# New Lower and Upper Bounds for Representing Sequences ⋆

Djamal Belazzougui[1] and Gonzalo Navarro[2]

[1] LIAFA, Univ. Paris Diderot - Paris 7, France. dbelaz@liafa.jussieu.fr
[2] Department of Computer Science, University of Chile. gnavarro@dcc.uchile.cl

**Abstract.** Sequence representations supporting queries *access*, *select* and *rank* are at the core of many data structures. There is a considerable gap between different upper bounds, and the few lower bounds, known for such representations, and how they interact with the space used. In this article we prove a strong lower bound for *rank*, which holds for rather permissive assumptions on the space used, and give matching upper bounds that require only a compressed representation of the sequence. Within this compressed space, operations *access* and *select* can be solved within almost-constant time.

## 1 Introduction

A large number of data structures build on sequence representations. In particular, supporting the following three queries on a sequence $S[1,n]$ over alphabet $[1,\sigma]$ has proved extremely useful:

- $access(S,i)$ gives $S[i]$;
- $select_a(S,j)$ gives the position of the $j$th occurrence of $a \in [1,\sigma]$ in $S$; and
- $rank_a(S,i)$ gives the number of occurrences of $a \in [1,\sigma]$ in $S[1,i]$.

For example, Ferragina and Manzini's FM-index [9], a compressed indexed representation for text collections that supports pattern searches, is most successfully implemented over a sequence representation supporting *access* and *rank* [10]. Grossi et al. [18] had used earlier similar techniques for text indexing, and invented *wavelet trees*, a compressed sequence representation that solves the three queries in time $O(\lg\sigma)$. The time was reduced to $O(\frac{\lg\sigma}{\lg\lg n})$ with multiary wavelet trees [10, 17].[3] Golynski et al. [16] used these operations for representing labeled trees and permutations, and proposed another representation that solved the operations in time $O(\lg\lg\sigma)$, and some even in constant time. This representation was made compressed by Barbay et al. [1]. Further applications of the three operations to multi-labeled trees and binary relations were uncovered by Barbay et al. [2]. Ferragina et al. [8] and Gupta et al. [20] devised new

⋆ Partially funded by Fondecyt Grant 1-110066, Chile. First author also partially supported by the French ANR-2010-COSI-004 MAPPI Project.

[3] For simplicity, throughout this paper we will assume that $\lg x$ means $\max(1, \lg x)$. Similarly, $O(x)$ will mean $O(\max(1,x))$ and $o(x)$ will mean $o(\max(1,x))$.

applications to XML indexing. Barbay et al. [3, 1] gave applications to representing permutations and inverted indexes. Claude and Navarro [7] presented applications to graph representation. Mäkinen and Välimäki [29] and Gagie et al. [13] applied them to document retrieval on general texts.

The most basic case is that of bitmaps, when $\sigma = 2$. In this case obvious applications are set representations supporting membership and predecessor search. We assume throughout this article the RAM model with word size $w = \Omega(\lg n)$. Jacobson [21] achieved constant-time *rank* using $o(n)$ extra bits on top of a plain representation of $S$, and Munro [23] and Clark [6] achieved also constant-time *select*. Golynski [14] showed a lower bound of $\Omega(n \lg\lg n / \lg n)$ extra bits for supporting both operations in constant time if $S$ is to be represented in plain form (i.e., as an array of $n$ bits), and gave matching upper bounds. When $S$ can be represented arbitrarily, Patrascu [25] achieved $\lg \binom{n}{m} + O(n/\lg^c n)$ bits of space, where $m$ is the number of 1s in $S$ and $c$ is any constant, and showed this is optimal [28].

For general sequences, a useful measure of compressibility is the *zero-order entropy* of $S$, $H_0(S) = \sum_{a\in[1,\sigma]} \frac{n_a}{n} \lg \frac{n}{n_a}$, where $n_a$ is the number of occurrences of $a$ in $S$. This can be extended to the *k-th order entropy*, $H_k(S) = \frac{1}{n}\sum_{A\in[1,\sigma]^k} |T_A| H_0(T_A)$, where $T_A$ is the string of symbols following $k$-tuple $A$ in $S$. It holds $0 \le H_k(S) \le H_{k-1}(S) \le H_0(S) \le \lg\sigma$ for any $k$, but the entropy measure is only meaningful for $k < \lg_\sigma n$. See Manzini [22] and Gagie [12] for a deeper discussion.

