Compact Binary Relation Representations with Rich Functionality

Jérémy Barbay[a,1], Francisco Claude[b,2], Gonzalo Navarro[a,1,3]

[a] *Department of Computer Science, University of Chile, Chile.*
{jbarbay,gnavarro}@dcc.uchile.cl
[b] *Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Chile.*
fclaude@recoded.cl

Abstract

Binary relations are an important abstraction arising in many data representation problems. The data structures proposed so far to represent them support just a few basic operations required to fit one particular application. We identify many of those operations arising in applications and generalize them into a wide set of desirable queries for a binary relation representation. We also identify reductions among those operations. We then introduce several novel binary relation representations, some simple and some quite sophisticated, that not only are space-efficient but also efficiently support a large subset of the desired queries.

Keywords: Binary Relations, Wavelet Trees, Compression, Compact Data Structures.

1. Introduction

Binary relations appear everywhere in Computer Science. Graphs, trees, inverted indexes, strings and permutations are just some examples. They also arise as a tool to complement existing data structures (such as trees [4]

An early version of this article appeared in *Proc. LATIN 2010.*

[1]Partially funded by Fondecyt grant 1-110066, Chile.

[2]Funded in part by Google U.S./Canada PhD Fellowship Program and David R. Cheriton Scholarships Program. Work partially done while the author was at the David R. Cheriton School of Computer Science, University of Waterloo, Canada.

[3]Partially funded by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F, Chile.

	1	2	3	4	5	6	7	8	9
A	.	.	1	.	.	.	.	.	.
B	.	.	.	.	.	1	1	.	.
C	.	.	.	1	.	1	.	1	.
D	.	1	.	.	.	.	.	.	.
E	1	.	.	1	1	.	.	.	.
F	.	.	.	.	.	.	.	.	1
G	.	.	.	.	1	.	1	.	.
H	1	1	.	.	.	.	.	.	.

Figure 1: An example of binary relation.

or graphs [1]) 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 start such a study in this article.

Let us say that a binary relation $\mathcal{R}$ relates *objects* in $[1, n]$ with *labels* in $[1, \sigma]$, containing t pairs out of the $n\sigma$ possible ones. We focus on space-efficient representations considering a simple entropy measure,

$$H(\mathcal{R}) \;=\; \lg \binom{n\sigma}{t} \;=\; t \lg \frac{n\sigma}{t} + O(t)$$

bits, which ignores any other possible regularity. Figure 1 illustrates a binary relation (we identify labels with rows and objects with columns henceforth).

Previous work focused on relatively basic primitives for binary relations: extract the list of all the labels associated with an object or of all the objects associated with a label (an operation called `access`), or extracting the j-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 [4] supports `rank`, `select` and `access` on the rows (labels) of the relation, for the purpose of supporting faster joins on labels. The idea is to write the labels of the pairs in object-major order and to operate on the resulting string plus some auxiliary data. This approach was extended to support more general operations needed for text indexing, such as giving labels to the pairs and querying them [17]. The first technique [4] was later refined [5] into a

scheme that allows one to compress the string while still supporting the basic operations on both labels and objects. The idea is to store auxiliary data on top of an arbitrary representation of the binary relation, which can thus be compressed. This was used to support labeled operations on planar and quasi-planar labeled graphs [1], such as finding the neighbors of a node that have a given label, determining whether two nodes are connected, listing the neighbors of a node in order, etc.

Ad-hoc compressed representations for inverted lists [38] and Web graphs [16] can also be considered as supporting binary relations. The idea here is to write the objects of the pairs in label-major order, and to support extracting substrings of the resulting string, that is, little more than `access` on labels. One can add support for `access` on objects by means of string `select` operations [15]. The string can be compressed by different means depending on the application. Typical operations supported in this way on Web graphs are counting and listing the nodes that are pointed from a node, or that point to a node, or even from/to ranges of nodes. On inverted lists one can list the documents where a term appears, the terms appearing in a document, and also extend those queries to ranges of terms (e.g., supporting on-the-fly stemming) and ranges of documents (e.g., supporting queries on subcollections).

In this paper we aim at settling the foundations of efficient compact data structures for binary relations. In particular, we address the following points:

- We define a large set of operations of relevance to binary relations, widely extending the classic set of `rank`, `select` and `access`. We give a number of reductions among operations in order to define a core set that allows one to efficiently support the extended set of operations.
- We explore the power of the reduction to string operators [4] when the operations supported on the string are limited to `rank`, `select` and `access`. This structure is called BinRel-Str, and we show it achieves interesting bounds only for a reduced set of operations.
- We show that a particular string representation, the wavelet tree [25], although not being the fastest one, provides native support for a much wider set of operations within logarithmic time. We call BinRel-WT this binary relation representation.
- We extend wavelet trees to generalized wavelet trees [19], and design new algorithms for various operations that take advantage of their

larger fan-out. As a result we speed up most of the operations within the same space. This structure is called BinRel-GWT.

- We present a new structure, *binary relation wavelet tree* (BRWT), that is tailored to represent binary relations. Although the BRWT gives weaker support to the operations, it is the only one that approaches the entropy space $H(\mathcal{R})$ within a multiplicative factor (of 1.272).
- At the end, we show how some of our results can be used to provide new representations of point grids within asymptotically optimal space, supporting various geometric operations in improved time.

For the sake of brevity, we aim at the simplest description of the operations, ignoring any practical improvement that does not make a difference in terms of asymptotic time complexity, or trivial extensions such as interchanging labels and objects to obtain other space/time tradeoffs.

2. Compact Data Structures for Sequences

Given a sequence S of length n, drawn from an alphabet Σ of size σ, we define the following queries (omitting S if clear from context):

- $\texttt{rank}_a(S, i)$ counts the occurrences of symbol $a \in \Sigma$ in $S[1, i]$.
- $\texttt{select}_a(S, j)$ finds the position of the j-th occurrence of symbol $a \in \Sigma$ in S.
- $\texttt{access}(S, i) = S[i]$.

The solutions in the literature are quite different depending on the alphabet size, σ.

2.1. Binary Sequences

For the special case $\Sigma = \{0, 1\}$, there exist representations using $o(n)$ bits on top of a plain representation of S, and answering the three queries in constant time [14]. The extra space can be made as low as $O(n \lg\lg n / \lg n)$, which is optimal [22]. This extra data structure on top of a plain representation of S is called an *index*.

If we are allowed, instead, to represent S in a specific way, then one can *compress* the sequence while still supporting the three operations in constant

time. The overall space can be reduced to $nH_0(S) + o(n)$ bits [37]. Here $H_0(S)$ is the *zero-order entropy* of sequence S, defined as

$$H_0(S) \quad = \quad \sum_{a \in \Sigma} \frac{n_a}{n} \lg \frac{n}{n_a},$$

where n_a is the number of occurrences of symbol a in S. The $o(n)$ extra space on top of the entropy can be made as small as $O(n/\lg^c n)$ for any constant c [36].

2.2. Sequences over Small Alphabets

If $\sigma = O(\lg^\epsilon n)$, for a constant $0 < \epsilon < 1$, it is still possible to retain the space and time complexities of binary sequence representations [19]. The main idea is that we only need a compressed sequence representation that provides constant-time `access`, or more precisely, that can access $O(\lg_\sigma n)$ consecutive symbols from S in constant time (this is achieved by extending similar compressed bitmap representations [37]).

Then, in order to support $\texttt{rank}_a$ and $\texttt{select}_a$, we act as if we had an explicit bitmap $B_a[1, n]$, where $B_a[i] = 1$ iff $S[i] = a$. Then $\texttt{rank}_a(S, i) = \texttt{rank}_1(B_a, i)$ and $\texttt{select}_a(S, j) = \texttt{select}_1(B_a, j)$. We store only the `rank`/`select` *index* of each B_a, and can solve `rank`/`select` on B_a by extracting any desired chunk from S. The only difference is that we cannot access $O(\lg n)$ contiguous bits from B_a in constant time, but only $O(\lg_\sigma n)$.

As a result, the index for each B_a requires $O(n \lg\lg n/\lg_\sigma n)$ bits of space. Added over all the $a \in \Sigma$, the total space for the indexes is $O(n\sigma \lg\lg n/\lg_\sigma n) = o(n)$.

2.3. General Sequences and Wavelet Trees

The wavelet tree [25] reduces the three operations on 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 $\lceil \lg \sigma \rceil$, 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,

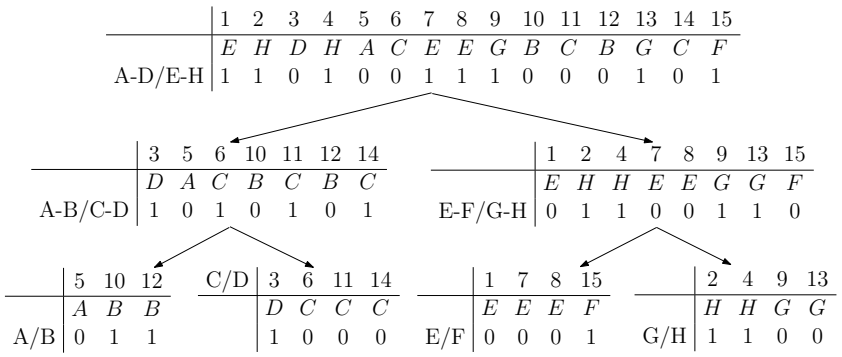

Figure 2: Example of a wavelet tree for the sequence `EHDHACEEGBCBGCF`.

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

If the bitmaps B_v are represented in plain form (with indexes that support binary `rank`/`select`), then the wavelet tree requires $n \lg \sigma + o(n) \lg \sigma$ bits of space, while answering all the queries in $O(\lg \sigma)$ time. If the bitmaps B_v are instead represented in compressed form [36], the wavelet tree uses $nH_0(S) + o(n)$ bits and retains the same time complexities. Figure 2 illustrates the structure. Wavelet trees are not only used to represent strings [19], but also grids [9], permutations [7], and many other structures. We refer in particular to a recent article [21] where in particular some (folklore) capabilities are carefully proved: any range of symbols $[\alpha, \beta]$ is covered by $k = O(\lg(\beta - \alpha + 1))$ wavelet tree nodes, and these can be reached from the root by traversing $O(k + \lg \sigma)$ nodes.

A way to speed up the wavelet tree operations is to use *generalized* wavelet trees [19]. These are multiary wavelet trees, with arity $\mu = O(\lg^\epsilon n)$, for a constant $0 < \epsilon < 1$. Bitmaps B_v are replaced by sequences S_v over a (small) alphabet of size μ. All the operations on those sequences are still solved

in constant time, but now the wavelet tree height is reduced to $O(\lg_\mu \sigma) = O(\lg \sigma / \lg \lg n)$, and thus this is the complexity of the operations. The space can still be bounded by $nH_0(S) + o(n)$ [24].

Wavelet trees are not the only sequence representation achieving basically $nH_0(S)$ bits of space [3, 23]. The best current alternative [3] uses $nH_0(S) + o(nH_0(S)) + o(n)$ bits and solves the three queries within time $O(\lg \lg \sigma)$. This is preferable when σ is large.

2.4. Range Minimum Queries (RMQs)

The Range Minimum Query (RMQ) operation on a sequence $S[1, n]$ was not listed among the basic ones, but we cover it because we will make use of it in Lemma 11. It is defined as $\textsc{rmq}(S, i, j) = \operatorname{argmin}_{i \le k \le j} S[k]$, that is, it returns the position of the minimum value in a range $S[i, j]$. If there are more than one, it returns the leftmost minimum.

It is possible to solve this query in constant time using just $2n + o(n)$ bits of space, without even accessing S itself [20].

3. Operations

3.1. Definition of operations

We now motivate the set of operations we define for binary relations. Their full list, formal definition, and illustration, are given in Appendix A.

