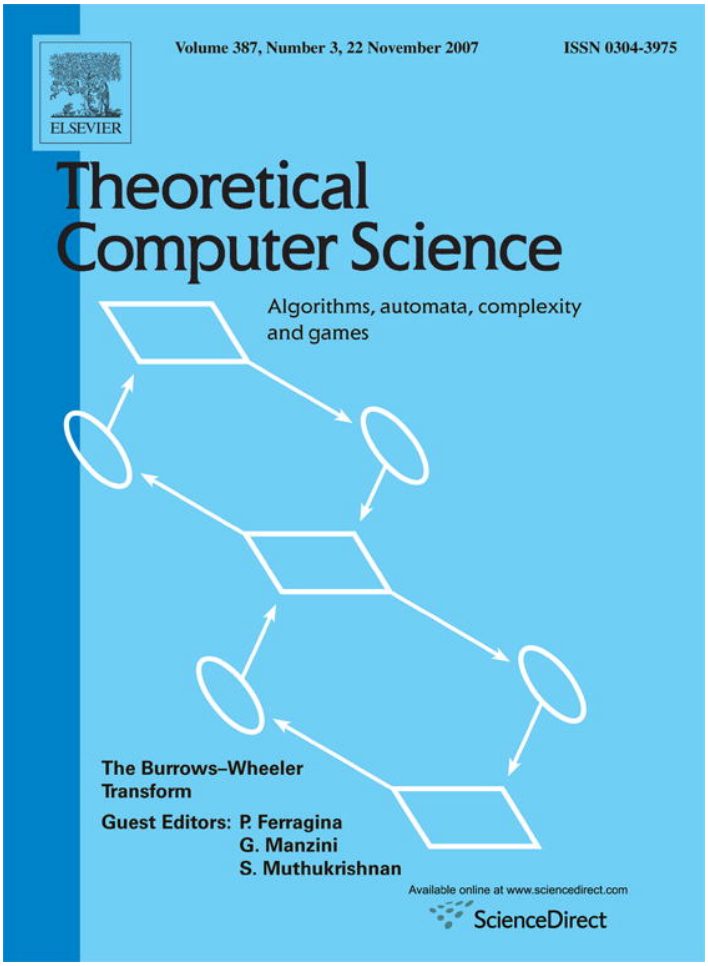


Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 387 (2007) 284–297
Theoretical Computer Science
www.elsevier.com/locate/tcs

Adaptive searching in succinctly encoded binary relations and tree-structured documents☆

Jérémy Barbay[a,*], Alexander Golynski[a], J. Ian Munro[a], S. Srinivasa Rao[b]

[a] *David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada*
[b] *Computational Logic and Algorithms Group, IT University of Copenhagen, 2300 Copenhagen S., Denmark*

Abstract

The methods most heavily used by search engines to answer conjunctive queries on binary relations (such as one associating keywords with web-pages) are based on computing the intersection of postings lists stored as sorted arrays and using variants of binary search. We show that a succinct representation of the binary relation permits much better results, while using less space than traditional methods. 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: Adaptive algorithms; Conjunctive queries; Intersection problem; Labeled trees; Multi-labeled trees; Path queries; Succinct data structures

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 such as Google, or documents as in the case of a search engine in a file system, or any other kind of data searched by keywords. Rather than scan the set of all objects, which is usually huge, a good search engine uses a *precomputed index* to represent the binary relation between the set of n objects and the set of σ admissible keywords.

Usually such an index is coded as a set of sorted arrays that are called *postings lists*, so that the answer to conjunctive queries is the intersection of the subsets corresponding to those arrays. This intersection can then be computed in time linear in the sum of the sizes of the arrays, but several adaptive algorithms have been studied for easier cases in which the result can be determined much more quickly. For example, if we are to intersect two sorted arrays whose values

☆ A preliminary version of this article appeared in the proceedings of the 17th annual Symposium on Pattern Matching (CPM 2006). The work was supported by grants from the Natural Science and Engineering Research Council of Canada and the Canada Research Chairs Program.
* Corresponding author.
E-mail addresses: jbarbay@uwaterloo.ca (J. Barbay), agolynski@uwaterloo.ca (A. Golynski), imunro@uwaterloo.ca (J. Ian Munro), ssrao@itu.dk (S. Srinivasa Rao).

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2007.07.015

are interleaved, linear time is clearly required. On the other hand, if all elements of one array fall between two values in the other, two comparisons suffice to demonstrate that the intersection is indeed empty. This admits the notion of describing the runtime as compared to the shortest possible proof of the correct answer in each particular instance rather than simply the worst case over all possible instances of the given size. Returning to our example, we note that comparison based methods would take time $\Theta(\lg n)$ to discover between which elements of the second array all the elements of the first array fit. Our results are motivated by the notion of improving on this "logarithmic style" bound for situations in which the basic objects come from a bounded range, e.g. identifiers referring to web-pages.

Our results are threefold:

- First, instead of encoding postings lists as sorted arrays, we develop a space efficient data structure that permits faster searches. We give a representation (Theorem 3.1) for binary relations associating n objects with σ labels in t pairs from $[n]\times[\sigma]$,[1] where n, σ and t are non-constant parameters. This representation uses $t\big(\lg\sigma+o(\lg\sigma)\big)$ bits,[2] and it generalizes the results from Golynski et al. [11] for strings on large alphabets. These results can be directly applied to conjunctive queries (Theorem 4.1), to improve the time complexity of the algorithm from Barbay and Kenyon [2], and thus to reduce the time required to answer a conjunctive query.
- Next, we give a representation of labeled trees (Theorem 3.3) that encodes the tree structure and the labels separately. This representation uses $n\big(\lg\sigma+o(\lg\sigma)\big)$ bits and supports structure-based navigation operators in constant time, and label-based search operators in time $O(\lg\lg\sigma)$, improving the space used by the solutions from both Geary et al. [10] and Ferragina et al. [9] for labeled trees, at the cost of increasing the time in which some operators are supported. These results can be immediately generalized to multi-labeled trees (such as XML documents or file-system indexes), so that the total number of (node, label) pairs is t, yielding a representation which uses $t\big(\lg\sigma+o(\lg\sigma)\big)$ bits and supports the same operators in the same time (Corollary 3.5).
- Our final results are motivated by performing searches in multilabeled trees (e.g. tree-structured file systems), in which each node is associated with labels (e.g. keywords). We introduce the concept of *path-subset queries*, answered by a description of all the nodes in the tree with ancestors matching a given set of labels. We prove tight upper (Theorem 4.4) and lower (Theorem 4.6) bounds on the complexity of any randomized algorithm solving these queries, thus generalizing the results of Barbay and Kenyon [2] from the intersection problem on arrays to the path-subset queries for labeled and multilabeled trees.

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

The paper is organized as follows. In the next section we describe related work on succinct encodings and adaptive algorithms, that we either use or improve upon. In Section 3, we present our data structures for the three data types considered: binary relations in Section 3.1, labeled and multi-labeled trees in Section 3.2. The encoding of binary relations is independent of the encoding of labeled trees, and both are combined to encode multi-labeled trees. Section 4 describes the algorithms that search efficiently those data structures: the adaptive algorithm for answering conjunctive queries using our encoding of binary relations in Section 4.1, and our new adaptive algorithm for searching multi-labeled trees in Section 4.2. We conclude in Section 5 with some perspectives on future work.

