Compact Rich-Functional Binary Relation Representations

Jérémy Barbay[1], Francisco Claude[2] *, and Gonzalo Navarro[1] **

[1] Department of Computer Science, University of Chile. {jbarbay|gnavarro}@dcc.uchile.cl.
[2] David R. Cheriton School of Computer Science, University of Waterloo. fclaude@cs.uwaterloo.ca.

Abstract. Binary relations are an important abstraction arising in a number of data representation problems. Each existing data structure specializes in the few basic operations required by one single application, and takes only limited advantage of the inherent redundancy of binary relations. We show how to support more general operations efficiently, while taking better advantage of some forms of redundancy in practical instances. As a basis for a more general discussion on binary relation data structures, we list the operations of potential interest for practical applications, and give reductions between operations. We identify a set of operations that yield the support of all others. As a first contribution to the discussion, we present two data structures for binary relations, each of which achieves a distinct tradeoff between the space used to store and index the relation, the set of operations supported in sublinear time, and the time in which those operations are supported. The experimental performance of our data structures shows that they not only offer good time complexities to carry out many operations, but also take advantage of regularities that arise in practical instances in order to reduce space usage.

1 Introduction

Binary relations appear everywhere in Computer Science. Graphs, trees, inverted indexes, strings and permutations are just some examples. Apart from their pervasiveness as such, binary relations have been used as a tool to complement existing data structures (such as trees [3] or graphs [2]) with additional information, such as weights or labels on the nodes or edges, that can be indexed and searched. Interestingly, the data structure support for binary relations has not undergone a systematic study, but rather one triggered by particular applications: we aim to remedy this fact.

Let us say that a binary relation $\mathcal{B}$ relates *objects* in $[1, n]$ with *labels* in $[1, \sigma]$, containing t pairs out of the $n\sigma$ possible ones. A simple entropy measure using these parameters and ignoring any other possible regularity is $H(\mathcal{B}) = \log \binom{n\sigma}{t} = t \log \frac{n\sigma}{t} + O(t)$ bits (logarithms are base 2 in this paper). Fig. 1 (left) shows an example of binary relation (identifying labels with rows and objects with columns henceforth).

Previous work focused on relatively basic primitives for binary relations: extract the list of all labels associated to an object or of all objects associated to a label (an operation called `access`), or extracting the r-th such element (an operation called `select`), or counting how many of these are there up to some object/label value (called operation `rank`).

The first representation specifically designed for binary relations [3] supports `rank`, `select` and `access` on the rows (labels) of the relation, for the purpose of supporting faster joins on labels, via a reduction to the rank and select operators on strings, later extended to index text [13], and to separate the content from the index [4], which in turn allows supporting labeled operations on planar and quasi-planar labeled graphs [2].

Ad-hoc compressed representations for inverted lists [21] and Web graphs [12] can also be considered as supporting binary relations. The idea here is to write the objects of the pairs, in

* Funded by NSERC of Canada and Go-Bell Scholarships Program.
** Funded in part by Fondecyt Grant 1-080019, Chile.

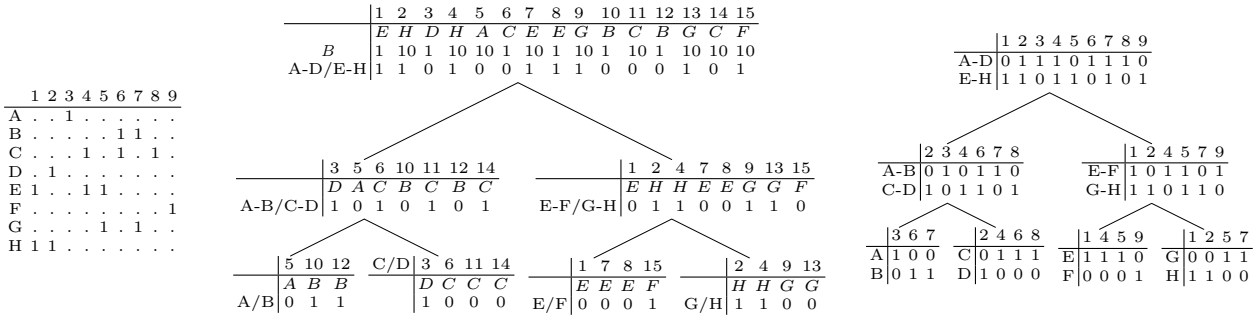

Fig. 1. An example of binary relation (left), its representation according to Sec. 4 (middle) and according to Sec. 5 (right). Note that the labels and object numbers are included in each node solely for ease of reading; in the encoding they are implicit.

label-major order, and support extracting substrings of the resulting string, that is, little more than `access` on labels. The string can be compressed by different means depending on the application.

In this paper we aim at describing the foundations of efficient compact data structures for binary relations. We list operations of potential interest for practical applications of binary relations; we give various reductions between operators, thus identifying a core set of operations whose support yields the support of all others; we present two data structures for binary relations (with some variants), each of which achieves a distinct tradeoff between the space used to store and index the relation, the set of operations supported in sublinear time, and the time in which those operations are supported; and we compare the practical performances of the suggested data structures between themselves and with the theoretical entropy, showing that our data structures not only offer good time complexities to carry out many operators, but also reduce the space used by taking advantage of the redundancy of practical instances.

Our first data structure uses the reduction of binary relation operators to string operators [3], but in conjunction with a wavelet tree rather than one based on permutations, which improves the time of many operations. Our second data structure extends the wavelet tree for strings to binary relations. The space used is potentially smaller than for the previous data structure (close to $H(\mathcal{B})$ bits), at the cost of worse time for some operations, but it permits taking further advantage of some common regularities present in real-life binary relations. For the sake of simplicity, we aim for the simplest description of the operations, ignoring any practical improvement that does not make a difference in terms of complexity, or trivial extensions such as interchanging labels and objects to obtain other space/time tradeoffs.

2 Basic Concepts

Given a sequence S of length n, drawn from an alphabet Σ of size σ, we want to answer the queries: (1) $\texttt{rank}_a(S, i)$ counts the occurrences of symbol $a \in \Sigma$ in $S[1, i]$; (2) $\texttt{select}_a(S, i)$ finds the i-th occurrence of symbol $a \in \Sigma$ in S; and (3) $\texttt{access}(S, i) = S[i]$. We omit S if clear from context.