One of the most pervasive examples of binary relations are directed graphs, which are precisely binary relations between a vertex set V and itself. Extracting rows or columns in this binary relation supports direct and reverse navigation from a node. To support powerful direct access to rows and columns we define operations $\texttt{obj\_acc1}(\alpha, x, y)$, which retrieves the objects in $[x, y]$ related to label α, in arbitrary order, and the symmetric one, $\texttt{lab\_acc1}(\alpha, \beta, x)$. In case we want to retrieve the pairs in order, $\texttt{obj\_min1}(\alpha, x)$, which gives the first object $\ge x$ related to label α, can be used as an iterator, and similarly $\texttt{lab\_min1}(\alpha, x)$. These operations are also useful to find out whether the link (α, x) exists. Note that adjacency list representations only support efficiently the retrieval of direct neighbors, and adjacency matrix only support efficiently the test for the existence of a link.

Web graphs, and their compact representation supporting navigation, have been a subject of intense research in recent years [8, 10, 16] (see many more references therein). In a Web graph, the nodes are Web pages and the edges are hyperlinks. Nodes are usually sorted by URL, which not only

gives good compression but also makes ranges of nodes correspond to domains and subdirectories[4]. For example, counting the number of connections between two ranges of nodes allows estimating the connectivity between two domains. This count of points in a range is supported by our operation `rel_num`(α, β, x, y), which counts the number of related pairs in $[\alpha, \beta] \times [x, y]$. The individual links between the two domains can be retrieved, in arbitrary order, with operation `rel_acc`(α, β, x, y). In general, considering domain ranges enables the analysis and navigation of the Web graph at a coarser granularity (e.g., as a graph of hosts, or institutions). Our operations `obj_acc`(α, β, x, y), which gives the objects in $[x, y]$ related to a label in $[\alpha, \beta]$, extends `obj_acc1` to ranges of labels, and similarly `lab_acc`(α, β, x, y) extends `lab_acc1`. Ordered enumeration and (coarse) link testing are supported by operations `obj_min`(α, β, x), which gives the first object $\geq x$ related to a label in $[\alpha, \beta]$, and similarly `lab_min`(α, x, y) for labels.

A second pervasive example of a binary relation are the two-dimensional grids, where objects and labels are simply coordinates, and where pairs of the relation are points at those coordinates. Grids arise in Geographic Information Systems and many other geometric applications. Operation `rel_num` allows us to count the number of points in a rectangular area. A second essential operation in these applications is to retrieve the points from such an area. If the retrieval order is not important, `rel_acc` is sufficient. Otherwise, operation `rel_acc_lab_maj`(x, y, α, z) serves as an iterator to retrieve the points in label-major order. It retrieves the first point in $[\alpha, \alpha] \times [z, y]$, if any, and otherwise the first point in $[\alpha{+}1, \sigma] \times [x, y]$. Operation `rel_acc_obj_maj`$(\alpha, \beta, \gamma, x)$ is similar, for object-major order. For an even more sophisticated processing of the points, `rel_sel_lab_maj`(α, j, x, y) and `rel_sel_obj_maj`(α, β, x, j) give access to the j-th element in such lists.

Grids also arise in more abstract scenarios. For example, several text indexing data structures [13, 17, 27, 29] resort to a grid, which relates for example text suffixes (in lexicographic order) with their text positions, or phrase prefixes and suffixes in Lempel-Ziv compression, or two labels that form a rule in grammar-based compression, etc. The operations most commonly needed are, again, counting and retrieving (in arbitrary order) the points in a rectangle.

Another important example of binary relations are inverted indexes [38],

[4]More precisely, we have to sort by the reversed site string concatenated with the path.

which support word-based searches on natural language text collections. Inverted indexes 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 (`obj_acc1`), a popular operation is the conjunctive query (e.g., in Google-like search engines), which retrieves the documents where k given words appear. These are solved using a combination of the complementary queries `obj_rnk1`(α, x) and `obj_sel1`(α, x, j) [4, 6]. The first operation counts the number of points in $[\alpha, \alpha] \times [1, x]$, whereas the second gives the j-th point in $[\alpha, \alpha] \times [x, n]$.

Extending these operations to a range of words allows for stemmed and/or prefix searches (by properly ordering the words), and are implemented using `obj_rnk`(α, β, x) and `obj_sel`(α, β, x, j), which extend `obj_rnk1` and `obj_sel1` to ranges of labels. Extracting a column, on the other hand (`lab_acc1`), gives important *summarization* information on a document: the list of its different words. Intersecting columns (using the symmetric operations `lab_rnk1`(α, x) and `lab_sel1`(α, x, j)) allows for analysis of content between documents (e.g., plagiarism or common authorship detection). Handling ranges of documents (supported with the symmetric operations `lab_acc`, `lab_rnk`(α, x, y), and `lab_sel`(α, j, x, y)) allows for considering hierarchical document structures such as XML or file systems (where one operates over a whole subtree or subdirectory).

Similar representations are useful to support join operations on relational databases and, in combination with data structures for ordinal trees, to support multi-labeled trees, such as those featured by semi-structured documents (e.g., XML) [4]. A similar technique [1] 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. For example, operations `rel_min_lab_maj` and `rel_min_obj_maj` support the search for the highest neighbor of a point, when the binary relation encodes the levels of points in a planar graph representing a topography map [1].

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. For example, in a relational database, consecutive labels may represent a range of parameter values (e.g., people of age between 20 and 40).

We define other operations for completeness: `rel_rnk_lab_maj` acts like the inverse of `rel_sel_lab_maj`, and similarly `rel_rnk_obj_maj`; `lab_num` and `obj_num` are more complete versions of `lab_rnk` and `obj_rnk`; and `rel_rnk` is

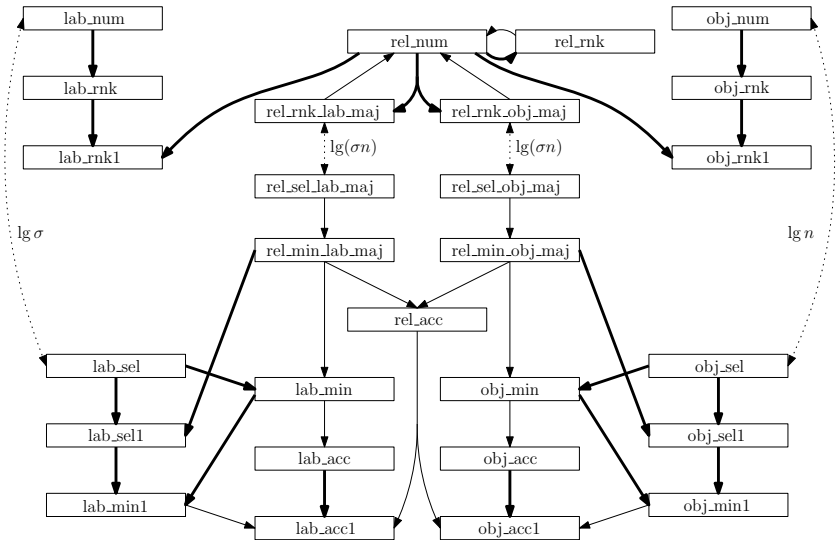

Figure 3: Reductions among operations.

a more basic version of `rel_num`.

3.2. Reductions among operations

We give a set of reductions among the operations introduced. The results are sumarized in the following theorem.

Theorem 1. *For any solid arrow $op \to op'$ in Figure 3, it holds that if op is solved in time t, then op' can be solved in time $O(t)$. For the dotted arrows with associated penalty factors $O(t')$, it holds that if op is solved in time t, then op' can be solved in time $O(tt')$.*

Proof. Several reductions are immediate from the definition of the operations in Appendix A (those arrows are in bold in Figure 3). We prove now the other ones. We consider only the reductions for the left side of the figure (operations related to labels); the same idea applies for the right side (objects).

- $\texttt{rel\_rnk} \to \texttt{rel\_num}$

$$\begin{aligned}\texttt{rel\_num}(\alpha,\beta,x,y) &= \texttt{rel\_rnk}(\alpha-1,x-1)-\texttt{rel\_rnk}(\alpha-1,y)\\ &\quad -\texttt{rel\_rnk}(\beta,x-1)+\texttt{rel\_rnk}(\alpha-1,x-1).\end{aligned}$$

- $\texttt{rel\_rnk\_lab\_maj} \rightarrow \texttt{rel\_num}$

$$\begin{aligned}\texttt{rel\_num}(\alpha, \beta, x, y) &= \texttt{rel\_rnk\_lab\_maj}(\beta, x, y, y) \\ &\quad -\texttt{rel\_rnk\_lab\_maj}(\alpha - 1, x, y, y).\end{aligned}$$

- $\texttt{rel\_acc} \rightarrow (\texttt{lab\_acc1}, \texttt{obj\_acc1})$

$$\begin{aligned}\texttt{lab\_acc1}(\alpha, \beta, x) &= \{\gamma,\ (\gamma, x) \in \texttt{rel\_acc}(\alpha, \beta, x, x)\}, \\ \texttt{obj\_acc1}(\alpha, x, y) &= \{z,\ (\alpha, z) \in \texttt{rel\_acc}(\alpha, \alpha, x, y)\}.\end{aligned}$$

- $\texttt{rel\_sel\_lab\_maj} \rightarrow \texttt{rel\_min\_lab\_maj}$: in order to solve query $\texttt{rel\_min\_lab\_maj}(\alpha, x, y, z)$ we first test if $\texttt{rel\_sel\_lab\_maj}(\alpha, 1, z, y)$ gives a pair of the form (α, w), in which case we return it. Otherwise, we return $\texttt{rel\_sel\_lab\_maj}(\alpha + 1, 1, x, y)$.

- $\texttt{rel\_min\_lab\_maj} \rightarrow \texttt{rel\_acc}$: to solve $\texttt{rel\_acc}(\alpha, \beta, x, y)$, we find a first point $(\gamma, z) = \texttt{rel\_min\_lab\_maj}(\alpha, x, y, x)$. The next element is obtained as $(\gamma', z') = \texttt{rel\_min\_lab\_maj}(\gamma, x, y, z + 1)$ and so on, until we reach the first answer with label greater than β.

- $\texttt{rel\_min\_lab\_maj} \rightarrow \texttt{lab\_min}$: let $(\gamma, z) = \texttt{rel\_min\_lab\_maj}(\alpha, x, y, x)$, then $\texttt{lab\_min}(\alpha, x, y) = \gamma$.

- $\texttt{lab\_min} \rightarrow \texttt{lab\_acc}$: we report $\gamma = \texttt{lab\_min}(\alpha, x, y)$, $\gamma' = \texttt{lab\_min}(\gamma + 1, x, y)$, and so on until reaching a result larger than β. The points reported form $\texttt{lab\_acc}(\alpha, \beta, x, y)$.

- $\texttt{lab\_min1} \rightarrow \texttt{lab\_acc1}$: similar to the previous reduction.

Finally, the non-constant time reductions are explained the following way:

- $(\texttt{rel\_rnk\_lab\_maj}, \texttt{rel\_sel\_lab\_maj})$ works in both ways by doing a binary search over the results of the other operation, in the worst case considering $n\sigma$ elements.

- $(\texttt{lab\_num}, \texttt{lab\_sel})$ operates the same way as the previous one, but searching among σ elements in the worst case, thus the $O(\lg \sigma)$ penalty. (For the objects this becomes $O(\lg n)$.)

□

The reductions presented here allow us to focus on a small subset of the most difficult operations. In some cases, however, we will present more efficient solutions for the simpler operations and will not use the reduction.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	E	H	D	H	A	C	E	E	G	B	C	B	G	C	F
B	1	10	1	10	10	1	10	1	10	1	10	1	10	10	10

Figure 4: Sequence S and bitmap B for representing the binary relation shown in Figure 1.

4. Reduction to Strings: BinRel-Str

A simple representation [4, 17] for a binary relation $\mathcal{R}$ formed by t pairs in $[1, n] \times [1, \sigma]$ uses a bitmap $B[1, n + t]$ and a string $S[1, t]$ over the alphabet $[1, \sigma]$. The bitmap B 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. Figure 4 shows the representation for the binary relation shown in Figure 1. Barbay et al. [4] showed that an easy way to support operations `obj_rnk1` and `obj_sel1` on the binary relation is to support the operations `rank` and `select` on B and S, using any data structure known for bitmaps and strings (recall Section 2). Note also that the particular case $\texttt{rel\_num}(1, \sigma, x, y)$ can be answered in $O(1)$ time using B. In the sequel we extend Barbay et al.'s work as much as possible considering our considerably larger set of operations. This approach, building only on `rank`, `select` and `access` on B and S, will be called BinRel-Str.