2. Related work

2.1. Succinct data structures

Succinct data structures were introduced by Jacobson [13], to encode bit vectors, (unlabeled) trees and planar graphs in space essentially equal to the information-theoretic lower bound, while supporting appropriate operators on them efficiently. For *bit vectors*, Jacobson defined two useful operators: given a bit vector $B[0,\ldots,n-1]$, a bit $\alpha\in\{0,1\}$, an object $x\in[n]$ and an integer $r\in\{1,\ldots,n\}$, the operator $\texttt{bin\_rank}_B(\alpha,x)$ returns the number of occurrences of α in $B[0,...,x]$, and the operator $\texttt{bin\_select}_B(\alpha,r)$ returns the position of the r-th label α in B. We omit the subscript B when it is clear from the context. Lemma 2.1 gives two ways to support those operators, in which part (a) is from Jacobson [13] and Clark and Munro [5], while part (b) is from Raman et al. [19].

[1] We use $[x]$ to denote the set $\{0,\ldots,x-1\}$.

[2] We use $\lg\sigma$ to denote $\log_2\sigma$, and the notation $o(\lg\sigma)$ infers that σ is not a constant.

Lemma 2.1 (*[4,13,19]*). *A bit vector B of length n with v 1s can be represented using either:* (a) $n + o(n)$ *bits, or* (b) $\lg \binom{n}{v} + O(n \lg\lg n / \lg n)$ *bits, to support the access to each bit, and the operators* `bin_rank` *and* `bin_select` *in constant time in the RAM model with word size* $\Theta(\lg n)$.

Grossi et al. [12] generalized this problem to alphabets of arbitrary size σ, extending the operators to `string_rank`(α, x), the number of occurrences of α before position x; `string_select`(α, r), the position of the r-th occurrence of α in the sequence, and `string_access`(x), the character at position x in the sequence. Their *wavelet tree* structure encodes a string of length n from an alphabet of size σ using $nH_0 + n\,o(\lg\sigma)$ bits, where H_0 is the 0-th order entropy of the given string, to support all three operators in $O(\lg\sigma)$ time. Golynski et al. [11] gave two different encodings supporting the same operators more efficiently. The following lemma describes their result (we name the two encodings as `select` encoding and `access` encoding):

Lemma 2.2 (*[11]*). *A sequence of n elements from an alphabet of size σ can be encoded using $n(\lg\sigma + o(\lg\sigma))$ bits to achieve the following time bounds for the operators:*

	`select` *encoding*	`access` *encoding*
`string_access`	$O(\lg\lg\sigma)$	$O(1)$
`string_select`	$O(1)$	$O(\lg\lg\sigma)$
`string_rank`	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma \lg\lg\lg\sigma)$

We extend the problem to the encoding of sequences of n objects, where each object can be associated with several labels through a binary relation of t pairs from $[n]\times[\sigma]$. Raman et al. [19] considered this problem as representing a set of dictionaries (multiple indexable dictionaries) by treating the set of labels associated with each object as a dictionary, and considered supporting rank and select operators on these sets. Here, we consider a different set of operators, which is suitable for our application. We give a representation (Theorem 3.1) which uses $t\big(\lg\sigma + o(\lg\sigma)\big)$ bits and supports the extended operators in the same time as them.

An *ordinal tree* is a rooted tree in which the children of a node are ordered and specified by their rank. Jacobson originally proposed a succinct data structure to store a tree of n nodes, that supports the operations parent and i-th child in $O(\lg n)$ bit probes. Other representations were proposed later: Munro and Raman [16] introduced the parenthesis representation; Benoit et al. [4] introduced the DFUDS (Depth-First Unary Degree Sequence) representation; and Geary et al. [10] proposed an encoding based on a recursive decomposition of the tree.

Each of these data structures uses $2n + o(n)$ bits which is close to the information-theoretic space lower bound of $2n - o(n)$ bits. These support various operators to navigate in the tree in constant time, on a Random Access Machine (RAM model) with word size $\Theta(\lg n)$, defined as follows:

Definition 2.3. The *navigation operators* on ordinal trees are defined as follows:

- `tree_ancestor`(x, i), the i-th ancestor of node x (x is its own 0-th ancestor);
- `tree_rank`$_{pre/post/dfuds}(x)$, the position of node x in the given tree-traversal;
- `tree_select`$_{pre/post/dfuds}(r)$, the r-th node in the given tree-traversal;
- `tree_child`(x, i), the i-th child of node x for $i \geq 1$;
- `tree_child_rank`(x), the number of siblings to the left of node x;
- `tree_depth`(x), the number of edges in the rooted path to x;
- `tree_nbdesc`(x), the number of descendants of x;
- `tree_deg`(x), the number of children of x.

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* [9,10]. Geary et al. [10] extended the navigation operators to consider labels from $[\sigma]$ associated with the nodes of the tree and to support those operators in constant time, but their data structure for label-based operators on labeled trees uses $n\big(\lg\sigma + O(\sigma \lg\lg\lg n / \lg\lg n)\big)$ bits, which is much more than the information-theoretic lower bound of $2n - o(n) + n\lg\sigma$ suggested by information theory when σ is large. Ferragina et al. [9] also consider data structures for labeled trees, but for a different set of operators. The structures they propose, however, either use $2n\lg\sigma + O(n)$ bits, which is roughly twice the minimum space required to encode the tree, or do not support efficiently the operators necessary for our application. We give a near optimal encoding (using space close to the information-theoretic minimum plus a lower order term) for labeled trees using $n\big(\lg\sigma + o(\lg\sigma)\big)$ bits (Theorem 3.3). Our encoding supports

the search for an α-ancestor or α-descendant of any node x in time $O(\lg\lg\sigma)$. We extend the concept to multi-labeled trees (Corollary 3.5), in which each node can have more than one label, and obtain similar results.

2.2. Adaptive algorithms

Adaptive algorithms are algorithms that take advantage of "easy" instances of the problem at hand, i.e. their runtime depends on some measure of difficulty, which could, for example, be a function of instance size and other parameters. For example, Kirkpatrick and Seidel [15] proposed an algorithm for computing the convex hull that has running time $O(n \lg h)$, where n is the number of input vertices, and h is the number of output vertices in the resulting convex hull. As previously known algorithms guarantee only a running time of $O(n \lg n)$ in the worst case, clearly, the adaptive algorithm performs better when the size of the convex hull size is small (e.g. a triangle). Another example is adaptive algorithms for sorting, which have been studied under various measures of difficulty. Estivill-Castro and Wood summarized many of these results in a survey [8].

Closer to our applications, Demaine et al. [6], motivated by the manipulation of postings lists in search engines, studied adaptive algorithms for the union, intersection and difference of sets represented by sorted arrays. Their measure of difficulty is defined as the cost of encoding a certificate (e.g. a proof) confirming the correctness of the result. They proved that their algorithm is optimal in the comparison-based model with respect to this measure of difficulty. Barbay and Kenyon [2] defined another measure of difficulty (denoted δ) for the intersection problem. This measure is based on the number of steps required by a non-deterministic algorithm to check the correctness of the answer. They proved that their deterministic algorithm is optimal in the class of randomized algorithms in the comparison-based model, with respect to this measure of difficulty.