For the special case $\Sigma = \{0, 1\}$, the problem has been solved using $n + o(n)$ bits of space while answering the three queries in constant time [10]. This was later improved to use $nH_0(S) + o(n)$ bits [20]. Here $H_0(S)$ is the *zero-order entropy* of sequence S, defined as $H_0(S) = \sum_{a \in \Sigma} \#_a/n \log(n/\#_a)$, where $\#_a$ is the number of occurrences of symbol a in S.

The wavelet tree [15] reduces the `rank/select/access` problem for general alphabets to those on binary sequences. It is a perfectly balanced tree that stores a bitmap of length n at the root; every position in the bitmap is either 0 or 1 depending on whether the symbol at this position belongs to the first half of the alphabet or to the second. The left child of the root will handle the subsequence of S marked with a 0 at the root, and the right child will handle the 1s. This decomposition into alphabet subranges continues recursively until reaching level $\lceil \log \sigma \rceil$, where the leaves correspond to individual symbols. We call B_v the bitmap at node v.

The `access` query $S[i]$ can be answered by following the path described for position i. At the root v, if $B_v[i] = 0/1$, we descend to the left/right child, switching to the bitmap position $\texttt{rank}_{0/1}(B_v, i)$ in the left/right child, which then becomes the new v. This continues recursively until reaching the last level, when we arrive at the leaf corresponding to the answer symbol.

Query $\texttt{rank}_a(S, i)$ can be answered in a way similar to `access`, the difference being that we descend according to a and not to the bit of B_v. We update position i for the child node just as before. At the leaves, the final bitmap position i corresponds to the answer.

Query $\texttt{select}_a(S, i)$ proceeds as `rank`, but upwards. We start at the leaf representing a and update i to $\texttt{select}_{0/1}(B_v, i)$ where v is the parent node, depending on whether the current node is its left/right child. At the root, the position i is the final result.

Wavelet trees require $n \log \sigma + o(n) \log \sigma$ bits of space, while answering all the queries in $O(\log \sigma)$ time. If the bitmaps B_v are represented using the technique of Raman et al. [20], the wavelet tree uses $nH_0(S) + o(n) \log \sigma$ bits. Fig. 1 (middle) illustrates the structure. Wavelet trees are not only used to represent strings [14], but also grids [7], permutations [5], and many other structures.

3 Operations

3.1 Definition of operations

Data structures for binary relations which support efficiently the `rank` and `select` operations on the row (label) yield faster searches in relational databases and text search engines [3] and, in combination with data structures for ordinal trees, yield faster searches in multi-labeled trees, such as those featured by semi-structured documents [3] (e.g. XML). A similar technique [2] combining various data structures for graphs with binary relations yields a family of data structures for edge-labeled and vertex-labeled graphs that support labeled operations on the neighborhood of each vertex. The extension of those operations to the union of labels in a given range allows them to handle more complex queries, such as conjunctions of disjunctions.

As a simple example, an inverted index [21] can be seen as a relation between vocabulary words (the labels) and the documents where they appear (the objects). Apart from the basic operation of extracting the documents where a word appears (`access` on the row), we want to intersect rows (implemented on top of row `rank` and `select`) for phrase and conjunctive queries (popular in Google-like search engines). Extending these operations to a range of words allows for stemmed and/or prefix searches (by properly ordering the words). Extracting a column gives important *summarization* information on a document: the list of its different words. Intersecting columns allows for analysis of content between documents (e.g. plagiarism or common authorship detection). Handling ranges of documents allows for considering hierarchical document structures such as XML or filesystems (search within a subtree or subdirectory).

As another example, a directed graph is just a binary relation between vertices. Extracting rows or columns supports direct and reverse navigation from a node. In Web graphs, where the nodes

(Web pages) are usually sorted by URL, ranges of nodes correspond to domains and subdirectories. For example, counting the number of connections between two ranges of nodes allows estimating the connectivity between two domains. In general, considering domain ranges permits the analysis and navigation of the Web graph at a coarser granularity (e.g. as a graph of hosts, or institutions).

Several text indexing data structures [8, 13, 16, 17, 19] resort to a grid, which relates for example text suffixes (in lexicographical order) with their text positions, or phrase prefixes and suffixes in grammar compression, or two labels that form a rule in straight-line programs, etc. The most common operation needed is counting and returning all the points in a range.

Obviously, the case where the relation represents a geometric grid, where objects and labels are simply coordinates, and where pairs of the relation are points at those coordinates, is useful for GIS and other geometric applications. The generalization of the basic operations to ranges allows for counting the number of points in a rectangular area, and retrieving them in different orders.

These examples illustrate several useful ways to extend the definition of the `rank` and `select` operations from single rows (labels) or columns (objects) to ranges over both rows and columns. Consider for instance the extension of `select` to ranges of labels: `select`(α, r) yields the position of the r-th 1 in the row α of the matrix (see Fig. 1 (middle)), corresponding to the r-th object associated to label α. On the range of rows $[\alpha, \beta]$, the expression "the r-th 1" requires a total order on the two-dimensional area defined by the range (e.g. label-major or object-major), which yields two distinct extensions of the operation. Other applications require instead a `select` operation that retrieves the r-th object associated to any label from a given range, regardless of how many pairs the object participates in. That is, to skip over the columns that are empty in that label range.

We generalize the `access` operation to ranges of labels and objects by supporting the search for the minimal (resp. maximal) label or object that participates in a given rectangular area of the relation, and the search for the first related pair (in label-major or object-major order) in this area. Among other applications, this supports the search for the highest (resp. lowest) neighbor of a point, when the binary relation encodes the levels of points in a planar graph representing a topography map [2].

The sum of those examples yields many distinct extensions for each of the `rank`, `select` and `access` operations. We list in Table 1 their formal definitions. Each of them is useful to improve the performance of various applications of binary relation data structures.

3.2 Reductions between operations

The solid arrows in Fig. 2 show the constant-time reductions that we identified among the operations; disregard the rest for now. A solid arrow $op \rightarrow op'$ means that solving op we also solve op'. First, `rel_rnk` is a particular case of `rel_num`, whereas the latter can be supported by adding/subtracting four `rel_rnk` queries at the corners of the rectangle. Hence they are equivalent. With a constant number of any of these we also cover the areas described by `rel_rnk_obj_maj` and `rel_rnk_lab_maj`, and vice versa, thus these are equivalent too. Obviously, `obj_rnk1` and `lab_rnk1` are particular cases of `rel_num`. Also, `lab_rnk1` is a particular case of `lab_rnk`, itself a particular case of `lab_num`. Note that `lab_num` does not reduce to `lab_rnk` because a label could be related with objects inside and outside the range $[x, y]$. Similar reductions hold for objects.