We define some notation used in the rest of the paper. First, we call $\texttt{map}(x)$ the mapping from a column number x to its last element in S: $\texttt{map}(x) = \texttt{rank}_1(B, \texttt{select}_0(B, x))$. The inverse, from a position in S to its column number, is called $\texttt{unmap}(m) = \texttt{rank}_0(B, \texttt{select}_1(B, m)) + 1$. Both mappings take constant time. Finally, let us also define for brevity $\texttt{rank}_c(B, x, y) = \texttt{rank}_c(B, y) - \texttt{rank}_c(B, x - 1)$.

Assume our representation of S supports `access` in time a, `rank` in time r and `select` in time s. Table 1 (column 2) shows the complexity achieved for each binary relation operation using this approach. As it can be seen, the scheme extends nicely only to operations involving one row or one column. In all the other cases, the complexities are linear in the lengths of the ranges to consider. As the algorithms are straightforward and their complexities uninteresting, we defer them to Appendix B.[5]

[5]To simplify, in column 2 of Table 1 we omit some complexities that are most likely to be inferior to their alternatives.

Operation \ Space	BinRel-Str ($t \lg \sigma(1+o(1))$)	BinRel-WT ($t \lg \sigma$)	BinRel-GWT ($t \lg \sigma(1+o(1))$)	BRWT ($\lg(1+\sqrt{2})H(\mathcal{R})$)
`rel_num`(α, β, x, y)	$O((\beta-\alpha+1)r)$ or $O((y-x+1)a \lg \beta)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\beta-\alpha+\lg \sigma)$
`rel_rnk`(α, x)	$O(\alpha r)$ or $O(xa \lg \alpha)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\alpha+\lg \sigma)$
`rel_rnk_lab_maj`(α, x, y, z)	$O(\alpha r)$ or $O((y-x+1)a \lg \alpha)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\alpha+\lg \sigma)$
`rel_sel_lab_maj`(α, j, x, y)	$O((\sigma-\alpha+1)r+s)$	$O(\lg \sigma)$	$O(\lg \sigma)$	$O(j \lg \sigma)$
`rel_min_lab_maj`(α, x, y, z)	$O((\sigma-\alpha+1)r+s)$ or $O((y-x+1)a \lg \alpha)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n+\lg \lg n)$	$O(\lg \sigma)$
`rel_rnk_obj_maj`$(\alpha, \beta, \gamma, x)$	$O((\beta-\alpha+1)r)$ or $O(xa \lg \beta)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\beta-\alpha+\lg \sigma)$
`rel_sel_obj_maj`(α, β, x, j)	$O((n-x+1)a \lg \beta)$	$O(\lg j \lg(\beta-\alpha+1) \lg \sigma)$ or $O(\lg n \lg \sigma)$	$O(\lg j \lg(\beta-\alpha+1) \lg \sigma / \lg \lg n)$ or $O(\lg n \lg \sigma / \lg \lg n)$	$O(j \lg \sigma)$
`rel_min_obj_maj`$(\alpha, \beta, \gamma, x)$	$O((\beta-\alpha+1)(s+r))$ or $O((n-x+1)a \lg \alpha)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\lg \sigma)$
`rel_acc`(α, β, x, y)	$O((\beta-\alpha+1)r+sk)$ or $O((y-x+1)a \lg \alpha+ak)$	$O((k+1) \lg \sigma)$	$O((k+1) \lg \sigma / \lg \lg n)$	$O((k+1) \lg \sigma)$
`lab_num`(α, β, x, y)	$O((\beta-\alpha+1)r)$	$O(\beta-\alpha+\lg \sigma)$	$O(\beta-\alpha+\lg \sigma / \lg \lg n)$	$O(\beta-\alpha+\lg \sigma)$
`lab_rnk`(α, x, y)	$O(\alpha r)$	$O(\alpha+\lg \sigma)$	$O(\alpha+\lg \sigma / \lg \lg n)$	$O(\alpha+\lg \sigma)$
`lab_sel`(α, j, x, y)	$O((\sigma-\alpha+1)r)$	$O(j \lg \sigma)$	$O(j(\lg \sigma / \lg \lg n+\lg \lg n))$	$O(j \lg \sigma)$
`lab_acc`(α, β, x, y)	$O((\beta-\alpha+1)r)$ or $O((y-x+1)a \lg \alpha)$	$O((k+1) \lg \sigma)$	$O((k+1)(\lg \sigma / \lg \lg n+\lg \lg n))$	$O((k+1) \lg \sigma)$
`lab_min`(α, x, y)	$O((\sigma-\alpha+1)r)$ or $O((y-x+1)a \lg \alpha)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n+\lg \lg n)$	$O(\lg \sigma)$
`obj_num`(α, β, x, y)	$O((y-x+1)a \lg \alpha)$	$O((y-x+1) \lg \sigma)$	$O((y-x+1) \lg \sigma / \lg \lg n)$	$O((y-x+1) \lg \sigma)$
`obj_rnk`(α, β, x)	$O(xa \lg \alpha)$	$O(x \lg \sigma)$	$O(x \lg \sigma / \lg \lg n)$	$O(x \lg \sigma)$
`obj_sel`(α, β, x, j)	$O((n-x+1)a \lg \alpha)$	$O(j \lg \sigma)$	$O(j \lg \sigma / \lg \lg n)$	$O(j \lg \sigma)$
`obj_acc`(α, β, x, y)	$O((\beta-\alpha+1)(r+sk))$ or $O((y-x+1)a \lg \alpha)$	$O((k+1) \lg \sigma)$	$O((k+1) \lg \sigma / \lg \lg n)$	$O((k+1) \lg \sigma)$
`obj_min`(α, β, x)	$O((\beta-\alpha+1)(r+s))$ or $O((n-x+1)a \lg \alpha)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\lg \sigma)$
`lab_rnk1`(α, x)	$O(a \lg \alpha)$, e.g., $O(\lg \sigma)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\lg \sigma)$
`lab_sel1`(α, j, x)	$O(a \lg \alpha)$, e.g., $O(\lg \sigma)$	$O(\lg \sigma)$	$O(\lg \sigma)$	$O(j \lg \sigma)$
`lab_min1`(α, x)	$O(a \lg \alpha)$, e.g., $O(\lg \sigma)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n+\lg \lg n)$	$O(\lg \sigma)$
`lab_acc1`(α, β, x)	$O(a(k+\lg \alpha))$, e.g., $O(k+\lg \sigma)$	$O((k+1) \lg \sigma)$	$O((k+1) \lg \sigma / \lg \lg n)$	$O((k+1) \lg \sigma)$
`obj_rnk1`(α, x)	$O(r)$, e.g., $O(\lg \lg \sigma)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\lg \sigma)$
`obj_sel1`(α, x, j)	$O(r+s)$, e.g., $O(\lg \lg \sigma)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\lg \sigma)$
`obj_min1`(α, x)	$O(r+s)$, e.g., $O(\lg \lg \sigma)$	$O(\lg \sigma)$	$O(\lg \sigma / \lg \lg n)$	$O(\lg \sigma)$
`obj_acc1`(α, x, y)	$O(r+sk)$, e.g., $O((k+1) \lg \lg \sigma)$	$O((k+1) \lg \sigma)$	$O((k+1) \lg \sigma / \lg \lg n)$	$O((k+1) \lg \sigma)$

Table 1: Space of our representations (in bits, omitting $O(n+t+\sigma)$ in general) and their time complexity for each operation. Parameter k is the output size for the `access` operators; one can consider $k = 1$ for the reductions given in Theorem 1.

Various string representations [3, 23] offer times a, r, and s that are constant or log-logarithmic on σ. These yield the best time complexities we know of for the row-wise and column-wise operations, although these form a rather limited subset of the operations we have defined. We exemplify this in Table 1 using Golynski et al.'s representation [23] (the variant with constant-time `access`) for the space and the few interesting operations.

The space used by techniques based on representing B and S (including BinRel-WT and BinRel-GWT) is usually unrelated to $H(\mathcal{R})$, the entropy of the binary relation. Various representations for $S[1,t]$ achieve space $tH_0(S)$ plus some redundancy [3, 23, 25]. This is $tH_0(S) = \sum_{\alpha\in[1,\sigma]} n_\alpha \lg \frac{t}{n_\alpha}$, where n_α is the number of pairs of the form $(\alpha,\cdot)$ in $\mathcal{R}$. While this can be lower than $H(\mathcal{R})$ (which shows that our measure $H(\mathcal{R})$ is rather crude), it can also be arbitrarily higher. For example an almost full binary relation has an entropy $H(\mathcal{R})$ close to zero, but its $tH_0(S)$ is close to $n\sigma\lg\sigma$. A clearer picture is obtained if we assume that S is represented in plain form using $t\lg\sigma$ bits. This is to be compared to $H(\mathcal{R}) = t\lg\frac{n\sigma}{t} + O(t)$, which shows that the string representation is competitive for sparse relations, $t = O(n)$.

5. Using Wavelet Trees: BinRel-WT

Among the many string representations of S we can choose in BinRel-Str scheme, wavelet trees [25] turn out to be particularly interesting. Although the time wavelet trees offer for a, r and s is $O(\lg\sigma)$, which is not the best available for large σ, wavelet trees allow one to support many more operations efficiently, via other algorithms than those used by the three basic operations. We call this representation BinRel-WT. Table 1 (column 3) summarizes the time complexity for each operation using BinRel-WT. Next, we show how to support some key operations efficiently; the other complexities are inferred from Theorem 1.

The first lemma states a well-known algorithm on wavelet trees [29].

Lemma 1. BinRel-WT *supports* $\texttt{rel\_rnk}(\alpha,x)$ *in* $O(\lg\sigma)$ *time.*

Proof. This is $\texttt{rank}_{\leq\alpha}(S,\texttt{map}(x))$, where operation $\texttt{rank}_{\leq\alpha}(S,p)$ counts the number of symbols $\leq\alpha$ in $S[1,p]$. It can be supported in time $O(\lg\sigma)$ on the wavelet tree of S by following the root-to-leaf branch corresponding to α, while counting at each node the number of objects preceding position p 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 $p \leftarrow rank_0(B_v, p)$. Otherwise, enter the right subtree with $c \leftarrow c + rank_0(B_v, p)$ and $p \leftarrow \texttt{rank}_1(B_v, p)$. Continue recursively until a leaf is reached (indeed, that of α), where the answer is $c + p$. □

The next lemma solves an extended variant of a query called *range_quantile* in the literature, which was also solved with wavelet trees within the same $O(\lg \sigma)$ time complexity [21]. The lemma also gives a solution via reductions for `lab_min`, within the same $O(\lg \sigma)$ time complexity. This was called *range_next_value* in the literature [21] and solved with an ad-hoc algorithm, within the same space and time. In the Conclusions we discuss the relation with faster recent solutions that, however, use more space [11, 34].

Lemma 2. BinRel-WT *supports* $\texttt{rel\_sel\_lab\_maj}(\alpha, j, x, y)$ *in* $O(\lg \sigma)$ *time.*

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

The wavelet tree is asymmetric with respect to objects and labels. The transposed problem, `rel_sel_obj_maj`, turns out to be harder. We present, however, a polylogarithmic-time solution.

Lemma 3. BinRel-WT *supports* $\texttt{rel\_sel\_obj\_maj}(\alpha, \beta, x, j)$ *in* $O(\min(\lg n, \lg j \lg(\beta - \alpha + 1)) \lg \sigma)$ *time.*

Proof. Remember that the elements are written in S in object major order. First, we note that the particular case where $[\alpha, \beta] = [1, \sigma]$ is easily solved in $O(\lg \sigma)$ time, by doing $j' \leftarrow j + \texttt{rel\_num}(1, \sigma, 1, x - 1)$ and returning $(S[j'], \texttt{unmap}(j'))$. In the general case, one can obtain time $O(\lg n \lg \sigma)$ by binary searching the column y such that $\texttt{rel\_num}(\alpha, \beta, x, y - 1) < j \leq$

$\texttt{rel\_num}(\alpha, \beta, x, y)$. Then the answer is $(\texttt{lab\_sel1}(\alpha, j - \texttt{rel\_num}(\alpha, \beta, x, y - 1), y), y)$ (note that Lemma 2 already gives us $\texttt{lab\_sel1}$ in time $O(\lg \sigma)$).