Barbay and Kenyon's deterministic algorithm [2] answers a conjunctive query of k labels from $[\sigma]$ in time $O(\delta \sum_{i=1}^{k} \lg(n_i/\delta))$, where $n_1, n_2, \ldots, n_k$ are the sizes of the postings lists associated with each label of the query. We show that their algorithm can be adapted to use our data structure for binary relations in order to answer queries in time $O(\delta k \lg\lg\sigma)$ (Theorem 4.1) instead. We also extend their algorithm and analysis to path-subset queries on multi-labeled trees (Theorems 4.4 and 4.6).

3. Data structures

3.1. Binary relations

Consider a binary relation R between the ordered set $[n]$ of n objects, and the ordered set $[\sigma]$ of σ labels. Let t denote the cardinality of R, i.e. the number of pairs (object, label) 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.[3] Typically, such a relation is encoded as a collection of *postings lists*. Each postings list is a sorted list of the web-pages (objects) associated with (e.g. containing) a given keyword (label). A conjunctive query is then performed by intersecting the lists corresponding to the appropriate keywords [2,3,6,7].

Let α be a label from $[\sigma]$, x be an object from $[n]$, and $r \leq n$ be an integer. We consider the following operators on the relation R:

- $\texttt{label\_rank}_R(\alpha, x)$, the number of objects labeled α and preceding x;
- $\texttt{label\_select}_R(\alpha, r)$, the position of the r-th object labeled α if any, or ∞ otherwise;
- $\texttt{label\_nb}_R(\alpha)$, the number of objects with label α;
- $\texttt{object\_rank}_R(x, \alpha)$, the number of labels associated with object x and preceding label α;
- $\texttt{object\_select}_R(x, r)$, the r-th label associated with object x, if any, or ∞ otherwise;
- $\texttt{object\_nb}_R(x)$, the number of labels associated with object x;
- $\texttt{table\_access}_R(x, \alpha)$, checks whether object x is associated with label α.

[3] See the work of Demaine et al. [6] for a more detailed description.

We omit the subscript R when it is clear from the context. The information-theoretic lower bound to encode a binary relation on $[n] \times [\sigma]$ of cardinality t is $\lg \binom{n\sigma}{t}$, which is asymptotically equivalent to $t\big(\lg(n\sigma) - \lg t + O(1)\big)$. The naive encoding of such lists as sorted arrays uses $t\lceil \lg n \rceil + \sigma \lceil \lg t \rceil$ bits of space and supports the operators `label_select` and `label_nb` in constant time, but supports the operator `label_rank`(α, x) in time logarithmic in the number of objects associated with the label α. It is not clear how to support `object_rank`(x, α) and `object_select`(x, r) with such an encoding. Alternatively, each postings list can be represented using a bit vector to support the operators `label_rank` and `label_select` in constant time [5]. However, this representation uses a total of $\sigma n + o(\sigma n)$ bits, which is not optimal in the case where the number of pairs t is much smaller than σn. One can reduce the space complexity to $n \lg \sigma + O(\sigma n \lg \lg n / \lg n)$ bits using the fully indexable dictionaries of Raman et al. [19], but this space is still too large for most applications where σ is large.

Strings can be considered as binary relations where each object (i.e. a position in a string) is associated with a unique label (i.e. a character from an alphabet of size σ, that occurs at the given position in the string). The operators `label_rank` and `label_select` are extensions of the operators `string_rank` and `string_select` defined by Golynski et al. [11], 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). Our structures support the `label_rank` and `label_select` operators in the same time as theirs. The operators `object_rank` and `object_select` are extensions of `string_access`; while `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. Some intersection algorithms require the operator `label_nb` (e.g. `SmallAdaptive` [7]). The techniques from Golynski et al. are not directly applicable to the case of binary relations; however using similar ideas we obtain an efficient implementation of the operators `object_rank`, `object_select`, `label_nb`, `object_nb` and `table_access`.

In most applications, and in particular for conjunctive queries, it is reasonable to assume that each label and each object is used, i.e. that each object is associated with at least one label, and that each label is associated with at least one object, so that $t \geq \max\{n, \sigma\}$. This is because one can use the encoding of Lemma 2.1(a) to encode the sets of objects and labels effectively used, using bit vectors of size n and σ respectively. The space taken by this additional structure is at most linear in σ and n, so we can always assume that each label and object is used, and hence that $t \geq \max\{n, \sigma\}$.

Theorem 3.1. *Consider a binary relation R on $[n] \times [\sigma]$ of cardinality t with $t \geq \max\{n, \sigma\}$. 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` *encoding*	`object` *encoding*
`label_rank`(α, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$
`label_select`(α, r)	$O(1)$	$O(\lg \lg \sigma)$
`label_nb`(α)	$O(1)$	$O(1)$
`object_rank`(x, α)	$O(\lg \lg \sigma \lg \lg \lg \sigma)$	$O(\lg \lg \sigma)$
`object_select`(x, r)	$O(\lg \lg \sigma)$	$O(1)$
`object_nb`(x)	$O(1)$	$O(1)$
`table_access`(α, x)	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma)$

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

Proof. This proof is organized as follows: first we show how to reduce the problem of encoding a relation of size $\sigma \times n$ to the problem of encoding n/σ relations of size of $\sigma \times \sigma$; then we show how to encode each of these relations using a string on an alphabet of size σ that supports `string_rank`, `string_select`, and `string_access` operators; and then we complement this encoding with one additional data structure that would allow us to achieve the claimed run-time complexities. The first and the last steps of this proof are based on the techniques from Golynski et al. [11, proof of Theorem 2].

We associate the binary relation with a binary matrix where the labels correspond to rows, the objects correspond to columns, and object x is associated with label α when the entry of row α and column x is 1.

To encode this binary relation, we first assume that σ divides n: we show at the end of the proof that the additional space required when it is not the case is small. Divide the original binary matrix R into n/σ *chunks* $C_1, C_2, \ldots, C_{n/\sigma}$

of size $\sigma \times \sigma$ and each chunk into σ *blocks* of size $1 \times \sigma$. The cardinalities of the blocks are then stored in the binary vector X using a unary representation. Namely,

$$X = 1^{b_{00}}01^{b_{01}}\ldots01^{b_{0\,n/\sigma-1}}01^{b_{10}}01^{b_{11}}\ldots01^{b_{1\,n/\sigma-1}}0\ldots1^{b_{\sigma-1\,0}}01^{b_{\sigma-1\,1}}\ldots01^{b_{\sigma-1\,n/\sigma-1}}0$$

where b_{ij} is the cardinality of the i-th block of the j-th chunk (the integer b_{ij} is encoded by b_{ij} 1-bits followed by a 0-bit). We encode the binary vector X using the encoding (a) from Lemma 2.1, so that it supports the operators `bin_rank` and `bin_select` in constant time.