Obviously the support for the select operation `rel_sel_lab_maj` implies the support for the access operation `rel_acc_lab_maj`, and accessing the first result of the latter gives the solution for the minimum operation `rel_min_lab_maj`. In turn this gives the minimum label in a range $[\alpha, \sigma] \times [x, y]$, thus if this label is β, we get the next label by rerunning the query on $[\beta+1, \sigma] \times [x, y]$,

Table 1. Operations of interest for binary relations on $[1,\sigma] \times [1,n]$ (labels × objects). x, y, z are objects (usually such that $x \le z \le y$); α, β, γ are labels (usually such that $\alpha \le \gamma \le \beta$); r is an integer (typically an index, parameter of a `select` operation) and '#' is short for 'number'. The solutions for maxima are similar to those for minima. The last two columns are the complexities we achieve in Section 4 and 5, respectively, per delivered datum.

Operation	Meaning	String	BRWT
`rel_num`(α,β,x,y)	# of pairs in $[\alpha,\beta]\times[x,y]$	$O(\log\sigma)$	$O(\beta-\alpha+\log\sigma)$ (*)
`rel_rnk`(α,x)	# of pairs in $[1,\alpha]\times[1,x]$	$O(\log\sigma)$	$O(\alpha+\log\sigma)$
`rel_rnk_lab_maj`(x,y,α,z)	# of pairs within $[x,y]$, up to (α,z), in label-major order	$O(\log\sigma)$	$O(\alpha+\log\sigma)$ (*)
`rel_sel_lab_maj`(α,r,x,y)	r-th pair within $[x,y]\times[\alpha,\sigma]$ in label-major order	$O(\log\sigma)$	$O(r\log\sigma)$ (*)
`rel_acc_lab_maj`(α,x,y)	consecutive pairs within $[\alpha,\sigma]\times[x,y]$ in label-major order	$O(\log\sigma)$	$O(\log\sigma)$
`rel_min_lab_maj`(α,x,y)	minimum pair within $[\alpha,\sigma]\times[x,y]$ in label-major order	$O(\log\sigma)$	$O(\log\sigma)$
`rel_rnk_obj_maj`(α,β,γ,x)	# of pairs within $[\alpha,\beta]$, up to (γ,x), in object-major order	$O(\log\sigma)$	$O(\beta-\alpha+\log\sigma)$
`rel_sel_obj_maj`(α,β,x,r)	r-th pair within $[\alpha,\beta]\times[x,n]$ in object-major order	(+)	$O(r\log\sigma)$ (*)
`rel_acc_obj_maj`(α,β,x)	consecutive pairs within $[\alpha,\beta]\times[x,n]$ in object-major order	$O(\log\sigma)$	$O(\log\sigma)$
`rel_min_obj_maj`(α,β,x)	minimum pair within $[\alpha,\beta]\times[x,n]$ in object-major order	$O(\log\sigma)$	$O(\log\sigma)$
`lab_num`(α,β,x,y)	# of distinct labels within $[\alpha,\beta]\times[x,y]$	$O(\beta-\alpha+\log\sigma)$	$O(\beta-\alpha+\log\sigma)$
`lab_rnk`(α,x,y)	# of distinct labels within $[1,\alpha]\times[x,y]$	$O(\alpha+\log\sigma)$	$O(\alpha+\log\sigma)$
`lab_sel`(α,r,x,y)	r-th distinct label within $[\alpha,\sigma]\times[x,y]$	$O(r\log\sigma)$	$O(r\log\sigma)$
`lab_acc`(α,x,y)	consecutive labels within $[\alpha,\sigma]\times[x,y]$	$O(\log\sigma)$	$O(\log\sigma)$
`lab_min`(α,x,y)	minimum label within $[\alpha,\sigma]\times[x,y]$	$O(\log\sigma)$	$O(\log\sigma)$
`obj_num`(α,β,x,y)	# of distinct objects within $[\alpha,\beta]\times[x,y]$	$O(r\log\sigma)$	$O(r\log\sigma)$
`obj_rnk`(α,β,x)	# of distinct objects within $[\alpha,\beta]\times[1,x]$	$O(r\log\sigma)$	$O(r\log\sigma)$
`obj_sel`(α,β,x,r)	r-th distinct object within $[\alpha,\beta]\times[x,n]$	$O(r\log\sigma)$	$O(r\log\sigma)$
`obj_acc`(α,β,x)	consecutive objects within $[\alpha,\beta]\times[x,n]$	$O(\log\sigma)$	$O(\log\sigma)$
`obj_min`(α,β,x)	minimum object within $[\alpha,\beta]\times[x,n]$	$O(\log\sigma)$	$O(\log\sigma)$
`lab_rnk1`(α,x)	# of distinct labels within $[1,\alpha]\times x$	$O(\log\sigma)$	$O(r\log\sigma)$
`lab_sel1`(α,r,x)	r-th distinct label within $[\alpha,\sigma]\times x$	$O(\log\sigma)$	$O(r\log\sigma)$
`obj_rnk1`(α,x)	# of distinct objects within $\alpha\times[1,x]$	$O(\log\sigma)$	$O(\log\sigma)$
`obj_sel1`(α,x,r)	r-th distinct object within $\alpha\times[x,n]$	$O(\log\sigma)$	$O(\log\sigma)$

(+) $O(\min(r,\log n,\log r\log(\beta-\alpha+1))\log\sigma)$
(*) $O(\log\sigma)$ if $[x,y]=[1,n]$

this way supporting `lab_acc`. The latter, in turn, gives the solution to `lab_min` in its first iteration, whereas successive invocations to `lab_min` (in a fashion similar to `rel_min_lab_maj`) solves `lab_acc`. Also analogously as before, `lab_sel` allows supporting `lab_min` by asking the first occurrence, and `lab_sel1` is a particular case of `lab_sel`. Note also that `rel_sel_lab_maj` allows supporting `lab_sel1`, by requiring the pairs starting at the desired rows, and extracting the resulting objects. By symmetry, analogous reductions hold for objects instead of labels.

The rest of the following theorem stems from inverse-function relations between `rank` and `select` queries, as well as one-by-one solutions to counting and direct-access problems.