To obtain the other complexity, we find the $O(\lg(\beta - \alpha + 1))$ wavelet tree nodes that cover the interval $[\alpha, \beta]$; let these be $v_1, v_2, \ldots, v_k$. We map position $p = \texttt{map}(x - 1) + 1$ from the root towards those v_is, obtaining all the mapped positions p_i in $O(k + \lg \sigma)$ time [21]. Now the answer is within the positions $[p_i, p_i + j - 1]$ of some i. We cyclically take each v_i, choose the middle element of its interval, and map it towards the root, obtaining position q, corresponding to pair $(S[q], \texttt{unmap}(q))$. If $\texttt{rel\_rnk\_obj\_maj}(\alpha, \beta, S[q], \texttt{unmap}(q)) - \texttt{rel\_num}(\alpha, \beta, 1, x - 1) = j$, the answer is $(S[q], \texttt{unmap}(q))$. Otherwise we know whether q is before or after the answer. So we discard the left or right interval in v_i. After $O(k \lg j)$ such iterations we have reduced all the intervals of length j of all the nodes v_i, finding the answer. Each iteration costs $O(\lg \sigma)$ time. □

The next lemma solves a more general variant of a problem that was called *prevLess* in the literature, and also solved with wavelet trees [28] (again, we discuss in the Conclusions the relation with recent faster solutions that use more space [34]). We achieve the same complexity for this more general variant. Note this is a simplification of $\texttt{rel\_sel\_obj\_maj}$ that we can solve within time $O(\lg \sigma)$, whereas for general j we cannot.

Lemma 4. BinRel-WT *supports* $\texttt{rel\_min\_obj\_maj}(\alpha, \beta, \gamma, x)$ *in* $O(\lg \sigma)$ *time per pair output.*

Proof. We first use $\texttt{lab\_min1}(\gamma, x)$ (which we already can solve in time $O(\lg \sigma)$ as a consequence of Lemma 2) to search for a point in the band $[\gamma, \beta] \times [x, x]$. If we find one, then this is the answer, otherwise we continue with the area $[\alpha, \beta] \times [x + 1, n]$.

Just as for the second solution of Lemma 3, we obtain the positions p_i at the nodes v_i that cover $[\alpha, \beta]$. The first element to deliver is precisely one of those p_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 $q = p_i$. Otherwise, if the left child of v returned q, we map it to $q' \leftarrow \texttt{select}_0(B_v, q)$. Similarly, if the right child of v returned q, we map it to $q'' \leftarrow \texttt{select}_1(B_v, q)$. If we have only q' (q''), we return $q = q'$ ($q = q''$); if we have both we return $q = \min(q', q'')$. The process takes $O(\lg \sigma)$ time. When we arrive at the root we have the next position q where a label in $[\alpha, \beta]$ occurs in S, and thus return $\texttt{unmap}(q)$. □

The next result can also be obtained by considering the complexity of the BinRel-Str scheme implemented over a wavelet tree.

Lemma 5. BinRel-WT *supports* `obj_sel1`(α, x, j) *in* $O(\lg \sigma)$ *time.*

Proof. This is a matter of selecting the j-th occurence of the label α in S, after the position of the pair (α, x). The formula is `unmap`$(\texttt{select}_\alpha(S, j + \texttt{obj\_rnk1}(\alpha, x-1)))$. □

The next operation is the first of the set we cannot solve within polylogarithmic time.

Lemma 6. BinRel-WT *supports* `lab_num`(α, β, x, y) *in* $O(\beta - \alpha + \lg \sigma)$ *time.*

Proof. After mapping $[x, y]$ to positions $S[p, q]$, we descend in the wavelet tree to find all the leaves in $[\alpha, \beta]$ while remapping $[p, q]$ appropriately. We count one more label each time we arrive at a leaf, and we stop descending from an internal node if its range $[p, q]$ is empty. The complexity comes from the number of wavelet tree nodes accessed to reach such leaves [21]. □

The remaining operations are solved naively: `lab_sel` and `obj_sel` use, respectively, `lab_min` and `obj_min` successively, and `obj_num` and `obj_rnk` use `obj_rnk1` successively,

The overall result is stated in the next theorem and illustrated in Figure 5.

Theorem 2. *The structure* BinRel-WT*, for a binary relation $\mathcal{R}$ of t pairs over $[1, \sigma] \times [1, n]$, uses $t \lg \sigma + O(n + t)$ bits of space and supports the operations within the time complexities given in Table 1 (column 3).*

Proof. The space assumes a plain uncompressed wavelet tree and bitmap representations, and the time complexities have been obtained throughout the section. □

6. Using a Generalized Wavelet Tree: BinRel-GWT

The results we obtained for the wavelet tree can be extended to the generalized wavelet tree, improving complexities in many cases (recall Section 2.3). We refer to the structure that represents S using the generalized wavelet tree as BinRel-GWT, and we use μ to represent the fan-out of the tree. We

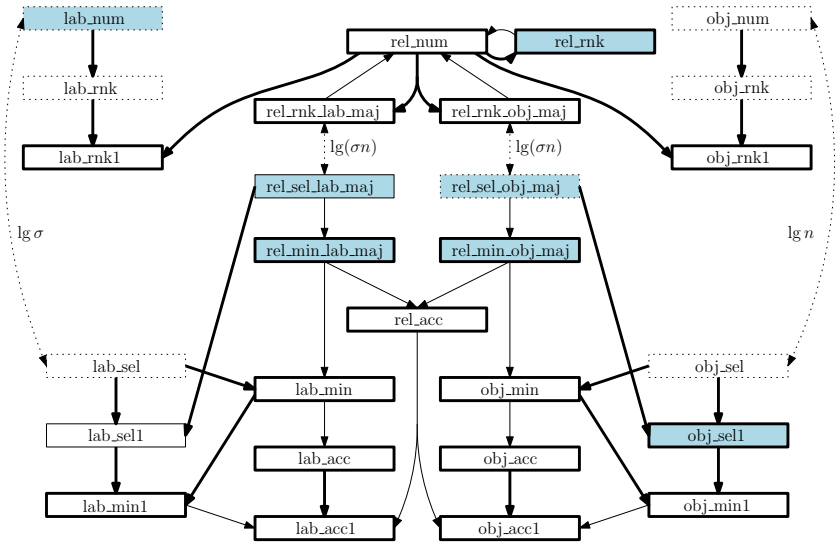

Figure 5: Reductions among operations supported by BinRel-WT and BinRel-GWT. The dotted boxes are operations supported in $\omega(\lg \sigma)$ time, the filled boxes represent operations directly addressed in the paper, and the blank boxes are supported via reductions. The solid boxes are solved in time $O(\lg \sigma)$, but BinRel-GWT achieves time $O(\lg_\mu \sigma)$ or $O(\lg_\mu \sigma + \lg \mu)$ in those with thick solid lines. Note that box `rel_min_lab_maj` is not directly addressed by BinRel-WT, but just by a reduction to `rel_sel_lab_maj`, whereas BinRel-GWT directly addresses it and obtains better time.

require $\mu \in O(\lg^\epsilon(n\sigma))$, assuming that the RAM machine can address up to $n \times \sigma$ cells. To simplify we will assume $\sigma \leq n$ and then use simply $\mu \in O(\lg^\epsilon n)$.

A first simple result stems from the faster string operations afforded by generalized wavelet trees (it also holds for `obj_rnk1`, but we obtain this one via reductions from `rel_rnk`).

Lemma 7. BinRel-GWT *supports* `obj_sel1` *in time* $O(\lg_\mu \sigma)$ *for any* $\mu \in \Theta(\lg^\epsilon n)$ *and any constant* $0 < \epsilon < 1$.

Proof. This follows directly from the results on general BinRel-Str structures. □

The following notation will be useful to describe our algorithms within wavelet tree nodes. Note that all are easily computed in constant time.

- $\texttt{child}(k)$: Given a symbol $k \in [1, \mu]$, this is the subtree labeled k of the current node.
- $\texttt{g}(\alpha)$: Given a symbol $\alpha \in [1, \sigma]$, $\texttt{g}(\alpha)$ is the symbol k such that $\texttt{child}(k)$ contains the leaf corresponding to α.
- $\texttt{g}^{-1}(k)$: Given a symbol $k \in [1, \mu]$, $\texttt{g}^{-1}(k) = \min\{\alpha, k = \texttt{g}(\alpha)\}$.

The next lemma shows how to speed up all the range counting operations. The result is known in the literature for $n \times n$ grids, even within $n \lg n(1+o(1))$ bits of space [9].

Lemma 8. BinRel-GWT *supports* $\texttt{rel\_rnk}(\alpha, x)$ *in* $O(\lg_\mu \sigma)$ *time, for any* $\mu \in \Theta(\lg^\epsilon n)$ *and any constant* $0 < \epsilon < 1$.

Proof. As in the case of BinRel-WT, we reduce this problem to the one of computing $\texttt{rank}_{\leq\alpha}(S, \texttt{map}(x))$, which can be done by following a similar procedure: We follow the path for α starting at the root in position $p = \texttt{map}(x)$ and with a counter $c \leftarrow 0$. Every time we move to a subtree, we increase $c \leftarrow c + \texttt{rank}_{\leq g(\alpha)-1}(S_v, p)$. When we arrive at the leaf, the answer is $c + p$.

Operation $\texttt{rank}_{\leq k}(S_v, p)$ can be solved in constant time for $\mu \in \Theta(\lg^\epsilon n)$ with a procedure similar to $\texttt{rank}_k$ on small alphabets (recall Section 2.2). We store for each $k \in [1, \mu]$ the index of a bitmap $B_{\leq k}$ such that $B_{\leq k}[i] = 1$ iff $S_v[i] \leq k$. Thus $\texttt{rank}_{\leq k}(S_v, p) = \texttt{rank}_1(B_{\leq k}, p)$ is computed in constant time and the whole process takes $O(\lg_\mu \sigma)$ time. □

The next lemma covers operation $\texttt{rel\_sel\_lab\_maj}$, on which we could not improve upon the complexity given by BinRel-WT.

Lemma 9. BinRel-GWT *supports* $\texttt{rel\_sel\_lab\_maj}(\alpha, j, x, y)$ *in* $O(\lg \sigma)$ *time.*

Proof. This is solved in a similar way to the one presented for BinRel-WT. We find v such that $\texttt{rank}_{\leq\beta-1}(S, p, q) < v \leq \texttt{rank}_{\leq\beta}(S, p, q)$. The only difference is that in this case we have to do, at each node, a binary search for the right child $k \in [1, \mu]$ to descend, and thus the time is $O(\lg \mu \lg_\mu \sigma) = O(\lg \sigma)$. □

For the next operations we will augment the generalized wavelet tree with a set of bitmaps inside each node v. More specifically, we will add $\mu(\mu+1)/2$ bitmaps $B_{k,l}$, where $B_{k,l}[i] = 1$ iff $S_v[i] \in [k,l]$. Just as with bitmaps $B_{\leq k}$, bitmaps $B_{k,l}$ are not represented explicitly, but only their index is stored (recall Section 2.2), and their content is simulated in constant time using S_v. Their total space for a sequence $S_v[1,n]$ is $O(n\mu^2 \lg\lg n/\lg_\mu n)$. To make this space $o(n)$, it is sufficient that $\mu = O(\lg^\epsilon n)$ for any constant $0 < \epsilon < 1/2$. (A related idea has been used by Farzan et al. [18].)

The next lemma shows that the current solution for operation *prevLess* [28] can be sped up by a $\Theta(\lg\lg n)$ factor within the same space.