The operators `object_rank`, `object_select` and `object_nb` on R translate directly to the corresponding operators on the appropriate chunk. Formally, for $0 \leq i \leq n/\sigma$ and $0 \leq j < \sigma$ such that $x = i\sigma + j$, the operators are supported as follows:

$$\begin{aligned}
\texttt{object_rank}_R(x, \alpha) &= \texttt{object_rank}_{C_i}(j, \alpha)\\
\texttt{object_select}_R(x, \alpha) &= \texttt{object_select}_{C_i}(j, \alpha)\\
\texttt{object_nb}_R(x) &= \texttt{object_nb}_{C_i}(j).
\end{aligned}$$

The operators `label_rank`, `label_select` and `label_nb` on R are more complex, we detail their implementation one by one. We define for any label α and block number i the function $p(\alpha, i) = \texttt{bin\_select}_X(0, \alpha\sigma + i) + 1$ as the position in X at which the description of the α-th block of the i-th chunk starts, and for convenience we define $p(0, 0) = 1$. Then, the operator `label_rank` on R is reduced to the corresponding operator on the chunk as follows:

$$\texttt{label\_rank}_R(\alpha, x) = \texttt{label\_rank}_{C_i}(\alpha, j) + \texttt{bin\_rank}_X(1, p(\alpha, i)) - \texttt{bin\_rank}_X(1, p(\alpha, 0)).$$

Let $s = \texttt{bin\_select}_X(1, \texttt{bin\_rank}_X(1, p(\alpha, 0)) + r)$ be the position of the corresponding 1-bit in X (every 1-bit in R has the corresponding 1-bit in X). Then the operator `label_select` on R is easy to reduce to the corresponding operator on the chunk:

$$\texttt{label\_select}_R(\alpha, r) = \texttt{label\_select}_{C_i}(\alpha, j) + \sigma i$$

where $i = \texttt{bin\_rank}_X(0, s) - \alpha\sigma$ is the number of the chunk where this 1-bit occurs, and $j = r - (\texttt{bin\_rank}_X(1, p(\alpha, i)) - \texttt{bin\_rank}_X(1, p(\alpha, 0)))$ is the rank of this 1-bit inside the α-th block of the i-th chunk. Finally,

$$\texttt{label\_nb}_R(\alpha) = \texttt{bin\_rank}_X(1, p(\alpha + 1, 0)) - \texttt{bin\_rank}_X(1, p(\alpha, 0)).$$

Now we only need to implement the six operators on a given chunk C. Our approach is to encode all occurrences of 1-bits in C in the row- or column-major order [4] using a string on an alphabet of size σ, and then reduce our problem to the problem of supporting `string_rank`, `string_select`, and `string_access` on this string. Let us first describe the row-major order encoding. For each label in ascending order, we list all the objects associated with it (also in the ascending order), and denote the corresponding string by S. In addition to this string, we also maintain a binary vector P that allows us to navigate in S efficiently.

$$P = 1^{b_0}01^{b_1}\ldots01^{b_{\sigma-1}}0$$

where b_i is the cardinality of the i-th block of C. Note that this vector is analogous to X; however the cardinalities $b_{00}, b_{01}, \ldots, b_{\sigma-1\,n/\sigma-1}$ are stored in row-major order in X, and in column-major order in P. However for the case of the column-major order encoding of C, the vector X is just the concatenation of vectors P for each chunk, and hence we do not need to store P explicitly in this case. See the following example:

$$C = \begin{pmatrix} 0 & 1 & 0 & 0\\ 1 & 1 & 1 & 0\\ 1 & 0 & 0 & 1\\ 0 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{array}{lccccccccccccc} \mathrm{S} & = & 1, & & 0, & 1, & 2, & & 0, & 3, & & 1\\ \mathrm{P} & = & 1, & 0, & 1, & 1, & 1, & 0, & 1, & 1, & 0, & 1, & 0 \end{array}$$

[4] *row (column)-major* order lists the elements from the first row (column), then from the second row (column), and so on.

The bit $C[\alpha, x]$ is 1 if and only if the label α is associated with the object x. Denote by $t_C = b_0 + b_1 + \cdots + b_{\sigma-1}$ the length of S. Now we have a choice to encode S using one of the two encodings, `select` encoding or `access` encoding, mentioned in Lemma 2.2. Vector P is encoded using the encoding (a) from Lemma 2.1. The operators `object_rank` and `object_select` are implemented using `string_rank` and `string_select` respectively as follows:

$$\begin{aligned}\texttt{object\_rank}_C(x, \alpha) &= \texttt{string\_rank}_S(x, p(\alpha + 1))\\ \texttt{object\_select}_C(x, r) &= q(\texttt{string\_select}_S(x, r))\end{aligned}$$

where $q(r) = \texttt{bin\_rank}_P(0, \texttt{bin\_select}_P(1, r))$ is the block number corresponding to the position r of S, $p(\alpha) = \texttt{bin\_select}_P(0, \alpha) + 1$ is the position in P at which the description of the α-th block of C starts, and $p(0) = 1$. The operator `label_select` is implemented as follows:

$$\texttt{label\_select}_C(\alpha, r) = \texttt{string\_access}_S(\texttt{bin\_rank}_P(1, p(\alpha)) + r).$$

We can implement the operator $\texttt{label\_rank}(\alpha, x)$ using a binary search for x on $S_\alpha = S[p(\alpha) \ldots p(\alpha + 1) - 2]$, the part of S that corresponds to the label α (recall that S_α is an ascending sequence of objects). If we denote by l the length of S_α, the run-time complexity of the operator is $O(\lg l \cdot (\text{complexity of } \texttt{string\_access}))$, which can be much larger than the run-time of the other operators, since l can be as large as σ.

Instead, let us fix the parameter $z = \lg\sigma$ and let Y be the string obtained by taking every z-th character of the original string S_α. We encode the set of objects corresponding to the string Y using a y-fast trie (as defined by Willard [21]). This structure supports the rank operator on Y in time $O(\lg\lg\sigma)$ using $O(\frac{l}{z}\log\sigma) = O(l)$ bits (which is $O(t)$ for all blocks), since we are using the word size $\lg\sigma$. Note that $\texttt{label\_rank}(\alpha, x) \in [z\,\texttt{rank}_Y(j), z\,(\texttt{rank}_Y(j) + 1)]$, where $\texttt{rank}_Y$ is the *set rank*, which denotes how many elements in Y are smaller than j. The result of $\texttt{label\_rank}(i, j)$ can be computed using a binary search in an interval of size $\lg\sigma$ in time $O(\lg\lg\sigma \cdot (\text{complexity of } \texttt{string\_access}))$.

The operator $\texttt{label\_nb}(\alpha)$ can be implemented as follows

$$\texttt{label\_nb}(\alpha) = p(\alpha + 1) - p(\alpha) - 1.$$

The operator $\texttt{table\_access}(\alpha, x)$ can be implemented either as the difference between $\texttt{label\_rank}(\alpha, x + 1)$ and $\texttt{label\_rank}(\alpha, x)$, or as the difference between $\texttt{object\_rank}(x, \alpha + 1)$ and $\texttt{object\_rank}(x, \alpha)$.

Each chunk C can be encoded in four different ways: using the row or column major order, and then encoding the resulting string S using the `select` or `access` encoding. Regardless of the choice of the encoding, the operator `table_access` is supported in time $O(\lg\lg\sigma)$, and the operators `label_nb` and `object_nb` are supported in constant time. But each encoding gives a different combination of supporting times for the operators `label_rank`, `label_select`, `object_rank`, and `object_select`:

	`label_rank`	`label_select`	`object_rank`	`object_select`
row/select	$O\big((\lg\lg\sigma)^2\big)$	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma)$	$O(1)$
row/access	$O(\lg\lg\sigma)$	$O(1)$	$O(\lg\lg\sigma \lg\lg\lg\sigma)$	$O(\lg\lg\sigma)$
column/select	$O(\lg\lg\sigma)$	$O(1)$	$O\big((\lg\lg\sigma)^2\big)$	$O(\lg\lg\sigma)$
column/access	$O(\lg\lg\sigma \lg\lg\lg\sigma)$	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma)$	$O(1)$

As the column/select encoding is always worse than the row/access encoding and the row/select encoding is always worse than the column/access encoding, we can safely ignore them. We call the row/access encoding *label* encoding as it performs better for label operators, and similarly the column/access encoding is called *object* encoding.