Theorem 1. *All the arrows in Figure 2 represent constant-time reductions that hold for the operations. In addition, the pairs* (`lab_num`, `lab_sel`) *support each other with an* $O(\log\sigma)$ *penalty factor,* (`obj_num`, `obj_sel`) *with an* $O(\log n)$ *penalty factor, and* (`rel_rnk_lab_maj`, `rel_sel_lab_maj`) *and* (`rel_rnk_obj_maj`, `rel_sel_obj_maj`) *with an* $O(\log(\sigma n))$ *penalty factor. Finally, in pairs* (`lab_acc`, `lab_sel`), (`rel_acc_lab_maj`, `rel_sel_lab_maj`), (`obj_acc`, `obj_sel`), *and* (`rel_acc_obj_maj`, `rel_sel_obj_maj`), *the first operation supports the second with an* $O(r)$ *penalty factor, where* r *is the parameter of the* `select` *operation. Finally, the* `access` *operations support the corresponding* `rank` *(and counting) operations in time proportional to the answer of the latter.*

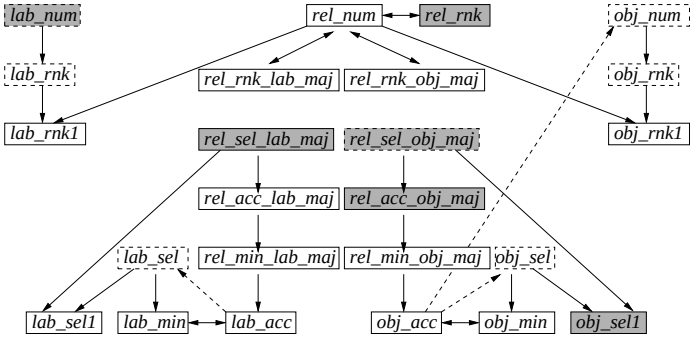

Fig. 2. Results achieved by reducing to strings. Grayed boxes are the operations we adressed directly; all the others are supported via reductions given in Thm. 1: constant-time ones are represented by solid arrows, and non-constant-time ones by dashed arrows. Operations supported in time $O(\log \sigma)$ are in solid squares. Dashed squares represent operations supported in higher time. We draw only the dashed arrows needed to follow the source of the operations supported via non-constant-time reductions.

4 Reduction to Strings

A simple representation for binary relations [3, 13] consists in a bitmap $B[1, n+t]$ and a string $S[1, t]$ over the alphabet $[1, \sigma]$. The bitmap $B[n + t]$ concatenates the consecutive cardinalities of the n columns of the relation in unary. The string S contains the rows (labels) of the pairs of the relation in column (object)-major order (see Fig. 1 (right)). Barbay *et al.* showed [3] that an easy way to support the `rank` and `select` operations on the rows of the binary relation is to support the `rank` and `select` operations on B and S, using any of the several data structures known for bitmaps and strings. We show that representing S using a wavelet tree yields the support for more complex operations. For this purpose, we define the mapping from a column number x to its last element in S as $\texttt{map}(x) = \texttt{rank}_1(B, \texttt{select}_0(B, x))$. The inverse, from a position in S to its column number, is $\texttt{unmap}(m) = \texttt{rank}_0(B, \texttt{select}_1(B, m)) + 1$. Both mappings take constant time. Finally, let us also define for shortness $\texttt{rank}_c(B, x, y) = \texttt{rank}_c(B, y) - \texttt{rank}_c(B, x - 1)$. The following operations are supported efficiently, and many others are derived with Thm. 1.

• $\texttt{rel\_rnk}(\alpha, x)$ **in** $O(\log \sigma)$ **time.** This is $\texttt{rank}_{\leq\alpha}(S, \texttt{map}(x))$, where operation $\texttt{rank}_{\leq\alpha}(S, m)$ counts the number of symbols $\leq \alpha$ in $S[1, m]$. It can be supported in time $O(\log \sigma)$ in a string wavelet tree by following the root-to-leaf branch corresponding to α, while counting at each node the number of objects preceding position m that are related with a label preceding α, as follows. Start at the root v with counter $c \leftarrow 0$. If α corresponds to the left subtree, then enter the left subtree with $m \leftarrow rank_0(B_v, m)$. Else enter the right subtree with $c \leftarrow c+rank_0(B_v, m)$ and $m \leftarrow \texttt{rank}_1(B_v, m)$. When a leaf is reached (indeed, that of α), the answer is $c + m$.

• $\texttt{rel\_sel\_lab\_maj}(\alpha, r, x, y)$ **in** $O(\log \sigma)$ **time.** We first get rid of α by setting $r \leftarrow r + \texttt{rel\_num}(1, \alpha-1, x, y)$ and thus reduce to the case $\alpha = 1$. Furthermore we map x and y to the domain of S by $x \leftarrow \texttt{map}(x-1)+1$ and $y \leftarrow \texttt{map}(y)$. We first find which is the symbol β whose row contains the r-th element. For this sake we first find the β such that $\texttt{rank}_{\leq\beta-1}(S, x, y) < r \leq \texttt{rank}_{\leq\beta}(S, x, y)$. This is achieved in time $O(\log \sigma)$ as follows. Start at the root v and set $r' \leftarrow r$. If $\texttt{rank}_0(B_v, x, y) \geq r$, then continue to the left subtree with $x \leftarrow \texttt{rank}_0(B_v, x - 1) + 1$ and $y \leftarrow \texttt{rank}_0(B_v, y)$. Else continue to the right subtree with $r' \leftarrow r' - \texttt{rank}_0(B_v, x, y)$, $x \leftarrow \texttt{rank}_1(B_v, x - 1) + 1$, and $y \leftarrow \texttt{rank}_1(B_v, y)$. The leaf arrived at is β. Finally, we set $r \leftarrow r - r'$, and answer $(\beta, \texttt{unmap}(\texttt{select}_\beta(S, r + \texttt{rank}_\beta(S, x - 1))))$.

