of sorted arrays has been studied from several points of view, all of which are based on various search methods in sorted arrays: Several people have studied the intersection of a pair of sorted arrays, Baeza-Yates [1] being the most recent. Other efforts have been considering the intersection of a larger number of sorted arrays [2, 3, 5, 6], measuring the performance of the algorithms relative to the complexity of the description of a *certificate* of the intersection, such as the set of comparisons performed by a non-deterministic algorithm to check the result of the instance. We refer the reader to Figure 4 for a simple example, and to the cited papers for more details.

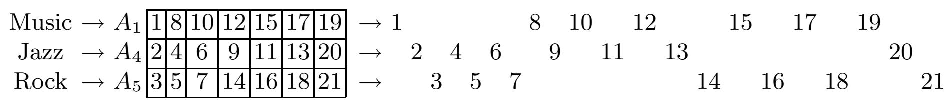
Music → A_1 1 8 10 12 15 17 19 → 1 8 10 12 15 17 19
Jazz → A_4 2 4 6 9 11 13 20 → 2 4 6 9 11 13 20
Rock → A_5 3 5 7 14 16 18 21 → 3 5 7 14 16 18 21

Fig. 4. An example of how a conjunctive query corresponds to the intersection of sets. A non-deterministic algorithm can check that the intersection is empty in $\delta = 4$ comparisons ($1 < 2, 7 < 8, 13 < 14, 19 < 20$). Barbay and Kenyon's algorithm performs $8 < \delta k$ searches ($1 < 2 < 3 < 8 < 9 < 14 < 15 < 20 < 21$). Most intersection algorithms use variants of binary search in the sorted array. We propose to use the rank operator on a succinct encoding of the binary relation.

These search methods are limited to a complexity logarithmic in the size of the array. But the use of inverted lists in sorted arrays is far from being a mandatory step to computing the intersection. Our implementation for binary relations described in Section 2.1 permits us to search faster in the list of references associated with an object, and hence improves the performance of intersection algorithms.

Theorem 3. *Consider a set of objects $[n]$ and a set of labels $[\sigma]$, associated in t pairs from $[n] \times [\sigma]$, and a conjunctive query Q composed of k labels from $[\sigma]$. There is a deterministic algorithm solving Q in time $\mathcal{O}(\delta k \lg\lg \sigma)$, 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 [3, Theorem 3.3] proposed a deterministic algorithm for the conjunctive query that uses $\mathcal{O}(\delta k)$ doubling searches. We replace the doubling search by a combination of `label_rank`, `label_select` and `label_access` operators, and the result follows. Suppose that x is initialized as the first object of $[n]$, and α as the first label of the query. If we introduce the bogus object ∞, which matches all labels and is a successor to all objects, the algorithm now goes as follows:

```
1. If $x = \infty$, exit;
2. If $k$ labels are matched, output $x$, set it to the next object matching $\alpha$, and go to 1;
   Otherwise, set $\alpha$ to the next label from $Q$, in cyclic order;
3. If $x$ has matches $\alpha$, go to 2;
   Otherwise, set $x$ to the next object matching $\alpha$, and go to 1.
```
□

3.2 File System Search

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

Definition 2 (Unordered Path-Subset Query). *Given a 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; and,*
2. *this path contains no node satisfying* (1) *other than x.*

Such queries are motivated by the search in file systems, where the result corresponds to folders or files whose path matches the set of keywords. Multi-labeled trees associate several keywords with 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 result:

Theorem 4. *Consider a Multi-Labeled Tree of n nodes and σ labels, associated by t pairs. Given an unordered path-subset query composed of k labels, there is an algorithm solving it which performs $\mathcal{O}(\delta k)$ operator calls in time $\mathcal{O}(\delta k \lg\lg\sigma)$, where δ is the minimum number of operation performed by a non-deterministic algorithm to solve the query.*

Proof (sketch). Suppose that x is initialized to the root of the tree and that α is initialized to the first label of the query. If we consider the nodes in pre-order, and introduce the bogus node ∞ that matches all labels and is a successor to all nodes, our algorithm proceeds as follow:

1. **If** $x = \infty$, exit;
2. **If** k labels are matched, output x, set it to the next node matching α, and go to 1; **Otherwise**, set α to the next label from Q in cyclic order;
3. **If** x has an ancestor labeled α, go to 2;
4. **If** x has a descendant labeled α, set it to the first such descendant, and go to 2; **Otherwise**, set x to the next node matching α, and go to 1.