The encoding of S using the structure of Golynski et al. [11] uses $t\,(\lg\sigma + o(\lg\sigma))$ bits (summed over all chunks). The encodings of y-fast tries and P vectors use $O(t + n)$ bits in total (summed over all chunks), and the bit vector X also uses $O(t + n)$ bits.

Now consider the case where σ does not divide n. We can pad the last chunk with extra columns so that it has the same size as other chunks. There is no extra space for such padding in the case where we use the row-major encoding. For the column-major encoding, in the bit vector P, we do not store the last $\sigma - (n \mod \sigma)$ 1-bits corresponding to the empty columns, but instead we just store the number of such columns, and use this number

when performing rank/select operations on P. The extra space for such padding is $O(\lg \sigma)$, hence the total space requirement is $t\big(\lg \sigma + o(\lg \sigma)\big)$ bits for each encoding. □

This encoding is reasonably close to the information-theoretic lower bound when $\sigma \leq n$, which is a reasonable assumption in the case of web search engines, where the number of web-pages indexed (objects) is much larger than the number of keywords indexed. There are some other applications where it might not be the case though: for instance in a typical HTML document, where each word of a paragraph can be indexed and where the tree structure is poor, the number of nodes (objects) can be smaller than the number of keywords indexed (labels). If this is the case, we can split our matrix into matrices of size $n \times n$ in the first step. Hence, we can show the following corollary:

Corollary 3.2. *Consider a binary relation on $[\sigma] \times [n]$ of cardinality t, and let $\mu = \min\{n, \sigma\}$. We can encode it using $t(\lg \mu + o(\lg \mu))$ bits in order to support the desired operators in the run-times shown below:*

	`label` *encoding*	`object` *encoding*
`label_rank`(α, x)	$O(\lg \lg \mu)$	$O(\lg \lg \mu \lg \lg \lg \mu)$
`label_select`(α, r)	$O(1)$	$O(\lg \lg \mu)$
`label_nb`(α)	$O(1)$	$O(1)$
`object_rank`(x, α)	$O(\lg \lg \mu \lg \lg \lg \mu)$	$O(\lg \lg \mu)$
`object_select`(x, r)	$O(\lg \lg \mu)$	$O(1)$
`object_nb`(x)	$O(1)$	$O(1)$
`table_access`(α, x)	$O(\lg \lg \mu)$	$O(\lg \lg \mu)$

In the rest of the paper, to simplify notation we assume that the number of labels is smaller than the number of objects ($\sigma \leq n$). The space used by our encoding, $t(\lg \sigma + o(\lg \sigma))$, is almost optimal (i.e. equal to the information-theoretical minimum plus a lower order term) under the assumption that the average number of labels associated with an object is small, that is $t/n = \sigma^{o(1)}$.

3.2. Labeled and multi-labeled trees

Given a labeled tree on n nodes with labels from the alphabet $[\sigma]$, we define the following operators for the preorder traversal of the tree, where α is a label from $[\sigma]$ and x is a node from $[n]$:

- `labeltree_desc`(α, x), the first α-descendant of x, or ∞ if there is none;
- `labeltree_nbdesc`(α, x), the number of α-descendants of x;
- `labeltree_anc`(α, x), the node closest to the root among the α-ancestors of x, or ∞ if there is none;
- `labeltree_succ`(α, x), the first α-node after x in preorder, or ∞ if there is none;

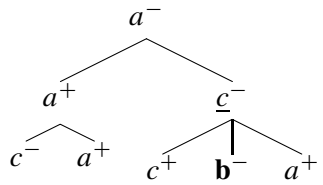
The three operators `labeltree_desc`, `labeltree_nbdesc`, and `labeltree_anc` were previously defined and supported by Geary et al. [10]. We introduce the operator `labeltree_succ` for our application.

The structure proposed by Geary et al. [10] partitions the tree into smaller trees, and supports both the navigation operators and the label-based operators in constant time. However, the structure requires $2n + n\big(\lg \sigma + O(\sigma \lg \lg \lg n / \lg \lg n)\big)$ bits, and this amount could substantially exceed the information-theoretic lower bound of $2n - o(n) + n \lg \sigma$ when σ is large.

Our data structure is inspired by the work of Ferragina et al. [9]. Their idea is to encode the structure of the tree and the labels separately to support a different set of operators. We use their idea to support the operators described by Geary et al. [10] instead, using a space close to the information-theoretic minimum space.

Theorem 3.3. *Consider a labeled tree of n nodes associated with labels from $[\sigma]$. There is an encoding using $n\big(\lg \sigma + o(\lg \sigma)\big)$ bits that supports the tree navigation operators in constant time, and the operators* `labeltree_anc`*,* `labeltree_desc`*,* `labeltree_nbdesc` *and* `labeltree_succ` *in $O(\lg \lg \sigma)$ time.*

Proof. We represent the tree structure using an encoding supporting the operators `tree_isanc`(x, y) and `tree_lastdesc`(x), and the operators rank and select on the preorder traversal of the tree (e.g. the encoding defined by Geary et al. [10], Jansson et al. [14] or Munro and Raman [16] combined with the results from Munro and Rao [17]). It uses $2n + o(n)$ bits and supports the tree navigation operators in constant time. In order to support the `labeltree_anc`



Preorder trace:
$a^-, a^+, c^-, a^+, \underline{c}^-, c^+, \mathbf{b}^-, a^+.$

Fig. 1. Extended labels in a labeled tree. The first c-ancestor of the b-node x (in bold) corresponds to the last occurrence of c^- (underlined) in preorder before x.

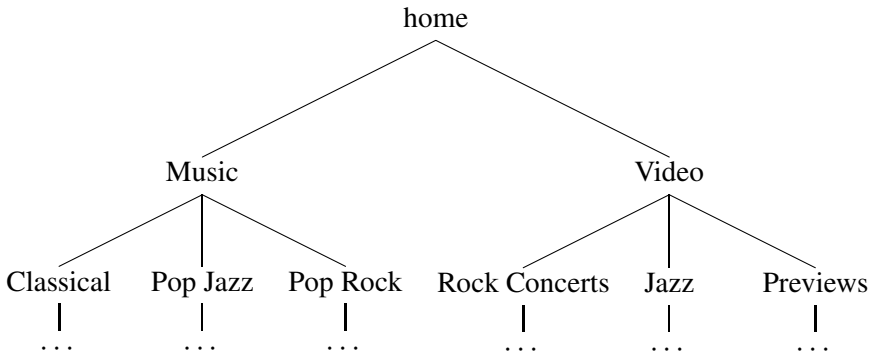


Fig. 2. A simple example of file system.

operator, we extend the labeling scheme by one bit (i.e. to an alphabet of size 2σ), using n additional bits in the encoding. Formally, any node x originally labeled α is now labeled

- α^- if x has no ancestor labeled α;
- α^+ if x has at least one ancestor labeled α.