• $\texttt{rel\_sel\_obj\_maj}(\alpha,\beta,x,r)$ **in** $O(\min(\log n, \log r \log(\beta-\alpha+1))\log\sigma)$ **time.** Object-major is the order in which the elements are written in S. First, we note that the particular case where $[\alpha,\beta]=[1,\sigma]$ is easily solved in $O(\log\sigma)$ time, by doing $r' \leftarrow r + \texttt{rel\_num}(\alpha,\beta,1,x-1)$ and returning $(S[r'],\texttt{unmap}(r'))$. In the general case, one can obtain time $O(\log n\log\sigma)$ by binary searching the column y such that $\texttt{rel\_num}(\alpha,\beta,x,y) < r \le \texttt{rel\_num}(\alpha,\beta,x,y+1)$. Then the answer is $(\texttt{lab\_sel1}(\alpha, r-\texttt{rel\_num}(\alpha,\beta,x,y),y),y)$. Finally, to obtain the other complexity, we find the $O(\log(\beta-\alpha+1))$ wavelet tree nodes that cover the interval $[\alpha,\beta]$; let these be $v_1,\ v_2,\ldots,v_k$. We map position x from the root towards those v_is, obtaining all the mapped positions x_i in $O(k+\log\sigma)$ time. Now the answer is within the positions $[x_i, x_i+r-1]$ of some i. We cyclically take each v_i, choose the middle element of its interval, and map it towards the root, obtaining position y, corresponding to pair $(S[y],\texttt{unmap}(y))$. If $\texttt{rel\_rnk\_obj\_maj}(\alpha,\beta,S[y],\texttt{unmap}(y)) - \texttt{rel\_rnk}(\alpha,\beta,1,x-1) = r$, the answer is $(S[y],\texttt{unmap}(y))$. Otherwise we know whether y is before or after the answer. So we discard the left or right interval in v_i. After $O(k\log r)$ such iterations we have reduced all the intervals of length r of all the nodes v_i, finding the answer. Each iteration costs $O(\log\sigma)$ time.
• $\texttt{rel\_acc\_obj\_maj}(\alpha,\beta,x)$ **in** $O(\log\sigma)$ **time per pair output.** Just as for the last solution of the previous operator, we obtain the positions x_i at the nodes v_i that cover $[\alpha,\beta]$. The first element to deliver is precisely one of those x_i. We have to merge the results, choosing always the smaller, as we return from the recursion that identifies the v_i nodes. If we are in v_i, we return $y=x_i$. Else, if the left child of v returned y, we map it to $y' \leftarrow \texttt{rank}_0(B_v,y)$. Similarly, if the right child of v returned y, we map it to $y'' \leftarrow \texttt{rank}_1(B_v,y)$. If we have only y' (y''), we return $y=y'$ ($y=y''$); if we have both we return $y=\min(y',y'')$. The process takes $O(\log\sigma)$ time. When we arrive at the root we have the next position y where a label in $[\alpha,\beta]$ occurs in S. We can then report all the pairs $(S[y+j],\texttt{unmap}(y))$, for $j=0,1,\ldots$, as long as $\texttt{unmap}(y+j)=\texttt{unmap}(y)$ and $S[y+j]\le\beta$. Once we have reported all the pairs corresponding to object $\texttt{unmap}(y)$, we can obtain those of the next objects by repeating the procedure from $\texttt{rel\_acc\_obj\_maj}(\alpha,\beta,\texttt{unmap}(y)+1)$.
• $\texttt{lab\_num}(\alpha,\beta,x,y)$ **in** $O(\beta-\alpha+\log\sigma)$ **time.** After mapping x and y to positions in S, we descend in the wavelet tree to find all the leaves in $[\alpha,\beta]$ while remapping $[x,y]$ appropriately. We count one more label each time we arrive at a leaf, and we stop descending from an internal node if its range $[x,y]$ is empty.
• $\texttt{obj\_sel1}(\alpha,x,r)$ **in** $O(\log\sigma)$ **time:** This is a matter of selecting the r-th occurence of the label α in S, after the position of the pair (α,x). The formula is $\texttt{unmap}(\texttt{select}_\alpha(S, r+\texttt{obj\_rnk1}(\alpha,x-1)))$.

The overall result is stated in the next theorem and illustrated in Fig. 2.

Theorem 2. *There is a representation for a binary relation $\mathcal{B}$, of t pairs over $[1,\sigma]\times[1,n]$, using $t\log\sigma + o(t)\log\sigma + O(\min(t, n\log(t/n)))$ bits of space. The structure supports operations $\texttt{rel\_rnk}(\alpha,x)$, $\texttt{rel\_sel\_lab\_maj}(\alpha,r,x,y)$, $\texttt{rel\_sel\_obj\_maj}(1,\sigma,x,r)$ (note the limitation), $\texttt{rel\_acc\_obj\_maj}(\alpha,\beta,x)$, and $\texttt{obj\_sel1}(\alpha,x,r)$, in time $O(\log\sigma)$, plus $\texttt{rel\_sel\_obj\_maj}(\alpha,\beta,x,r)$ in time $O(\min(\log n, \log r\log(\beta-\alpha+1))\log\sigma)$, and $\texttt{lab\_num}(\alpha,\beta,x,y)$ in time $O(\beta-\alpha+\log\sigma)$. This yields the support for other operations via the reductions from Thm. 1.*

Proof. The operations have been obtained throughout the section. For the space, B contains n 1s out of $n+t$, so a compressed representation [20] requires $O(n\log\frac{n+t}{n}) = O(\min(t, n\log(t/n)))$. The wavelet tree for $S[1,t]$ requires $t\log\sigma + o(t)\log\sigma$ bits of space. □

Note that the particular case $\texttt{rel\_num}(1,\sigma,x,y)$ can be answered in $O(1)$ time using B's succinct encoding. In general the space result is incomparable with $tH(\mathcal{B})$: if all the $n\sigma$ pairs are related,

then $tH_0(S) = n\sigma \log \sigma$ and $H(\mathcal{B}) = 0$; but if all the pairs are within a row, then $tH_0(S) = 0$ and $H(\mathcal{B}) > 0$. In the particular case where $t \leq n$, $t \log \sigma \leq tH(\mathcal{B}) + O(t)$, while the wavelet tree for S requires $tH_0(S) \leq t \log \sigma$ bits: this difference can be relevant depending on the distribution of pairs across the rows.

5 Binary Relation Wavelet Trees (BRWT)

We propose now a special wavelet tree structure to represent binary relations. This wavelet tree contains two bitmaps per level at each node v, B_v^l and B_v^r. At the root, $B_v^l[1, n]$ has the x-th bit set to 1 iff there exists a pair of the form (α, x) for $\alpha \in [1, \lfloor \sigma/2 \rfloor]$, and B_v^r has the x-th bit set to 1 iff there exists a pair of the form (α, x) for $\alpha \in [\lfloor \sigma/2 \rfloor + 1, \sigma]$. Left and right subtrees are recursively built on the positions set to 1 in B_v^l and B_v^r, respectively. The leaves (where no bitmap is stored) correspond to individual rows of the relation. We store a bitmap $B[1, n + t]$ recording in unary the number of elements in each row. See Fig. 1. Let us define constant-time functions $\texttt{lab}(r) = 1 + \texttt{rank}_0(B, \texttt{select}_1(B, r))$ and $\texttt{poslab}(\alpha) = \texttt{rank}_1(B, \texttt{select}_0(B, \alpha))$ on B.