Lemma 10. BinRel-GWT *supports* $\texttt{rel\_min\_obj\_maj}(\alpha,\beta,\gamma,x)$ *in* $O(\lg_\mu \sigma)$ *time, for any* $\mu \in \Theta(\lg^\epsilon n)$ *and any constant* $0 < \epsilon < 1/2$.

Proof. We first run query $\texttt{rel\_min\_obj\_maj}(\gamma,\beta,\gamma,x)$, and if the result is on column x, we report it. Otherwise we run query $\texttt{rel\_min\_obj\_maj}(\alpha,\beta,\alpha,x+1)$. This means that we can focus on a simpler query of the form $\texttt{rel\_min\_obj\_maj}(\alpha,\beta,x)$, which finds the first pair in $[\alpha,\beta]\times[x,n]$, in object-major order. We map $[x,n]$ to $S[p,t]$ as usual and then proceed recursively on the wavelet tree, remapping p. At each node v, we decompose the query into three subqueries, and then take the minimum result of the three:

1. $\texttt{rel\_min\_obj\_maj}(\alpha, \texttt{g}^{-1}(\texttt{g}(\alpha)+1)-1, x)$ on node $\texttt{child}(\alpha)$;
2. $\texttt{rel\_min\_obj\_maj}(\texttt{g}^{-1}(\texttt{g}(\alpha)+1), \texttt{g}^{-1}(\texttt{g}(\beta))-1, x)$ on the same node v;
3. $\texttt{rel\_min\_obj\_maj}(\texttt{g}^{-1}(\texttt{g}(\beta)), \beta, x)$ on node $\texttt{child}(\beta)$.

Note that queries of type 1 will generate, recursively, only $O(\lg_\mu \sigma)$ further queries of type 1 and 2, and similarly queries of type 3 will generate $O(\lg_\mu \sigma)$ further queries of type 3 and 2. The only queries that actually deliver values are those of type 2, and we will have to take the minimum over $O(\lg_\mu \sigma)$ such results.

A query of type 2 is solved in constant time using bitmap $B_{\texttt{g}(\alpha)+1,\texttt{g}(\beta)-1}$, by computing $q = \texttt{select}_1(B_{\texttt{g}(\alpha)+1,\texttt{g}(\beta)-1}, \texttt{rank}_1(B_{\texttt{g}(\alpha)+1,\texttt{g}(\beta)-1}, p-1)+1)$. This returns a position $S_v[q]$. As we return from the recursion, we remap q in its parent in the usual way, and then (possibly) compare q with the result of a query of type 1 or 3 carried out on the parent. We keep the minimum q value along the way, and when we arrive at the root we return $(S[q], \texttt{unmap}(q))$. □

For the next lemma we need a further data structure. For each sequence $S_v[1,n]$, we store an RMQ structure (Section 2.4), using $O(n) = o(n\lg\mu)$ bits

and finding in constant time the position of a minimum symbol in any range $S_v[i,j]$. This result improves upon the result for query *range_next_value* [21] within the same space.

Lemma 11. BinRel-GWT *supports* $\texttt{rel\_min\_lab\_maj}(\alpha,x,y,z)$ *in* $O(\lg_\mu \sigma + \lg \mu)$ *time, for any* $\mu \in \Theta(\lg^\epsilon n)$ *and any constant* $0 < \epsilon < 1$.

Proof. Again, we can focus on a simpler query $\texttt{rel\_min\_lab\_maj}(\alpha,x,y)$. We map $[x,y]$ to $S[p,q]$ as usual, and the goal is to find the leftmost minimum symbol in $S[p,q]$ that is larger than α.

Assume we are in a wavelet tree node v and the current interval of interest is $S_v[p,q]$. Then, if $S_v[p,q]$ contains symbol $\texttt{g}(\alpha)$ (which is known in constant time with $\texttt{rank}_{\texttt{g}(\alpha)}(S_v,p,q) > 0$), we have to consider it first, by querying recursively the child labeled $\texttt{g}(\alpha)$. If this recursive call returns an answer p', we return it in turn, remapping it to the parent node. If it does not, then any symbol larger than α in the range must correspond to a symbol strictly larger than $\texttt{g}(\alpha)$ in $S_v[p,q]$. We check in constant time whether there is any value larger than $\texttt{g}(\alpha)$ in $S_v[p,q]$, using $\texttt{rank}_{\leq \texttt{g}(\alpha)}(S_v,p,q) < q-p+1$. If there is none, we return in turn with no answer.

If there is an answer, we binary search for the smallest $k \in [\texttt{g}(\alpha)+1, \mu]$ such that $\texttt{rank}_{\leq k}(S_v,p,q) > \texttt{rank}_{\leq \texttt{g}(\alpha)}(S_v,p,q)$. This binary search takes $O(\lg \mu)$ time and is done only once along the whole process. Once we identify the right k, we descend to the appropriate child and start the final stage of the process.

The final stage starts at a node where all the local symbols represent original symbols that are larger than α, and therefore we simply look for the position $m = \textsc{rmq}(S_v,p,q)$, which gives us, in constant time, the first occurrence of the minimum symbol in S_v, and descend to child $S[m]$. This is done until reaching a leaf, from where we return to the root, at position p', and return $(S[p'], \texttt{unmap}(p'))$. It is easy to see that we work $O(1)$ time on $O(\lg_\mu \sigma)$ nodes and $O(\lg \mu)$ once. □

Lemma 12. BinRel-GWT *supports* $\texttt{rel\_sel\_obj\_maj}(\alpha,\beta,x,j)$ *in* $O(\min(\lg n, \lg j \lg(\beta-\alpha+1)) \lg_\mu \sigma)$ *time, for any* $\mu \in \Theta(\lg^\epsilon n)$ *and any constant* $0 < \epsilon < 1/2$.

Proof. The complexities are obtained the same way as for BinRel-WT. The binary search over `rel_num` is sped up because BinRel-GWT supports this operation faster. The other complexity is in principle higher, because the

interval $[\alpha, \beta]$ is split into as many as $O(\mu \lg(\beta - \alpha + 1))$ nodes. However, this can be brought down again to $O(\lg(\beta - \alpha + 1))$ by using the parent node v of each group of (up to μ) contiguous leaves $[k, l]$, and using $\texttt{select}_1$ on the bitmaps $B_{k,l}$ of those parent nodes in order to simulate a contiguous range with all the values in $[k, l]$. So we still have $O(\lg(\beta - \alpha + 1))$ binary searches of $O(\lg j)$ steps, and now each step costs $O(\lg_\mu \sigma)$. □

Lemma 13. BinRel-GWT *supports* $\texttt{lab\_num}(\alpha, \beta, x, y)$ *in* $O(\beta - \alpha + \lg_\mu \sigma)$ *time, for any* $\mu \in \Theta(\lg^\epsilon n)$ *and any constant* $0 < \epsilon < 1/2$.

Proof. We follow the same procedure as for BinRel-WT. The main difference is how to compute the nodes covering the range $[\alpha, \beta]$. This can be done in a naïve way by just verifying whether each symbol appears in the range of S_v, but this raises the complexity by a factor of μ. Thus we need a method to list the symbols appearing in a range of S_v without probing non-existent ones. We resort to a technique loosely inspired by Muthukrishnan [30]. To list the symbol from a range $[k, l]$ that exist in $S_v[p, q]$, we start with the first symbol of the range that appears in $S_v[p, q]$. This is obtained with $p' = \texttt{select}_1(B_{k,l}, \texttt{rank}_1(B_{k,l}, p - 1) + 1)$. If $p' > q$ then there are no such symbols. Otherwise, let $k' = S_v[p']$. Then we know that k' appears in $S_v[p, q]$. Now we continue recursively with subranges $[k, k' - 1]$ and $[k' + 1, l]$. The recursion stops when no p' is found, and it yields all the symbols appearing in $S_v[p, q]$ in $O(1)$ time per symbol. □

The remaining operations are obtained by brute force, just as with BinRel-WT. Figure 5 illustrates the reductions used.

Theorem 3. *The structure* BinRel-GWT*, for a binary relation* $\mathcal{R}$ *of* t *pairs over* $[1, \sigma] \times [1, n]$*, requires* $t \lg \sigma(1 + o(1)) + O(n + t)$ *bits of space and supports the operations within the time complexities given in Table 1 (column 4).*

Proof. The space comes from the $t \lg \sigma + O(n + t)$ bits of the wavelet tree, even in multiary form (Section 2.2), plus the indexes for the virtual bitmaps $B_{\leq k}$. To this we must add the $o(t \lg \sigma)$ bits added by the structures $B_{k,l}$ and the RMQs on the virtual sequences S_v, which add up to $O(t \lg_\mu \sigma) = O(t \lg \sigma / \lg \lg n)$. The time complexities of the operations have been obtained throughout the section. □

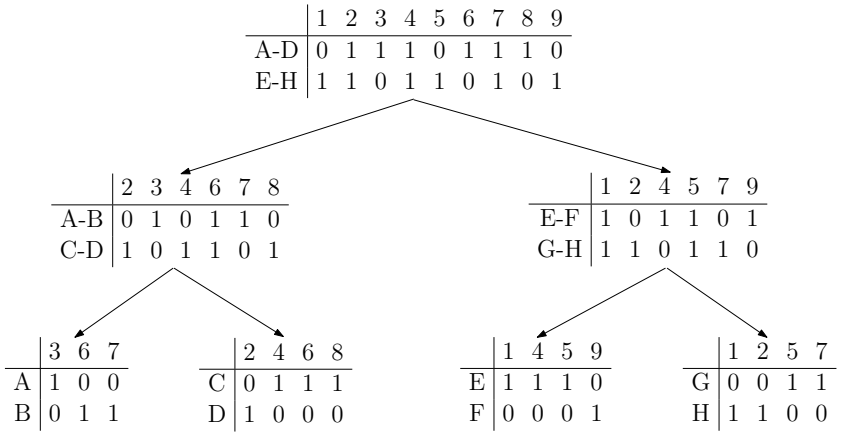

Figure 6: Example of the BRWT for the binary relation in Figure 1.

7. Binary Relation Wavelet Trees (BRWT)

We propose now a special wavelet tree structure tailored to the representation of 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 (α, x) with $\alpha \in [1, \lfloor \sigma/2 \rfloor]$, and B_v^r has the x-th bit set to 1 iff there exists a pair (α, x) with $\alpha \in [\lfloor \sigma/2 \rfloor + 1, \sigma]$. 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 $B[1, \sigma + t]$ recording in unary the number of elements in each row. See Figure 6 for an example. For ease of notation, we define the following functions on B, trivially supported in constant-time: $\texttt{lab}(r) = 1 + \texttt{rank}_0(B, \texttt{select}_1(B, r))$ gives the label of the r-th pair in a label-major traversal of R; while its inverse $\texttt{poslab}(\alpha) = \texttt{rank}_1(B, \texttt{select}_0(B, \alpha))$ gives the position in the traversal where the pairs for label α start.

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. As we will see, the BRWT has weaker functionality than our former structures based on wavelet trees, but it reaches space proportional to $H(\mathcal{R})$.

Lemmas 14 to 18 give a set of operations that can be supported with the BRWT structure.