See Fig. 1 for an example. The sequence of extended labels in preorder is encoded using the representation of Golynski et al. [11, Theorem 3.2], which uses $n(\lg(2\sigma) + o(\lg(2\sigma))) = n(\lg\sigma + o(\lg\sigma))$ bits and supports on the extended alphabet the operators `string_access` and `string_rank` in $O(\lg\lg\sigma)$ time, and the operator `string_select` in constant time. The total encoding size is $2n + o(n) + n(\lg\sigma + o(\lg\sigma)) = n\big(\lg\sigma + o(\lg\sigma)\big)$ bits.

The operator `labeltree_anc`(α, x) is supported by finding the last node y before x labeled α^- in preorder traversal, which takes time $O(\lg\lg\sigma)$, and checking whether y is an ancestor of x, which takes constant time.

The operator `labeltree_desc`(α, x) is similarly supported by finding the first node y labeled α^- or α^+ in preorder traversal after x, which takes time $O(\lg\lg\sigma)$, and checking whether y is a descendant of x, which takes constant time.

As the descendants of the node x are all consecutive in the preorder traversal of the tree, the operator `labeltree_nbdesc`(α, x) returns the number of occurrences of the label α (i.e., the sum of the number of occurrences of both α^- and α^+) in the string `Labels` between the positions corresponding to the first and last descendants of x.

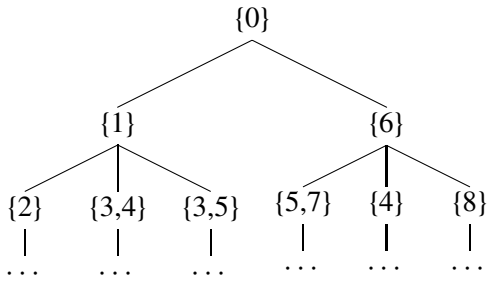
The operator `labeltree_succ`(α, x) is supported in time $O(\lg\lg\sigma)$: it corresponds to two pairs of calls to operators `string_rank` and `string_select` to find first the first node labeled α^- or α^+ after x in preorder. □

Note that the space used by our data structure, $n\big(\lg\sigma + o(\lg\sigma)\big)$, is asymptotically equal to the information-theoretic lower bound of $2n - o(n) + n\lg\sigma$ bits for storing a labeled tree on n nodes with labels from $[\sigma]$.

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 one or more labels could be associated with each leaf in an XML document, or with each internal node (folder) and leaf (file) in a file system. Fig. 2 shows an example of a file system storing music and video files.

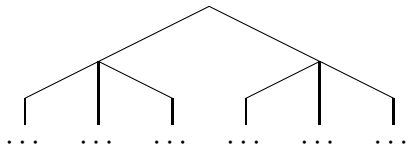
Definition 3.4. A **multi-labeled tree** is an ordinal tree on n nodes associated with labels from $[\sigma]$, and a set of t pairs from $[n] \times [\sigma]$.

We extend the operators described on labeled trees to multi-labeled trees: structure-based navigation operators and label-based operators, as in Theorem 3.3.



0=home, 1=Music, 2=Classical, 3=Pop, 4=Jazz, 5=Rock, 6=Video, 7=Concerts, 8=Previews.

Fig. 3. The multi-labeled tree corresponding to the file system of Fig. 2. The keywords are replaced by simple label identifiers, and several ones can be assigned to each node.



0 1 0 1 0 1	0 0 1	0 0 1	0 1 0 0 1	0 1	0 1
0 1 2	3 4	3 5	6 5 7	4	8

Fig. 4. Fragments (separated by vertical bars) from the encoding of the multi-labeled tree of Fig. 3. Each label of a node x is aligned with the corresponding zero in the unary encoding of the number of labels associated to x.

Corollary 3.5. *Consider a multi-labeled tree on n nodes associated with labels from $[\sigma]$ in t pairs with $t \geq \max\{n, \sigma\}$. There is an encoding that uses $t\big(\lg \sigma + o(\lg \sigma)\big)$ bits of space and supports the operators of Theorem 3.3, with the same asymptotic run-time complexities.*

Proof. We use essentially the same encoding as in Theorem 3.3, except that we encode a binary relation (instead of a string) using Theorem 3.1, so that the operators `string_rank` and `string_select` are replaced by the operators `label_rank` and `label_select`. □

Fig. 3 shows the representation of the file system shown in Fig. 2 as a multi-labeled tree, where the text associated with each node is replaced by numbers from the set $[\sigma]$. Fig. 4 shows the succinct encoding of this multi-labeled tree: the structure of the ordinal tree, the string representing the labels in preorder, and a binary string where ones separate sequences of zeroes encoding the number of labels associated to a node. As in Section 3.1, the space used by our structure is only optimal under the assumption that $t/n = \sigma^{o(1)}$, i.e. that on average each node is associated with a small number of labels.

4. Algorithms

4.1. Efficient postings lists

As noted in Section 2.2, several algorithms have been proposed for computing the answer to conjunctive queries on a binary relation through the intersection of postings lists, and their complexity is expressed as a function of a difficulty measure of the instance. We consider the difficulty measure defined by Barbay and Kenyon [2], based on the definition of a *certificate* of the answer to a conjunctive query. Our implementation of binary relations, described in Section 3.1, allows us to search faster in the list of references associated with an object, and hence improves the performance of Barbay and Kenyon's algorithm:

Theorem 4.1. *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 that answers Q in time $O(\delta k \lg \lg \sigma)$, where δ is the alternation of Q, a measure of difficulty defined by Barbay and Kenyon [2].*

Proof. Barbay and Kenyon [2, Definition 2.5] defined the partition-certificate of an instance as a partition such that each interval has an empty intersection with at least one of the sets. They measure the difficulty of an instance through its "alternation", the minimal size of a partition-certificate of it [2, Definition 3.1]. The alternation is closely related to the non-deterministic complexity of the instance, the smallest number of operations required to certify the answer to

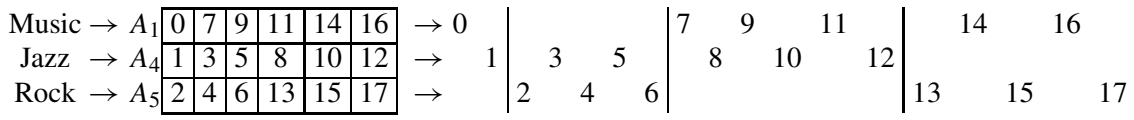


Fig. 5. An example of how a conjunctive query composed of three keywords corresponds to the intersection of the three corresponding sets. The alternation of the instance is $\delta = 4$, the number of intervals of a partition certificate where each interval has an empty intersection with at least one of the sets. Barbay and Kenyon's algorithm performs $7 \leq \delta k = 12$ searches (for the numbers 0, 1, 2, 7, 8, 13, 14).

the query: on instances where the answer is empty, the alternation is exactly one plus the non-deterministic complexity (see Fig. 5 for an example). Barbay and Kenyon proposed an algorithm for the conjunctive query that uses $O(\delta k)$ searches [2, Proof of Theorem 3.4], where k is the number of terms in the conjunctive query and δ is the alternation of the query.

We introduce an extra object ∞, which matches all labels and is a successor to all objects.