Note that, because an object x may propagate both left and right, the sizes of the second-level bitmaps may add up to more than n bits. Indeed, the last level contains t bits and represents all the pairs sorted in row-major order.

The following operations can be carried out efficiently on this structure.

• $\texttt{rel\_num}(\alpha, \beta, x, y)$ **in** $O(\beta - \alpha + \log \sigma)$ **time.** We project the interval $[x, y]$ from the root to each leaf in $[\alpha, \beta]$, adding up the resulting interval sizes at leaves. Of course we can stop earlier if the interval becomes empty. Note that we can only count pairs at the leaves. In the case $[x, y] = [1, n]$ we can achieve $O(1)$ time, as the answer is simply $\texttt{poslab}(\beta) - \texttt{poslab}(\alpha - 1)$. Note this allows solving the restricted case $\texttt{rel\_rnk\_lab\_maj}(1, n, \alpha, z)$ in $O(\log \sigma)$ time.

• $\texttt{rel\_sel\_lab\_maj}(\alpha, r, 1, n)$ **in** $O(\log \sigma)$ **time.** Let $r' \leftarrow r + \texttt{poslab}(\alpha - 1)$ and $\beta \leftarrow \texttt{lab}(r')$, thus β is the row where the answer is. Now we start at position $y = r' - \texttt{poslab}(\beta - 1)$ in leaf β and walk the wavelet tree upwards while mapping $y \leftarrow \texttt{select}_1(B_v^l, y)$ or $y \leftarrow \texttt{select}_1(B_v^r, y)$, depending on whether we are left or right child of our parent v, respectively. When we reach the root, the answer is (β, y). Note we are only solving the particular case $[x, y] = [1, n]$.

• $\texttt{rel\_acc\_lab\_maj}(\alpha, x, y)$ **in** $O(\log \sigma)$ **time per pair output.** Map $[x, y]$ from the root to each leaf in $[\alpha, \sigma]$, abandoning a path when $[x, y]$ becomes empty. (Because left and right child cannot become simultaneously empty, the total amount of work is proportional to the number of leaves that contain pairs to report.) Now, for each leaf γ arrived at with interval $[x', y']$, map each $z' \in [x', y']$ up to the root, to discover the associated object z, and return (γ, z).

• $\texttt{rel\_acc\_obj\_maj}(\alpha, \beta, x)$ **in** $O(\log \sigma)$ **time per pair output.** Just as in Section 4, we cover $[\alpha, \beta]$ with $O(\log \sigma)$ wavelet tree nodes $v_1, v_2, \ldots$, and map x to x_i at each such v_i, all in $O(\log \sigma)$ time. Now, in the way back of this recursion, we obtain the next $y \geq x$ in the root associated to some label in $[\alpha, \beta]$, by following a process analogous to that for $\texttt{rel\_acc\_obj\_maj}$ in Section 4. Finally, we start from position $y' = y$ at the root v and report all the pairs related to y: Recursively, we descend left if $B_v^l[y'] = 1$, and then right if $B_v^r[y'] = 1$, remapping y' appropriately at each step, and keeping within the interval $[\alpha, \beta]$. Upon reaching each leaf γ we report (γ, y). Then we continue from $\texttt{rel\_acc\_obj\_maj}(\alpha, \beta, y + 1)$.

• $\texttt{lab\_num}(\alpha, \beta, x, y)$ **in** $O(\beta - \alpha + \log \sigma)$ **time.** Map $[x, y]$ from the root to each leaf in $[\alpha, \beta]$, adding one per leaf where the interval is nonempty. Recursion can stop when $[x, y]$ becomes empty.

• $\texttt{obj\_sel1}(\alpha, x, r)$ **in** $O(\log \sigma)$ **time.** Map $x - 1$ from the root to x' in leaf α, then walk upwards the path from $x' + r$ to the root and report the position obtained.

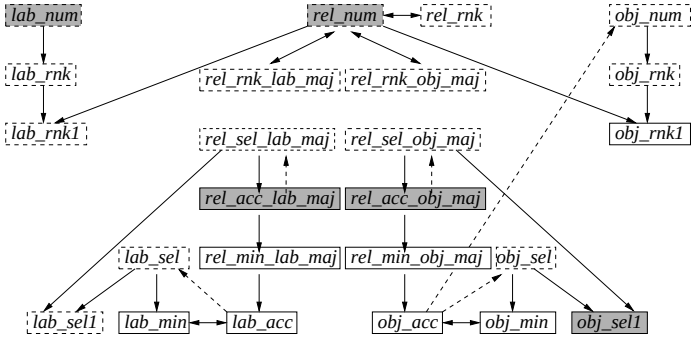

Fig. 3. Results achieved by the binary-relation wavelet tree. The nomenclature is as for Fig. 2.

We have obtained the following theorem, illustrated in Fig. 3 (we ignore the particular cases).

Theorem 3. *There is a representation for a binary relation $\mathcal{B}$, of t pairs over $[1,\sigma]\times[1,n]$, using $\log(1+\sqrt{2})tH(\mathcal{B}) + o(tH(\mathcal{B})) + O(t+n)$ bits of space. The structure supports operations* `rel_num`$(\alpha,\beta,1,n)$, `rel_rnk_lab_maj`$(1,n,\alpha,z)$, `rel_sel_lab_maj`$(\alpha,r,1,n)$ *(note the limitations of these three),* `rel_acc_lab_maj`(α,x,y), `rel_acc_obj_maj`(α,β,x), *and* `obj_sel1`(α,x,y), *in time $O(\log\sigma)$, plus* `rel_num`(α,β,x,y) *and* `lab_num`(α,β,x,y) *in time $O(\beta-\alpha+\log\sigma)$. This yields the support for other operations via the reductions from Thm. 1.*

Proof. The operations have been obtained throughout the section. For the space, B contains n 1s out of $n+t$, so a compressed representation [20] requires $O(n\log\frac{n+t}{n}) = O(\min(t, n\log(t/n)))$. The space of the wavelet tree can be counted as follows. Except for the $2n$ bits in the root, each other bit is induced by the presence of a pair. Each pair has a unique representative bit in a leaf, and also induces the presence of bits up to the root. Yet those leaf-to-root paths get merged, so that not all those bits are different. Consider an element x related to t_x labels. It induces t_x bits at t_x leaves, and their paths of bits towards the single x at the root. At worst, all the $O(t_x)$ bits up to level $\log t_x$ are created for these elements, and from there on all the t_x paths are different, adding up a total of $O(t_x)+t_x\log\frac{\sigma}{t_x}$. Adding over all x we get $O(t)+\sum_x t_x\log\frac{\sigma}{t_x}$. This is maximized when $t_x = t/n$ for all x, yielding $O(t)+t\log\frac{\sigma n}{t} = tH(\mathcal{B})+O(t)$ bits.