Lemma 14. BRWT *supports* `rel_num`(α, β, x, y) *in* $O(\beta - \alpha + \lg \sigma)$ *time.*

Proof. We project the interval $[x, y]$ from the root to each leaf in $[\alpha, \beta]$, adding up the resulting interval sizes at the leaves. Of course we can stop earlier if the interval becomes empty. Note that we can only count pairs at the leaves, not at internal nodes. □

Lemma 15. BRWT *supports* `rel_min_lab_maj`(α, x, y, z) *in* $O(\lg \sigma)$ *time.*

Proof. As before, we only need to consider the simpler query `rel_min_lab_maj`(α, x, y). We reach the $O(\lg \sigma)$ wavelet tree nodes $v_1, v_2, \ldots$ that cover the interval $[\alpha, \sigma]$, and map $[x, y]$ to all those nodes, in $O(\lg \sigma)$ time [21]. We choose the first such node, v_k, left to right, with a nonempty interval $[x, y]$. Now we find the leftmost leaf of v_k that has a nonempty interval $[x, y]$, which is easily done in $O(\lg \sigma)$ time. Once we arrive at such a leaf γ with interval $[x, y]$, we map x back to the root, obtaining x', and the answer is (γ, x'). □

Lemma 16. BRWT *supports* `rel_min_obj_maj`$(\alpha, \beta, \gamma, x)$ *in* $O(\lg \sigma)$ *time.*

Proof. As before, we only need to consider the simpler query `rel_min_obj_maj`(α, β, x). Similar to the proof of Lemma 4, we cover $[\alpha, \beta]$ with $O(\lg \sigma)$ wavelet tree nodes $v_1, v_2, \ldots$, and map x to x_i at each such v_i, all in $O(\lg \sigma)$ time. Now, on the way back out of this recursion, we obtain the smallest $y \geq x$ in the root associated to some label in $[\alpha, \beta]$. In this process we keep track of the node v_i that is the source of y, preferring the left child in case of ties. Finally, if we arrive at the root with a value y that came from node v_i, we start from position $x' = x_i$ at node v_i and find the leftmost leaf of v_i related to y. This is done by going left whenever possible (i.e., if $B_v^l[x'] = 1$) and right otherwise, and remapping x' appropriately at each step. Upon reaching a leaf γ, we report (γ, y). □

Lemma 17. BRWT *supports* `obj_sel1`(α, x, j) *in* $O(\lg \sigma)$ *time.*

Proof. We map $x - 1$ from the root to x' in leaf α, then walk up the path from $x' + j$ to the root and report the position obtained. □

Lemma 18. BRWT *supports* `lab_num`(α, β, x, y) *in* $O(\beta - \alpha + \lg \sigma)$ *time.*

Proof. We map $[x, y]$ from the root to each leaf in $[\alpha, \beta]$, adding one per leaf where the interval is non-empty. Recursion can also stop when $[x, y]$ becomes empty. □

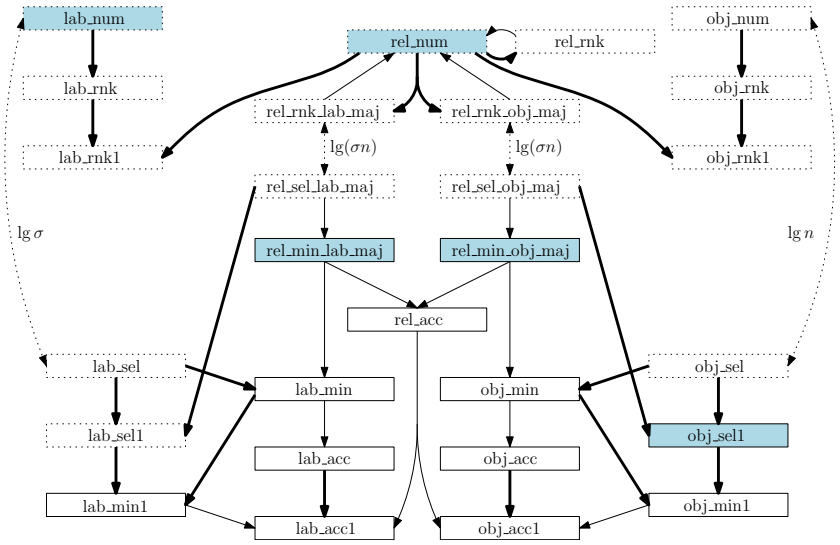

Figure 7: Reductions among operations supported by BRWT. The dotted boxes are operations supported in $\omega(\lg \sigma)$ time, the filled boxes represent operations directly addressed in the paper, and the blank boxes are supported via reductions.

The remaining complexities are obtained by brute force: `rel_sel_lab_maj` and `rel_sel_obj_maj` are obtained by iterating with `rel_min_lab_maj` and `rel_min_obj_maj`, respectively; and similarly `lab_sel`, `obj_sel` and `lab_sel1` using `lab_min`, `obj_min`, and `lab_min1`. Finally, as before `obj_num` and `obj_rnk` are obtained by iterating over `obj_rnk1`. We have obtained the following theorem, illustrated in Figure 7.

Theorem 4. *The* BRWT *structure, for a binary relation $\mathcal{R}$ of t pairs over $[1, \sigma] \times [1, n]$, uses $\lg(1+\sqrt{2})H(\mathcal{R}) + O(t+n+\sigma)$ bits of space and supports the operations within the complexities given in Table 1 (column 5).*

Proof. The operations have been obtained throughout the section. For the space, B is of length $\sigma + t$, and thus $O(t+n+\sigma)$ bits account for B and for the $2n$ bits at the root of the wavelet tree. The rest of the bits in the wavelet tree can be counted by considering that each bit not in the root 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 each such bit at a leaf induces a bit per level on a path from the leaf towards the single x at the root.[6] At worst, all the $O(t_x)$ bits up to level $\lg t_x$ are created for these elements, and from there on all the t_x paths are different, adding up a total of $O(t_x) + t_x \lg \frac{\sigma}{t_x}$ bits. Adding over all x we get $O(t) + \sum_x t_x \lg \frac{\sigma}{t_x}$. This is maximized when $t_x = t/n$ for all x, yielding $O(t) + t \lg \frac{\sigma n}{t} = H(\mathcal{R}) + O(t)$ 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 is 00 possible. Except for those $2n$ bits, we can represent the sequence over an alphabet of size 3 with a zero-order representation [24], to achieve at worst $(\lg 3)H(\mathcal{R}) + o(t)$ 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 $\lg 3$ to $\lg(1+\sqrt{2})$, we consider that the zero-order representation actually achieves $|B_v|H_0(B_v)$ bits. We call B_x the concatenation of all the symbols induced by x, $\ell_x = |B_x| \leq t_x$, and $H_x = |B_x|H_0(B_x)$. 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 $H_x = t_{01} \lg \frac{\ell_x}{t_{01}} + t_{10} \lg \frac{\ell_x}{t_{10}} + t_{11} \lg \frac{\ell_x}{t_{11}}$. 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 $H_x = \lg(1+\sqrt{2})t_x$ bits. This can be achieved separately for each symbol. Using the same distribution of 01's, 10's, and 11's for all x we add up to $\lg(1+\sqrt{2})t \lg \frac{\sigma n}{t} + O(t) = \lg(1+\sqrt{2})H(\mathcal{R}) + O(t)$ bits. (Note that, if we concatenate all the wavelet tree levels, the H_x strings are interleaved in this concatenation.) □

Note that this is a factor of $\lg(1+\sqrt{2}) \approx 1.272$ away of the entropy of $\mathcal{R}$.

[6] For example, in Figure 6, object $x = 4$ is related to labels C and D (see also Figure 1). Its 1 at the second leaf, for C, induces a 1 at its parent, for C-D, and a 1 at the root, for A-D. Its 1 at the third leaf, for E, induces a 1 at its parent for E-F and the 1 at the root for E-H. The fact that $(4, \mathrm{C}) \in \mathcal{R}$ induces the creation of one column at the leaf for C and one at its parent. On the other hand, there are two pairs related to object 1, but they are merged at the second level and thus there is only one path arriving at the root.

8. Conclusions and Future Work

Motivated by the many applications where a binary relation $\mathcal{R}$ between σ labels and n objects arises, we have proposed a rich set of primitives of interest in such applications. We first extended existing representations and showed that their potential is very limited outside single-row or single-column operations. Then we proposed a representation called BinRel-WT, that uses a wavelet tree to solve a large number of operations in time $O(\lg \sigma)$. This structure has been already used in particular cases, but here we have made systematic use of it and exposed its full potential. Furthermore, we have extended the results to generalized wavelet trees, to obtain structure BinRel-GWT, which achieves $O(\lg \sigma / \lg\lg n)$ time for many operations.

Our model generalizes the well-known computational geometry scenario where n points are laid on an $n \times n$ grid. In this case the entropy of the grid is $H(\mathcal{R}) = n \lg n + O(n)$ bits, and our BinRel-GWT obtains asymptotically optimal space, $n \lg n + o(n \lg n)$ bits, with time $O(\lg n / \lg\lg n)$ for many operations, including the basic range counting (i.e., our `rel_rnk` operation) and range reporting (i.e., our `rel_acc` operation). This time is already optimal, within $O(n \operatorname{polylog}(n))$ space, for range counting [35]. This achievement, however, is not a novelty. Bose et al. [9] also obtained those time and spaces before, for square grids. Within *linear* space (i.e., $O(n)$ words, or $O(n \lg n)$ bits), range reporting can be done in time $O(\lg^\epsilon n)$ per point, for any constant $\epsilon > 0$ [12]. Since this ϵ divides the space (i.e., $O(n/\epsilon)$ words), it seems unlikely that such a solution can be adapted to use optimal space.

However, for more complex operations, structure BinRel-GWT offers the best results to date within optimal space. One such operation of interest is the so-called *range successor* query, which corresponds to our `rel_min_lab_maj` and `rel_min_obj_maj` queries, as well as to the *range_next_value* and *prevLess* queries we have mentioned. The current solutions using optimal space [21, 28] achieve $O(\lg n)$ time, whereas our BinRel-GWT reduces it to $O(\lg n / \lg\lg n)$. This also speeds up rectangle visibility queries and the problem of finding the dominant points in a range [32].

Faster competing solutions for range successor queries offer at best linear space. Within this space, they also achieve $O(\lg n / \lg\lg n)$ time. Only very recently, those operations were improved to $O(\lg^\epsilon n)$ time within linear space (more precisely, $O(n/\epsilon)$ words) [34]. We believe it is unlikely that sorted range reporting can be done better than $O(\lg n / \lg\lg n)$ within optimal space.

The other interesting operation is the *range_quantile* query, which cor-

responds to our `rel_sel_lab_maj`. Our BinRel-GWT structure achieves $O(\lg n)$ time for it. This had been the best known complexity, within essentially subquadratic space, for some time. Recently, a linear-space data structure achieved $O(\lg n/\lg\lg n)$ time [11], and it has been shown this time is optimal within $O(n \operatorname{polylog}(n))$ space [26]. Obtaining this optimal time within optimal space is an interesting future challenge.

Although our structures solve many of the queries we have proposed in polylogarithmic (and usually logarithmic) time, supporting others remains a challenge. In particular, we have no good solutions for label-level or object-level `rank` and `select` queries on ranges.

Another challenge is space. All of the described structures use essentially $t \lg \sigma$ bits of space, where t is the number of pairs in the relation. While this is reasonable in many cases, it can be far from the entropy of the binary relation, $H(\mathcal{R})$, for dense relations. We have proposed a variant, BRWT, that uses space within a multiplicative factor 1.272 of $H(\mathcal{R})$, yet its functionality is more limited: Apart from label-level and object-level queries, this structure does not support efficient counting and selection of arbitrary points in ranges, though it can efficiently enumerate them.

Since the publication of the conference version of this article [2], a followup work [18] used our representation BinStr-WT as an internal structure to achieve asymptotically optimal space, $H(\mathcal{R}) + o(H(\mathcal{R}))$ bits, and was able to solve query `rel_rnk` in time $O(\lg n)$, and queries `rel_acc`, `rel_sel_lab_maj` and `rel_sel_obj_maj` in time $O(\lg^2 n)$, on $n \times n$ grids. Using structure BinRel-GWT their times for `rel_rnk` and `rel_acc` can be divided by $\lg\lg n$. It is an open challenge to reach the best possible times, for example those of BinRel-GWT, while approaching space $H(\mathcal{R})$.

On the other hand, $H(\mathcal{R})$ is a crude measure that does not account for regularities in the row/column distribution, or clustering, that arises in real-life binary relations. For example, structures BinRel-WT and BinRel-GWT can be compressed to the zero-order entropy of a string of length t, and this can in some cases be below $H(\mathcal{R})$, which shows that this measure is not sufficiently refined. It would be interesting to consider finer measures of entropy and try to match them.

There might also be other operations of interest apart from the set we have identified. For example, determining whether a pair is related in the *transitive closure* of $\mathcal{R}$ is relevant for many applications (e.g., ancestorship in trees, or paths in graphs). Another extension is to d-ary relations, which would more naturally capture joins in the relational model.