This algorithm cycles through the labels in the query set, maintains in x the lowest node of the current potential match, counts how many labels are currently matched, and eventually outputs the nodes matching the query.

The pre-order rank of successive nodes pointed to by x is strictly increasing at each update, so that at any time, all pre-order predecessors of x have been considered and have been output when adequate. Every k iterations of the loop the algorithm considered at least as many nodes as a non-deterministic algorithm would have in a single operation: it takes at most k steps to eliminate as many potential result nodes as a non-deterministic algorithm, which can "guess" which operation to perform to eliminate the largest number of potential result nodes.

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

We now prove that the number of operator calls performed by the above algorithm is optimal for deterministic algorithms:

Lemma 1. *Consider any deterministic algorithm Alg solving unordered path-subset 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 multi-labeled trees of $\mathcal{O}(n)$ objects and $\mathcal{O}(\sigma)$ labels, associated with $\mathcal{O}(t)$ pairs, and an unordered path-subset query composed of k labels which can be solved by a non-deterministic algorithm in at most $\mathcal{O}(\delta)$ operations on any multi-labeled trees from $\mathcal{D}$, such that Alg performs $\Omega(\delta k)$ operator calls on average to solve instances from $\mathcal{D}$.*

Proof (sketch). We define a distribution $\mathcal{D}$ on multi-labeled trees with δ branches of $2k + 1$ nodes such that any non-deterministic algorithm can show that the unordered path-subset query composed of labels $\{1, \ldots, k\}$ has no match in δ operations. We prove the lower bound by showing that no deterministic algorithm can check that this query has no match in less than δk operations on average. $\square$

The result on deterministic algorithms from Lemma 1 combines trivially with the Yao-von Neumann principle [11, 12, 14] to prove a lower bound on the complexity of any randomized algorithm:

Theorem 5. *Consider any randomized algorithm RandAlg solving unordered path-subset 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 Multi-Labeled tree of $\mathcal{O}(n)$ nodes and $\mathcal{O}(\sigma)$ labels, associated in $\mathcal{O}(t)$ pairs, and an unordered path-subset query composed of k labels which can be solved by a non-deterministic algorithm in at most $\mathcal{O}(\delta)$ operations, such that RandAlg performs on average $\Omega(\delta k)$ operator calls to answer the query.*

The proofs of these results is similar to their counterparts on the intersection problem [3]. In particular, Theorems 4 and 5 show that a deterministic algorithm performs as well as any randomized algorithm for unordered path-subset queries, in terms of the number of operator calls.

4 Conclusion

We considered succinct data structures for binary relations, labeled trees and multi-labeled trees, and their application to search algorithms in those structures. Our results are threefold:

– first, we give two succinct encodings for binary relations using asymptotically optimal space and efficiently supporting in different time trade-offs the rank and select operators on the rows and columns of the relation;
– second, we give a new representation for labeled trees, that we combine with binary relations to represent multi-labeled trees;
– Third, we show that those encodings have applications to conjunctive queries in binary relations and unordered path-subset queries in multi-labeled trees, such as XML documents or file-system indexes.

Obvious research prospects are to extend the range of operators supported (e.g. labeled child queries), and to apply similar encodings to other types of queries (e.g. ordered sub-path, Twig Pattern and XPath queries).

References

1. R. A. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 3109 of *LNCS*, pages 400–408. Springer, 2004.
2. 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)*, volume 2827 / 2003, pages 26–38. Springer-Verlag Heidelberg, 2003.
3. J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 390–399, 2002.
4. D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 7th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 383–391, Philadelphia, PA, USA, 1996.
5. E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.
6. 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.
7. 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)*, pages 184–196, 2005.
8. R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10. Society for Industrial and Applied Mathematics, 2004.
9. A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 368–373, 2006.
10. 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.
11. J. V. Neumann and O. Morgenstern. *Theory of games and economic behavior.* 1st ed. Princeton University Press, 1944.
12. M. Sion. On general minimax theorems. *Pacific Journal of Mathematics*, pages 171–176, 1958.
13. D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, Aug. 1983.
14. 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.