Instead of representing two bitmaps (which would multiply the above value by 2), we can represent a single sequence B_v with the possible values of the two bits at each position, 00, 01, 10, 11. Only at the root 00 is possible. Except for those $2n$ bits, we can represent the sequence over an alphabet of size 3 using the representation from Ferragina et al. [14], to achieve at worst $(\log 3)tH(\mathcal{B})+o(tH(\mathcal{B}))$ bits for this part while retaining constant-time `rank` and `select` over each B_v^l and B_v^r. (To achieve this, we maintain the directories for the original bitmaps, of sublinear-size.)

To improve the constant $\log 3$ to $\log(1+\sqrt{2})$, we consider that the representation by Ferragina et al. actually achieves $|B_v|H_0(B_v)$ bits. We call $\ell_x = |B_v| \le t_x$ and $H_x = |B_v|H_0(B_v)$. After level $\log t_x$, there is space to put all the t_x bits separately, thus using only 01 and 10 symbols we achieve $\ell_x = t_x$ and $H_x = t_x$ bits. Yet, this is not the worst that can happen. H_x can be increased by collapsing some 01's and 10's into 11's (thus reducing ℓ_x). Note that collapsing further 01's or 10's or 11's with 11's effectively removes one symbol from B_v, which cannot increase H_x, thus we do not consider these. Assume the t_x bits are partitioned into t_{01} 01's, t_{10} 10's, and t_{11} 11's, so that $t_x = t_{01}+t_{10}+2t_{11}$, $\ell_x = t_{01}+t_{10}+t_{11}$, and $H_x = t_{01}\log\frac{\ell_x}{t_{01}} + t_{10}\log\frac{\ell_x}{t_{10}} + t_{11}\log\frac{\ell_x}{t_{11}}$. As $t_{11} = (t_x - t_{01} - t_{10})/2$, the maximum of H_x as a function of t_{01} and t_{10} yields the worst case at

$t_{01} = t_{10} = \frac{\sqrt{2}}{4} t_x$, so $t_{11} = (\frac{1}{2} - \frac{\sqrt{2}}{4}) t_x$ and $\ell_x = (\frac{1}{2} + \frac{\sqrt{2}}{4}) t_x$, where $H_x = \log(1+\sqrt{2}) t_x$ bits. This can be achieved separately at each level. Using the same distribution of 01's, 10's, and 11's for all x we add up to $(1+\sqrt{2}) t \log \frac{\sigma n}{t} + O(t)$ bits. □

Note that this is a factor of $\log(1+\sqrt{2}) \approx 1.272$ away of the entropy of $\mathcal{B}$. On the other hand, it is actually better if the t_x do not distribute uniformly.

6 Exploiting Regularities

Real-life binary relations exhibit regularities that permit compressing them far more than to $tH(\mathcal{B})$ bits. For example, social networks, Web graphs, and inverted indexes follow well-known properties such as clustering of the matrix, uneven distribution of 1s across rows and/or columns, similarity across rows and/or columns, etc. [6, 1, 9].

The space $tH_0(S)$ achieved in Thm. 2 can indeed be improved upon certain regularities. The wavelet tree of S, when bitmaps are compressed with local encoding methods [20], achieves locality in the entropy [18]. That is, if $S = S_1 S_2 \ldots S_n$ then the space achieved is $\sum_x |S_x| H_0(S_x) + O(n \log t)$. In particular, if S_x corresponds to the labels related to object x, then the space will benefit from *clustering* in the binary relation: If each object is related only to a small subset of labels, then its S_x will have a small alphabet and thus a small entropy. Alternatively, similar columns (albeit not rows) induce copies in string S. This is not captured by the zero-order entropy, but it is by grammar compression methods. Some have been exploited for graph compression [12].

The space formula in Thm. 3 can also be refined: If some objects are related to many labels and others to few, then $\sum_x t_x \log \frac{\sigma}{t_x}$ can be smaller than $tH(\mathcal{B})$. This second approach can be easily modified to exploit several other regularities. Imagine we represent bitmaps B_v^l and B_v^r separately, but instead of B_v^r we store $B'_v = B_v^l$ xor B_v^r, while keeping the original sublinear structures for `rank` and `select`. Any access to $O(\log n)$ contiguous bits in B_v^r is achieved in constant time under the RAM model by *xor*-ing B_v^l and B'_v.

The following regularities turn into a highly compressible B'_v, that is, one with few or many 0's: (1) Row-wise similarities between nearby rows, extremely common on Web graphs [6], yield an almost-all-zero B'_v; (2) (sub)relations that are actually permutations or strings, that is, with exactly one 1 per column, yield an almost-all-one B'_v. This second kind of (sub)relations are common in relational databases, for example when objects or labels are primary keys in the table.

As there exists no widely agreed-upon notion of entropy for binary relations that goes further than $\log \binom{n\sigma}{t}$, we show now some experiments on the performance of these representations on some real-life relations. We choose instances of three types of binary relations: (1) Web graphs, (2) social networks, (3) inverted indexes. In a Web graph, pages are nodes and hyperlinks are edges, thus the relation is between nodes. In a social network, nodes are actors such as persons, and edges represent interactions like friendship. In an inverted index, words are related to the documents where they appear. All these are applications where compressed representations are relevant to manipulate very large binary relations.