The algorithm is as follows:

(1) $x \leftarrow 0$; $\alpha \leftarrow$ first label of Q;
(2) **If** $x = \infty$, exit;
(3) **If** k labels are matched, output x,
set x to the next object matching α, and go to (2);
Otherwise, set α to the next label from Q, in cyclic order;
(4) **If** x matches α, go to (3);
Otherwise, set x to the next object matching α, and go to (2).

The search for the next object matching a label α (lines (3) and (4)) is supported using the operators `label_rank` and `label_select`. The test of line (4) is supported using the operator `table_access`. As all operators are supported in time $O(\lg \lg \sigma)$, the complexity of the algorithm is $O(\delta k)$. □

Note that performing $O(\delta k)$ searches is within a constant factor of optimality, as any randomized algorithm performs $\delta(k-1)/2$ searches on some instance in the comparison model [2]. Any algorithm which performs searches in several postings lists can benefit from our data structure (e.g., the algorithms `SmallAdaptive` and `SvS` studied by Demaine et al. [7] and later by Barbay et al. [3], the deterministic and randomized algorithms proposed by Barbay and Kenyon [2], and the algorithm introduced by Baeza-Yates [1]). The only algorithms which do not directly benefit from our data structures are algorithms which control the search for elements at the level of the comparisons, such as the algorithm `Adaptive` proposed by Demaine et al. [6].

4.2. File system search

We suggest a file system index which associates several keywords with each folder or file (e.g., in the filename `final_soda.tex`, the words `final` and `soda` and the extension `tex`, or some of the words contained in the file): we represent it as a multi-labeled tree. We introduce a new type of query to search in labeled and multi-labeled trees, that corresponds to one of the most natural search queries that one can perform in a file-system.

Definition 4.2. Given a multi-labeled tree and a set Q of k labels, the answer to an *path-subset query* is 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. Condition (2) ensures the succinctness of the answer, as the subtrees corresponding to the answer are disjoint. For instance, the answer to the query "Rock Music" would be the root of the third subtree of the file system of Fig. 2, and the answer to the query "Rock" would be the root of the third and fourth subtrees.

As for conjunctive queries, the worst case analysis is not sufficient to differentiate the algorithms; hence we define a measure of difficulty on instances.

Definition 4.3. Consider a multi-labeled tree and a set Q of k labels. A *partition-certificate* is a partition $(I_i)_{i\in[\delta]}$ of size δ on the set of nodes $[n]$, such that for any $i \in [\delta]$, either

- the common rooted path of all nodes of I_i matches the k labels of the query; or
- there is a label α such that no node of I_i has an ancestor matching α.

We call δ the *alternation* of Q on this multi-labeled tree.

Using techniques similar to those used for the intersection problem, we prove the following result:

Theorem 4.4. *Consider a multi-labeled tree of n nodes associated with labels from $[\sigma]$ in t pairs. Given a path-subset query Q composed of k labels, there is an algorithm answering it which performs $O(\delta k)$ operator calls, and which takes time $O(\delta k \lg\lg\sigma)$, where δ is the alternation of Q on this multi-labeled tree.*

Proof. If we consider the nodes in preorder, and introduce an extra node ∞ that matches all labels and is a successor to all nodes (i.e. ∞ is appended at the end of the preorder sequence), our algorithm proceeds as follows:

(1) $x \leftarrow 0$; $\alpha \leftarrow$ first label of Q;
(2) **If** $x = \infty$, exit;
(3) **If** k labels are matched, output x,
set it to the next node matching α (in preorder)
which is not a descendant of x,[5] and go to (2);
Otherwise, set α to the next label from Q in cyclic order;
(4) **If** x has an ancestor labeled α, go to (3);
(5) **If** x has a descendant labeled α,
set it to the first such descendant (in preorder), and go to (3);
Otherwise, set x to the next node matching α (in preorder), and go to (2).

The search for an ancestor or a descendant labeled α is supported by the operators `labeltree_anc` and `labeltree_desc`, and the next node matching α in preorder is obtained through the operator `labeltree_succ` (see Section 3.2).

This algorithm cycles through the labels of Q, so that x refers to the node with smallest rank in preorder of the current potential match. The preorder rank of successive nodes pointed to by x is strictly increasing at each update, so that at any time, all preorder predecessors of x have been considered and have been output if adequate. Given a partition-certificate $(I_i)_{i\leq\delta}$ of size δ, we divide the execution of the algorithm in δ phases, so that the algorithm is in phase i if the node considered x is in I_i. As the algorithm considers the labels of Q in cyclic order, it performs at most k iterations of the loop in each phase, before it has eliminated at least all the nodes of the interval I_i.

When the preorder rank of x reaches its final value, all nodes have been considered (hence the correctness), and the algorithm has performed at most $O(\delta k)$ operator calls. As each operator call costs time $O(\lg\lg\sigma)$, the algorithm performs $O(\delta k)$ searches in time $O(\delta k \lg\lg\sigma)$ to solve the query. □

Unless the operators defined in Section 3.1 can be supported more efficiently, we prove that this result is optimal for deterministic algorithms, in the worst case (depending on the algorithm) as well as on average on a distribution independent of the algorithm (which is a much stronger result, leading to Theorem 4.6).

Lemma 4.5. *Consider any deterministic algorithm Alg answering path-subset queries, and $\delta \geq 1$, $k \geq 2$, $n \geq 2\delta k+1$, and $\sigma \geq 2k+1$. There is a probability distribution D on labeled trees with $O(n)$ nodes and $O(\sigma)$ labels, and a path-subset query composed of k labels of alternation at most $O(\delta)$ on any labeled tree from D, such that Alg performs $\Omega(\delta k)$ operator calls on average to solve instances from D.*

Proof. We first define a distribution D_1 proving the result in the case where $\delta = 1$, and we draw a random labeled tree from D with the desired properties by combining δ labeled trees randomly drawn from D_1.

Define a "double branch" tree as one consisting of a root with two children, each of which has a single chain of $k-1$ descendants. Hence the tree has $2k+1$ nodes, two of which are leaves at depth k. Let the tree P be the double

[5] This is easily supported as any encoding of ordinal tree supports the size of the subtree rooted at node x.

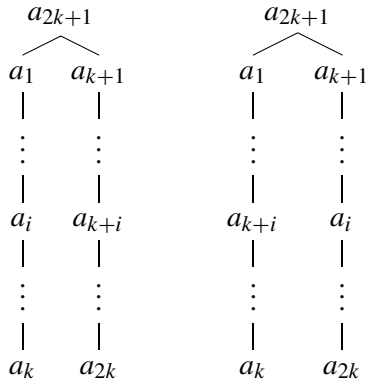


Fig. 6. The double branch trees P with a single match (on the left), and N_i without any match (on the right).