Finally, we have considered only static relations. Wavelet trees do allow dynamism, so that new pairs and/or objects can be inserted in/deleted from the relation, within an $O(\lg n/\lg\lg n)$ time penalty factor [33]. This is optimal, as it lets BinRel-GWT match the $\Omega((\lg n/\lg\lg n)^2)$ time lower bound on dynamic range counting [35]. Adding/removing labels is also possible, yet the best current times are amortized [31].

Appendix A. Formal Definition of Operations

We formally define our set of operations. Figure A.8 graphically illustrates some of them.

- $\texttt{rel\_acc}(\alpha,\beta,x,y) \;=\; \{(\gamma,z)\in\mathcal{R},\ \gamma\in[\alpha,\beta]\ \wedge\ z\in[x,y]\}$
- $\texttt{rel\_sel\_lab\_maj}(\alpha,j,x,y) \;=\; j$-th smallest pair of $\texttt{rel\_acc}(\alpha,\sigma,x,y)$ in order $(\alpha,x)\le(\beta,y)\Leftrightarrow\alpha<\beta\vee(\alpha=\beta\wedge x\le y)$
- $\texttt{rel\_min\_lab\_maj}(\alpha,x,y,z) \;=\;$ under the same order, smallest pair of $\texttt{rel\_acc}(\alpha,\alpha,z,y)\cup\texttt{rel\_acc}(\alpha{+}1,\sigma,x,y)$
- $\texttt{rel\_sel\_obj\_maj}(\alpha,\beta,x,j) \;=\; j$-th smallest pair of $\texttt{rel\_acc}(\alpha,\beta,x,n)$ in order $(\alpha,x)\le(\beta,y)\Leftrightarrow x<y\vee(x=y\wedge\alpha\le\beta)$
- $\texttt{rel\_min\_obj\_maj}(\alpha,\beta,\gamma,x) \;=\;$ under the same order, smallest pair of $\texttt{rel\_acc}(\gamma,\beta,x,x)\cup\texttt{rel\_acc}(\alpha,\beta,x{+}1,n)$
- $\texttt{lab\_acc}(\alpha,\beta,x,y) \;=\; \{\gamma,\ \exists z,\ (\gamma,z)\in\texttt{rel\_acc}(\alpha,\beta,x,y)\}$
- $\texttt{lab\_acc1}(\alpha,\beta,x) \;=\; \texttt{lab\_acc}(\alpha,\beta,x,x)$
- $\texttt{lab\_sel}(\alpha,j,x,y) \;=\; j$-th smallest label of $\texttt{lab\_acc}(\alpha,\sigma,x,y)$
- $\texttt{lab\_sel1}(\alpha,j,x) \;=\; \texttt{lab\_sel}(\alpha,j,x,x)$, or $\texttt{rel\_sel\_lab\_maj}(\alpha,j,x,x)$
- $\texttt{lab\_min}(\alpha,x,y) \;=\; \texttt{lab\_sel}(\alpha,1,x,y)$
- $\texttt{lab\_min1}(\alpha,x) \;=\; \texttt{lab\_min}(\alpha,x,x)$, or $\texttt{lab\_sel1}(\alpha,1,x)$
- $\texttt{obj\_acc}(\alpha,\beta,x,y) \;=\; \{z,\ \exists\gamma,\ (\gamma,z)\in\texttt{rel\_acc}(\alpha,\beta,x,y)\}$
- $\texttt{obj\_acc1}(\alpha,x,y) \;=\; \texttt{obj\_acc}(\alpha,\alpha,x,y)$

- $\texttt{obj\_sel}(\alpha, \beta, x, j) = j$-th smallest object of $\texttt{obj\_acc}(\alpha, \beta, x, n)$
- $\texttt{obj\_sel1}(\alpha, x, j) = \texttt{obj\_sel}(\alpha, \alpha, x, j)$, or $\texttt{rel\_sel\_obj\_maj}(\alpha, \alpha, x, j)$
- $\texttt{obj\_min}(\alpha, \beta, x) = \texttt{obj\_sel}(\alpha, \beta, x, 1)$
- $\texttt{obj\_min1}(\alpha, x) = \texttt{obj\_min}(\alpha, \alpha, x)$, or $\texttt{obj\_sel1}(\alpha, x, 1)$
- $\texttt{rel\_num}(\alpha, \beta, x, y) = |\texttt{rel\_acc}(\alpha, \beta, x, y)|$
- $\texttt{rel\_rnk}(\alpha, x) = \texttt{rel\_num}(1, \alpha, 1, x)$
- $\texttt{rel\_rnk\_lab\_maj}(\alpha, x, y, z) = \texttt{rel\_num}(1, \alpha{-}1, x, y){+}\texttt{rel\_num}(\alpha, \alpha, x, z)$
- $\texttt{rel\_rnk\_obj\_maj}(\alpha, \beta, \gamma, x) = \texttt{rel\_num}(\alpha, \beta, 1, x{-}1){+}\texttt{rel\_num}(\alpha, \gamma, x, x)$
- $\texttt{lab\_num}(\alpha, \beta, x, y) = |\texttt{lab\_acc}(\alpha, \beta, x, y)|$
- $\texttt{lab\_rnk}(\alpha, x, y) = \texttt{lab\_num}(1, \alpha, x, y)$
- $\texttt{lab\_rnk1}(\alpha, x) = \texttt{lab\_rnk}(\alpha, x, x)$, or $\texttt{rel\_num}(1, \alpha, x, x)$
- $\texttt{obj\_num}(\alpha, \beta, x, y) = |\texttt{obj\_acc}(\alpha, \beta, x, y)|$
- $\texttt{obj\_rnk}(\alpha, \beta, x) = \texttt{obj\_num}(\alpha, \beta, 1, x)$
- $\texttt{obj\_rnk1}(\alpha, x) = \texttt{obj\_rnk}(\alpha, \alpha, x)$, or $\texttt{rel\_num}(\alpha, \alpha, 1, x)$

Appendix B. Algorithms on BinRel-Str

We prove some of the complexities in Table 1 (column 2). The others can be derived using Theorem 1.

Lemma 19. BinRel-Str *supports* $\texttt{rel\_num}(\alpha, \beta, x, y)$ *in* $O(\min((\beta - \alpha + 1)r, (y - x + 1)a \lg \beta))$ *time.*

Proof. We can compute $\texttt{rel\_num}(\alpha, \beta, x, y)$ in two ways:

- Using the identities $\texttt{rel\_num}(\alpha, \beta, x, y) = \sum_{\alpha \le \gamma \le \beta} \texttt{rel\_num}(\gamma, \gamma, x, y)$ and $\texttt{rel\_num}(\gamma, \gamma, x, y) = \texttt{rank}_\gamma(S, \texttt{map}(x - 1) + 1, \texttt{map}(y))$, we achieve time $O((\beta - \alpha + 1)r)$.

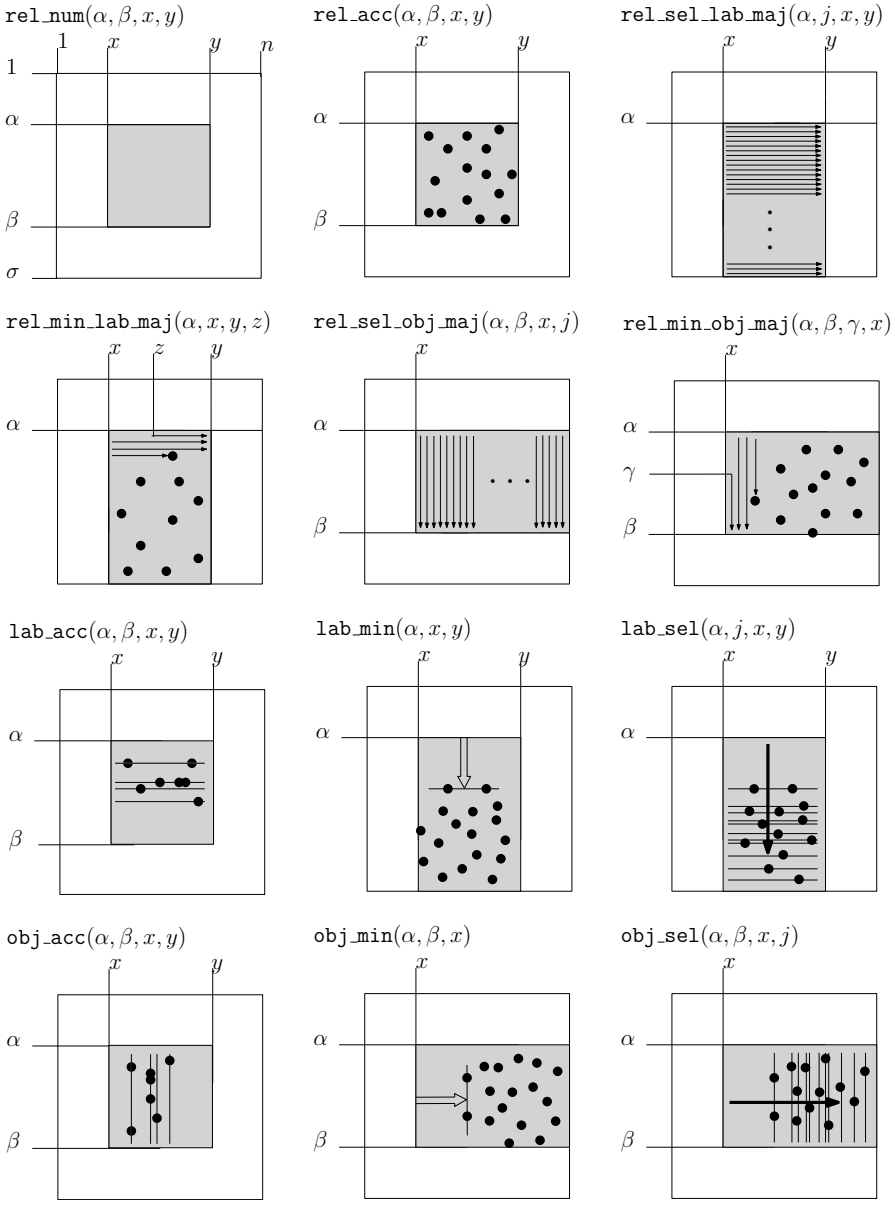

Figure A.8: Some operations illustrated.

- Since $\texttt{rel\_num}(\alpha, \beta, x, y) = \sum_{x \leq z \leq y} \texttt{rel\_num}(\alpha, \beta, z, z)$, we can compute the value for each z by searching for the successor of α and the predecessor of β in $S[\texttt{map}(z-1)+1, \texttt{map}(z)]$. As this range of S is sorted we can find the predecessors and successors using exponential search, which requires in $O(\lg \beta)$ `access` operations. Thus the whole process takes time $O((y-x+1)a \lg \beta)$.

□

Lemma 20. BinRel-Str *supports* $\texttt{lab\_num}(\alpha, \beta, x, y)$ *in* $O(\min((\beta - \alpha + 1)r, (y-x+1)(\lg \alpha + \beta - \alpha)a))$ *time.*

Proof. We can compute $\texttt{lab\_num}(\alpha, \beta, x, y)$ in two ways:

- Using that $\texttt{lab\_num}(\alpha, \beta, x, y) = \sum_{\alpha \leq \gamma \leq \beta} \texttt{lab\_num}(\gamma, \gamma, x, y)$, and that $\texttt{lab\_num}(\gamma, \gamma, x, y) = 1$ iff $\texttt{rel\_num}(\gamma, \gamma, x, y) > 0$ and zero otherwise, we can achieve the same time as in the first alternative of Lemma 19.

- As $\texttt{lab\_num}(\alpha, \beta, x, y) = |\cup_{x \leq z \leq y} \texttt{lab\_acc}(\alpha, \beta, z, z)|$, we can collect the labels in $[\alpha, \beta] \times [z, z]$ for each z and insert them into a dictionary. The labels related to a single z can be found by using a similar method as the one in Lemma 19: use exponential search to find the first element $\geq \alpha$ in z's area of S, and then scan the next symbols until surpassing β. We mark each label found in a bitmap of length $\beta - \alpha + 1$, and then we report the number of ones in it.