For (1), we downloaded two crawls from the WebGraph project [6], `http://law.dsi.unimi.it`. Crawl `EU` (2005) contains $n = \sigma = 862,664$ nodes and $t = 19,235,140$ edges. Crawl `Indochina` (2004) contains $n = \sigma = 7,414,866$ nodes and $t = 194,109,311$ edges. For (2), we downloaded a coauthorship graph from DBLP (`http://dblp.uni-trier.de/xml`), which is a symmetric relation, and kept the upper triangle of the symmetric matrix (this is reasonable because we are able to access

$\mathcal{B}$	$H(\mathcal{B})$	Gap	String	BRWT	$+xor$	Best Ad-Hoc
EU	16.68	5.52	12.57	7.72	6.87	4.38 (WebGraph)
Indochina	19.55	3.12	12.81	4.07	3.93	1.47 (WebGraph)
DBLP	18.52	6.18	15.97	13.54	11.67	21.9 (WebGraph)
FT	12.45	3.54	13.91	9.32	7.85	6.20 (Rice)

Table 2. Entropy and space consumption, in bits per pair, of different binary relation representations over relations from different applications. Ad-hoc representations have limited functionality.

the relation in both directions). The result contains $n = \sigma = 452,477$ authors and $t = 1,481,877$ coauthorships. For (3), we consider the relation FT, the inverted index for all of the Financial Times collections from TREC-4 (http://trec.nist.gov), converting the terms to lowercase. It relates $\sigma = 502,259$ terms with $n = 210,139$ documents, using $t = 51,290,320$ pairs.

Table 2 shows, for these relations $\mathcal{B}$, their entropy $H(\mathcal{B})$, their *gap complexity* (defined below), the space of the string representation of Section 4, the space of the BRWT representation of Section 5, and that using the *xor*-improvement described above. All spaces are measured in bits per pair of the relation.

The general entropy does not consider the regularities of the binary relation exploited by our data structures. The *gap complexity* is the sum of the logarithms of the consecutive differences of objects associated to each label. It is upper bounded by the entropy and gives a more refined measure that accounts for clustering in the matrix. The string representation of Section 4 already improves upon the entropy, but not much. Although it has more functionality, this representation requires significantly more space than the BRWT, which takes better advantage of regularities. Note, however, that for example Web graphs are much more amenable than the social network to exploiting such regularities, while the inverted index is in between. The *xor* improvement has a noticeable additional effect on the BRWT space, reducing it by about 5%–15%. Particularly on the Web graphs, this latter variant becomes close to the gap complexity.

The last column of the table shows the compression achieved by the best ad-hoc alternatives, which support a very restricted set of operations in sublinear time (namely, extracting all the labels associated to an object). The results for crawls Indochina and EU are the best reported in the WebGraph Project page, and they even break the gap complexity. For FT we measured the space required by Rice encoding of the differential inverted lists, plus pointers from the vocabulary to the sequence. This state-of-the-art in inverted indexes [21]. Finally, in absence of available software specifically targeted at compressing social networks, we tried WebGraph v. 1.7 (default parameters) on DBLP. As this is an undirected graph, we duplicate each edge $\{i, j\}$ as (i, j) and (j, i). This is not necessary on our representations, as we can extract direct and reverse neighbors. As it can be seen, our representations are by far the best in this case where no specific compressors exist.

7 Conclusions

Motivated by their many applications, we have proposed a rich set of primitives of interest in applications of binary relation data structures. We proposed/extended representations that achieve compressed space and logarithmic time for many of those operations, yet others remain a challenge. We have experimentally shown how these compression methods perform reasonably well on some real-life binary relations. The times we have achieved for most operations is $O(\log \sigma)$, where σ is the number of labels. These can probably be improved to $O(\frac{\log \sigma}{\log \log t})$ by using recent techniques on

multiary wavelet trees [7], which would reach the best results achieved with wavelet trees for much simpler problems [14]. Our representations allow dynamic variants, where new pairs and/or objects can be inserted in/deleted from the waveleet trees [18, 11]. Adding/removing labels, instead, is an open challenge, as it alters the wavelet tree shape. The space of our structures is close but does not reach the entropy of the binary relation, $H(\mathcal{B})$, in the worst theoretical case. An ambitious goal is to support all the operations we have defined in logarithmic time and within $H(\mathcal{B})(1+o(1))$ bits of space. A related issue is to define a finer notion of binary relation entropy that captures regularities that arise in real life, so as to express the space we achieve in terms of those finer measures.

Finally, there is no reason why our list of operations should be exclusive. For example, determining whether a pair is related in the *transitive closure* of $\mathcal{B}$ is relevant for many applications (e.g. ancestorship in trees, paths in graphs). Alternatively one could enrich the data itself, for example associating a *tag* to each object/label pair, so that one can not only ask for the tag of a pair but also find pairs with some tag range within a range of the relation, and so on. This extension has already found applications, e.g. [13]. Another extension is n-ary relations, which would more naturally capture joins in the relational model.

References

1. R. Baeza-Yates and G. Navarro. Modeling text databases. In *Recent Advances in Applied Probability*, pages 1–25. Springer, 2004.
2. J. Barbay, L. C. Aleardi, M. He, and J. I. Munro. Succinct representation of labeled graphs. In *ISAAC*, LNCS 2906, pages 575–584, 2007.
3. J. Barbay, A. Golynski, I. Munro, and S. Srinivasa Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theoretical Computer Science*, 387(3):284–297, 2007.
4. J. Barbay, M. He, I. Munro, and S. Srinivasa Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *SODA*, pages 680–689, 2007.
5. J. Barbay and G. Navarro. Compressed representations of permutations, and applications. In *STACS*, pages 111–122, 2009.
6. P. Boldi and S. Vigna. The WebGraph framework I: compression techniques. In *WWW*, pages 595–602, 2004.
7. P. Bose, M. He, A. Maheshwari, and P. Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *WADS*, 2009. To appear.
8. Y.-F. Chien, W.-K. Hon, R. Shah, and J. Vitter. Geometric Burrows-Wheeler transform: Linking range searching and text indexing. In *DCC*, pages 252–261, 2008.
9. F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.
10. D. Clark. *Compact Pat Trees*. PhD thesis, Univ. of Waterloo, Canada, 1996.
11. F. Claude. Compressed data structures for Web graphs. Master's thesis, University of Chile, 2008. Advisor: G. Navarro.
12. F. Claude and G. Navarro. A fast and compact Web graph representation. In *SPIRE*, LNCS 4726, pages 105–116, 2007.
13. F. Claude and G. Navarro. Self-indexed text compression using straight-line programs. In *MFCS*, LNCS 5734, pages 235–246, 2009.
14. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):article 20, 2007.
15. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.
16. J. Kärkkäinen. *Repetition-Based Text Indexing*. PhD thesis, Univ. of Helsinki, Finland, 1999.
17. V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007.
18. V. Mäkinen and G. Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Transactions on Algorithms*, 4(3):article 32, 2008. 38 pages.
19. G. Navarro. Indexing text using the Ziv-Lempel trie. *Journal of Discrete Algorithms*, 2(1):87–114, 2004.
20. R. Raman, V. Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.
21. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, 2nd edition, 1999.