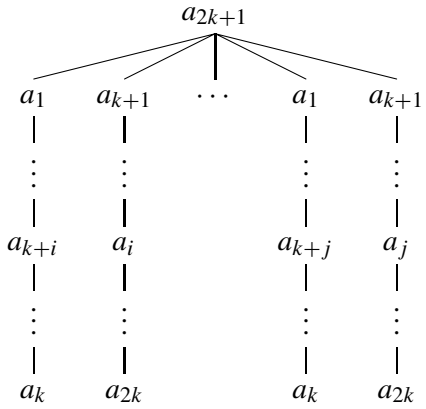


Fig. 7. A general tree, composed of δ double branch trees joined by the root, drawn randomly from $\{P, N_1, \ldots, N_k\}$.

branch tree with root labeled a_{2k+1} such that the nodes of one branch are labeled $a_1, \ldots, a_k$, and the nodes of the other branch are labeled $a_{k+1}, \ldots, a_{2k}$, both from top to leaf. Define for any $i \in \{1, \ldots, k\}$ the labeled tree N_i by switching the labels in P of the two nodes at depth i, as illustrated in Fig. 6. The trees $P, N_1, \ldots, N_k$ are very similar: to prove or disprove the existence of a match of query $\{a_1, \ldots, a_k\}$ any deterministic algorithm, given only the operators of the succinct encoding, has to perform k operator calls in the worst case. We define D_1 to be the uniform distribution on trees $P, N_1, \ldots, N_k$.

Any deterministic algorithm accessing the tree only through the three operators `labeltree_anc`, `labeltree_desc` or `labeltree_nbdesc` will perform on average more than $k/2$ operator calls before being able to decide if the tree has a match or not, hence the result for $\delta = 1$.

Draw a tree from distribution D by picking independently δ trees from D_1, and joining them at the root, as described in Fig. 7. The tree formed has $2\delta k + 1 \leq n$ nodes labeled from an alphabet of size $2k+1 \leq \sigma$, and 2δ operator calls are sufficient to check which nodes match the query $\{a_1, \ldots, a_k\}$, if any. As each double branch forming the tree has the same number of α-nodes for any label α, the operations performed in one particular double branch gives no clue about the presence of a match in another double branch, hence the lower bound of $\delta k/2$ operator calls on average, and the desired result. □

Now we use the Yao–von Neumann principle [18,20,22] to prove a lower bound on the complexity of any randomized algorithm:

Theorem 4.6. *Consider any randomized algorithm RandAlg answering path-subset queries, and $\delta \geq 1$, $n \geq 2\delta k+1$, $k \geq 2$, and $\sigma \geq 2k+1$. There is a labeled tree of $O(n)$ nodes in association with $O(\sigma)$ labels, and a path-subset query composed of k labels of alternation at most $O(\delta)$, such that RandAlg performs on average $\Omega(\delta k)$ operator calls to answer the query.*

Proof. Lemma 4.5 gives a distribution on which any deterministic algorithm performs poorly on average. The Yao–von Neumann principle permits the deduction from this distribution of a lower bound on the worst case complexity of randomized algorithms. □

The proof is similar to its counterpart on the intersection problem [2]. In particular, Theorems 4.4 and 4.6 show that a deterministic algorithm performs as well as any randomized algorithm for path-subset queries, in term of the number of operator calls. Note that, since labeled trees form a subset of multi-labeled trees, the lower bounds hold for multi-labeled trees as well.

5. Conclusion

In this paper, we consider data structures for binary relations, labeled trees and multi-labeled trees, and adaptive algorithms on these data structures. We propose various encodings which support the basic operators, and we use these operators to solve efficiently conjunctive queries on binary relations, and path-subset queries on labeled and multi-labeled trees. These results extend the results from Golynski et al. [11] on strings on large alphabets, and from

Barbay and Kenyon [2] on the intersection problem, and can be applied to other intersection algorithms as well. They can be further extended to support other operators on binary relations.

The representation defined by Golynski et al. [11] allows us to efficiently extend succinct encodings for unlabeled trees to the labeled case, and our representation of binary relations allows us to extend them further to multi-labeled cases. It would be interesting to apply similar techniques to other structures, such as graphs, and to extend them to multidimensional relations.

The relation between algorithms for conjunctive queries and for path-subset queries in labeled trees is interesting: the intersection problem has been largely studied, and applying this work to labeled or multi-labeled trees opens many perspectives. In particular, path-subset queries are fairly simple, but the technique can also be applied for more general pattern matching problems.

Acknowledgments

We would like to thank Aleh Veraskouski for his constructive comments on the adaptive algorithm answering path-subset 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, in: Lecture Notes in Computer Science, vol. 3109, Springer, 2004, pp. 400–408.
[2] J. Barbay, C. Kenyon, Alternation and redundancy analysis of the intersection problem, ACM Transactions on Algorithms 4 (1) (2008) 18 pp. (in press).
[3] J. Barbay, A. López-Ortiz, T. Lu, Faster adaptive set intersections for text searching, in: 5th International Workshop on Experimental Algorithms, in: Lecture Notes in Computer Science, vol. 4007, Springer, 2006, pp. 146–157.
[4] D. Benoit, E.D. Demaine, J.I. Munro, R. Raman, V. Raman, S.S. Rao, Representing trees of higher degree, Algorithmica (2005) 275–292.
[5] D.R. Clark, J.I. Munro, Efficient suffix trees on secondary storage, in: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 383–391.
[6] E.D. Demaine, A. López-Ortiz, J.I. Munro, Adaptive set intersections, unions, and differences, in: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 743–752.
[7] E.D. Demaine, A. López-Ortiz, J.I. Munro, Experiments on adaptive set intersections for text retrieval systems, in: Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments, in: Lecture Notes in Computer Science, vol. 2153, Springer, 2001, pp. 91–104.
[8] V. Estivill-Castro, D. Wood, A survey of adaptive sorting algorithms, ACM Computing Surveys 24 (4) (1992) 441–476.
[9] P. Ferragina, F. Luccio, G. Manzini, S. Muthukrishnan, Structuring labeled trees for optimal succinctness, and beyond, in: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science, 2005, pp. 184–196.
[10] R.F. Geary, R. Raman, V. Raman, Succinct ordinal trees with level-ancestor queries, in: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 1–10.
[11] A. Golynski, J.I. Munro, 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 Algorithms, 2006, pp. 368–373.
[12] R. Grossi, A. Gupta, J.S. Vitter, High-order entropy-compressed text indexes, in: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 841–850.
[13] G. Jacobson, Space-efficient static trees and graphs, in: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 549–554.
[14] W.-K.S. Jesper Jansson, Kunihiko Sadakane, Ultra-succinct representation of ordered trees, in: Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms, 2007.
[15] D.G. Kirkpatrick, R. Seidel, The ultimate planar convex hull algorithm? SIAM Journal on Computing 15 (1) (1986) 287–299.
[16] J.I. Munro, V. Raman, Succinct representation of balanced parentheses and static trees, SIAM Journal on Computing 31 (3) (2001) 762–776.
[17] J.I. Munro, S.S. Rao, Succinct representations of functions, in: 31st International Colloquium on Automata, Languages and Programming, in: Lecture Notes in Computer Science, Springer, 2004, pp. 1006–1015.
[18] J.V. Neumann, O. Morgenstern, Theory of Games and Economic Behavior, first ed., Princeton University Press, 1944.
[19] R. Raman, V. Raman, S.S. Rao, Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, in: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete algorithms, 2002, pp. 233–242.
[20] M. Sion, On general minimax theorems, Pacific Journal of Mathematics 8 (1958) 171–176.
[21] D.E. Willard, Log-logarithmic worst-case range queries are possible in space $\Theta(N)$, Information Processing Letters 17 (2) (1983) 81–84.
[22] A.C. Yao, Probabilistic computations: Toward a unified measure of complexity, in: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 222–227.