□

Lemma 21. BinRel-Str *supports* $\texttt{obj\_num}(\alpha, \beta, x, y)$ *in* $O(\min((y-x+1)a \lg \alpha, (\beta - \alpha + 1)(r + (y-x+1)s)))$ *time.*

Proof. We can compute $\texttt{obj\_num}(\alpha, \beta, x, y)$ in two ways:

- Using $\texttt{obj\_num}(\alpha, \beta, x, y) = \sum_{x \leq z \leq y} \texttt{obj\_num}(\alpha, \beta, z, z)$, and since $\texttt{obj\_num}(\alpha, \beta, z, z) = 1$ iff $\texttt{rel\_num}(\alpha, \beta, z, z) > 0$ and zero otherwise, we can proceed similarly as in the second alternative of Lemma 19, by exponentially searching for the first value $\geq \alpha$ in z's area of S and checking whether it is $\leq \beta$.

- Because $\texttt{obj\_num}(\alpha, \beta, x, y) = |\cup_{\alpha \leq \gamma \leq \beta} \texttt{obj\_acc}(\gamma, \gamma, x, y)|$, we can collect the objects in $[\gamma, \gamma] \times [x, y]$ for each γ and insert them into a dictionary, as in Lemma 20. The objects related to a single γ can be found by

using successive $\texttt{select}_\gamma(S, j)$ operations on $S[\texttt{map}(x-1)+1, \texttt{map}(y)]$, starting with $j = \texttt{rank}_\gamma(S, \texttt{map}(x-1)) + 1$. The complexity considers the worst case where each such γ appears $y - x + 1$ times.

□

Note that, when reducing to implement `obj_rnk`, the $\texttt{rank}_\gamma(S, \cdot)$ operation is not necessary. For `obj_rnk1` it is better to reduce from `rel_num`.

Lemma 22. BinRel-Str *supports* $\texttt{rel\_sel\_lab\_maj}(\alpha, j, x, y)$ *in* $O(\min((\sigma - \alpha)r + s, (y - x + 1)(\sigma - \alpha + 1)a))$ *time.*

Proof. Again, we have two possible solutions:

- Set $c \leftarrow 0$. For each label γ in $[\alpha, \sigma]$, compute $c' \leftarrow c + \texttt{rank}_\gamma(S, \texttt{map}(x-1)+1, \texttt{map}(y))$. If at some step it holds $c' \geq j$, the answer is $(\gamma, \texttt{unmap}(\texttt{select}_\gamma(S, j - c + \texttt{rank}_\gamma(S, \texttt{map}(x-1)))))$. Otherwise, update $c \leftarrow c'$. The whole process takes $O((\sigma - \alpha + 1)r + s)$ time.
- Similarly, but first accumulating all the occurrences of all the labels γ and then finding j using the accumulators. We simply traverse the area $S[\texttt{map}(z-1)+1, \texttt{map}(z)]$ backwards, for each $z \in [x, y]$, accessing each label $S[k]$ and incrementing the corresponding counter. The process takes $O((y - x + 1)(\sigma - \alpha + 1)a)$ time.

□

From this operation we can obtain complexities for `rel_min_lab_maj`, `lab_min`, `lab_min1`, `lab_acc`, and `lab_acc1`. Some of the results we give are better than those obtained by a blind reduction; we leave to the reader to check these improvements. We also note that an obvious variant of this algorithm is our best solution to compute `lab_sel`, which has the same time.

Lemma 23. BinRel-Str *supports* $\texttt{rel\_sel\_obj\_maj}(\alpha, \beta, x, j)$ *in* $O(\min((n - x + 1)a \lg \beta, (\beta - \alpha + 1)((n - x + 1)s + r)))$ *time.*

Proof. Once again, we have two possible solutions:

- Set $c \leftarrow 0$. For each object z in $[x, n]$, use exponential search on z's area of S to find the range $S[a, b]$ corresponding to $[\alpha, \beta]$, and set $c' \leftarrow c + b - a + 1$. If at some step it holds $c' \geq j$, the answer is $(S[j - c + a - 1], z)$. Otherwise, update $c \leftarrow c'$. The whole process takes $O((n - x + 1)a \lg \beta)$ time.

- Similarly, but first accumulating all the occurrences of all the objects z and then finding j using the accumulators. We traverse the area $S[\texttt{map}(x-1)+1, \texttt{map}(y)]$ for each label γ in $[\alpha, \beta]$, using successive $\texttt{select}_\gamma(S, j')$ queries, starting at $j' = \texttt{rank}_\gamma(S, \texttt{map}(x-1))+1$. The process takes $O((\beta-\alpha+1)((n-x+1)s+r))$ time.

□

From this operation we can obtain complexities for `rel_min_obj_maj`, `rel_acc`, `obj_min`, `obj_min1`, `obj_acc`, and `obj_acc1`. Once again, some of the results we give are better than a blind reduction and we leave the reader to verify those. Finally, an obvious variant of this algorithm is our best solution to compute `obj_sel`.

A final easy exercise for the reader is to show that $\texttt{lab\_sel1}(\alpha, j, x)$ can be solved in time $O(a \lg \alpha)$, and that $\texttt{obj\_sel1}(\alpha, x, j)$ is solved in time $O(r+s)$.

References

[1] Barbay, J., Aleardi, L., He, M., Munro, J., 2012. Succinct representation of labeled graphs. Algorithmica 62 (1-2), 224–257.

[2] Barbay, J., Claude, F., Navarro, G., 2010. Compact rich-functional binary relation representations. In: Proc. 9th Latin American Symposium on Theoretical Informatics (LATIN). LNCS 6034. pp. 170–183.

[3] Barbay, J., Gagie, T., Navarro, G., Nekrich, Y., 2010. Alphabet partitioning for compressed rank/select and applications. In: Proc. 21st Annual International Symposium on Algorithms and Computation (ISAAC). LNCS 6507. Springer, pp. 315–326 (part II), journal version to appear in *Algorithmica*, DOI 10.1007/s00453-012-9726-3.

[4] Barbay, J., Golynski, A., Munro, I., Rao, S. S., 2007. Adaptive searching in succinctly encoded binary relations and tree-structured documents. Theoretical Computer Science 387 (3), 284–297.

[5] Barbay, J., He, M., Munro, I., Rao, S. S., 2011. Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms 7 (4), article 52.

[6] Barbay, J., López-Ortiz, A., Lu, T., Salinger, A., 2009. An experimental investigation of set intersection algorithms for text searching. ACM Journal of Experimental Algorithmics 14, article 7.

[7] Barbay, J., Navarro, G., 2009. Compressed representations of permutations, and applications. In: Proc. 26th International Symposium on Theoretical Aspects of Computer Science (STACS). pp. 111–122.

[8] Boldi, P., Vigna, S., 2004. The WebGraph framework I: compression techniques. In: Proc. 13th World Wide Web Conference (WWW). pp. 595–602.

[9] Bose, P., He, M., Maheshwari, A., Morin, P., 2009. Succinct orthogonal range search structures on a grid with applications to text indexing. In: Proc. 20th Symposium on Algorithms and Data Structures (WADS). LNCS 5664. pp. 98–109.

[10] Brisaboa, N., Ladra, S., Navarro, G., 2009. k^2-trees for compact web graph representation. In: Proc. 16th International Symposium on String Processing and Information Retrieval (SPIRE). LNCS 5721. pp. 18–30.

[11] Brodal, G. S., Gfeller, B., Jørgensen, A. G., Sanders, P., 2011. Towards optimal range medians. Theoretical Computer Science 412 (24), 2588–2601.

[12] Chan, T., Larsen, K. G., Pătraşcu, M., 2011. Orthogonal range searching on the RAM, revisited. In: Proc. 27th ACM Symposium on Computational Geometry (SoCG). pp. 1–10, extended version in `http://arxiv.org/abs/1103.5510`.

[13] Chien, Y.-F., Hon, W.-K., Shah, R., Vitter, J., 2008. Geometric Burrows-Wheeler transform: Linking range searching and text indexing. In: Proc. 18th Data Compression Conference (DCC). pp. 252–261, to appear in *Algorithmica*, DOI 10.1007/s00453-013-9792-1.

[14] Clark, D., 1996. Compact PAT trees. Ph.D. thesis, Univ. of Waterloo, Canada.

[15] Claude, F., Navarro, G., 2010. Extended compact Web graph representations. In: Elomaa, T., Mannila, H., Orponen, P. (Eds.), Algorithms

and Applications (Ukkonen Festschrift). LNCS 6060. Springer, pp. 77–91.

[16] Claude, F., Navarro, G., 2010. Fast and compact Web graph representations. ACM Transactions on the Web 4 (4), article 16.

[17] Claude, F., Navarro, G., 2010. Self-indexed grammar-based compression. Fundamenta Informaticae 111 (3), 313–337.

[18] Farzan, A., Gagie, T., Navarro, G., 2010. Entropy-bounded representation of point grids. In: Proc. 21st Annual International Symposium on Algorithms and Computation (ISAAC). LNCS 6507. pp. 327–338, part II.

[19] Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G., 2007. Compressed representations of sequences and full-text indexes. ACM Transactions on Algorithms 3 (2), article 20.

[20] Fischer, J., Heun, V., 2011. Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM Journal of Computing 40 (2), 465–492.

[21] Gagie, T., Navarro, G., Puglisi, S., 2012. New algorithms on wavelet trees and applications to information retrieval. Theoretical Computer Science 426-427, 25–41.

[22] Golynski, A., 2007. Optimal lower bounds for rank and select indexes. Theoretical Computer Science 387 (3), 348–359.

[23] Golynski, A., Munro, I., Rao, S., 2006. Rank/select operations on large alphabets: a tool for text indexing. In: Proc. 17th Annual Symposium on Discrete Algorithms (SODA). pp. 368–373.

[24] Golynski, A., Raman, R., Rao, S., 2008. On the redundancy of succinct data structures. In: Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT). LNCS 5124. pp. 148–159.

[25] Grossi, R., Gupta, A., Vitter, J., 2003. High-order entropy-compressed text indexes. In: Proc. 14th Annual Symposium on Discrete Algorithms (SODA). pp. 841–850.

[26] Jørgensen, A. G., Larsen, K. D., 2011. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In: Proc. 22nd Symposium on Discrete Algorithms (SODA). pp. 805–813.

[27] Kärkkäinen, J., 1999. Repetition-based text indexing. Ph.D. thesis, U. Helsinki, Finland.

[28] Kreft, S., Navarro, G., 2013. On compressing and indexing repetitive sequences. Theoretical Computer Science 483, 115–133.

[29] Mäkinen, V., Navarro, G., 2007. Rank and select revisited and extended. Theoretical Computer Science 387 (3), 332–347.

[30] Muthukrishnan, S., 2002. Efficient algorithms for document retrieval problems. In: Proc 13th Annual Symposium on Discrete Algorithms (SODA). pp. 657–666.

[31] Navarro, G., Nekrich, Y., 2013. Optimal dynamic sequence representations. In: Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 865–876.

[32] Navarro, G., Nekrich, Y., Russo, L., 2013. Space-efficient data-analysis queries on grids. Theoretical Computer Science 482, 60–72.

[33] Navarro, G., Sadakane, K., 2012. Fully-functional static and dynamic succinct trees. ACM Transactions on Algorithms. To appear, see `http://arxiv.org/abs/0905.0768v5`.

[34] Nekrich, Y., Navarro, G., 2012. Sorted range reporting. In: Proc. 13th Scandinavian Symposium on Algorithmic Theory (SWAT). LNCS 7357. pp. 271–282.

[35] Pătraşcu, M., 2007. Lower bounds for 2-dimensional range counting. In: Proc. 39th Annual ACM Symposium on Theory of Computing (STOC). pp. 40–46.

[36] Pătraşcu, M., 2008. Succincter. In: Proc. 49th Annual Symposium on Foundations of Computer Science (FOCS). pp. 305–313.

[37] Raman, R., Raman, V., Rao, S. S., 2007. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3 (4), article 43.

[38] Witten, I., Moffat, A., Bell, T., 1999. Managing Gigabytes, 2nd Edition. Morgan Kaufmann Publishers.