When representing sequences supporting these operations, we may aim at using $O(n\lg\sigma)$ bits of space, but frequently one aims for less space. We may aim at *succinct* representation of $S$, taking $n\lg\sigma + o(n\lg\sigma)$ bits, at a *zero-order* compressed representation, taking at most $nH_0(S)+o(n\lg\sigma)$ bits (we might also wish to compress the redundancy, $o(n\lg\sigma)$, to achieve for example $nH_0(S) + o(nH_0(S))$), or at a *high-order* compressed representation, $nH_k(S) + o(n\lg\sigma)$.

Upper and lower bounds for sequence representations supporting the three operations are far less understood over larger alphabets. When $\sigma = O(\text{polylog}\, n)$, the three operations can be carried out in constant time over a data structure using $nH_0(S)+o(n)$ bits [10]. For larger alphabets, this solution requires the same space and answers the queries in time $O(\frac{\lg\sigma}{\lg\lg n})$ [10, 17]. Another class of solutions [16, 19, 1], especially attractive for "large alphabets", achieves time $O(\lg\lg\sigma)$ for *rank*. For *access* and *select* they offer complementary complexities, where one of the operations is constant-time and the other requires $O(\lg\lg\sigma)$ time. They achieve zero-order compressed space, $nH_0(S) + o(nH_0(S)) + o(n)$ bits [1], and even high-order compressed space, $nH_k(S) + o(n\lg\sigma)$ for any $k = o(\lg_\sigma n)$ [19].

There are several curious aspects in the map of the current solutions for general sequences. On one hand, the times for *access* and *select* seem to be complementary, whereas that for *rank* is always the same. On the other hand, there is no smooth transition between the complexity of one solution, $O(\frac{\lg\sigma}{\lg\lg n})$, and that of the other, $O(\lg\lg\sigma)$.

The complementary nature of *access* and *select* is not a surprise. Golynski [15] gave lower bounds that relate the time performance that can be achieved for these operations with the redundancy of an encoding of $S$ on top of its information content. The lower bound acts on the product of both times, that is, if $t$ and $t'$ are the time complexities, and $\rho$ is the bit-redundancy per symbol, then $\rho \cdot t \cdot t' = \Omega((\lg \sigma)^2/w)$ holds for a wide range of values of $\sigma$. The upper bounds for large alphabets [16, 19] match this lower bound.

Although operation *rank* seems to be harder than the others (at least no constant-time solution exists except for polylog-sized alphabets), no general lower bounds on this operation have been proved. Only a recent result for the case in which $S$ must be encoded in plain form states that if one solves *rank* within $a = O(\frac{\lg \sigma}{\lg \lg \sigma})$ access to the sequence, then the redundancy per symbol is $\rho = \Omega((\lg \sigma)/a)$ [19]. Since in the RAM model one can access up to $w/\lg \sigma$ symbols in one access, this implies a lower bound of $\rho \cdot t = \Omega((\lg \sigma)^2/w)$, similar to the one by Golynski [15] for the product of *access* and *select* times.

In this article we make several contributions that help close the gap between lower and upper bounds on sequence representation.

1. We prove the first general lower bound on *rank*, which shows that this operation is, in a sense, noticeably harder than the others: No structure using $O(n \cdot w^{O(1)})$ bits can answer *rank* queries in time $o(\lg \frac{\lg \sigma}{\lg w})$. Note the space includes the rather permissive $O(n \cdot \text{polylog}\, n)$. For this range of times our general bound is much stronger than the existing restricted one [19], which only forbids achieving it within $n \lg \sigma + O(n \lg^2 \sigma/(w \lg \frac{\lg \sigma}{\lg w}))$ bits. Our lower bound uses a reduction from predecessor queries.
2. We give a matching upper bound for *rank*, using $O(n \lg \sigma)$ bits of space and answering queries in time $O(\lg \frac{\lg \sigma}{\lg w})$. This is lower than any time complexity achieved so far for this operation within $O(n \cdot w^{O(1)})$ bits, and it elegantly unifies both known upper bounds under a single and lower time complexity. This is achieved via a reduction to a predecessor query structure that is tuned to use slightly less space than usual.
3. We derive succinct and compressed representations of sequences that achieve time $O(\frac{\lg \sigma}{\lg w})$ for *access*, *select* and *rank*, improving upon previous results [10]. This yields constant-time operations for $\sigma = w^{O(1)}$. Succinctness is achieved by replacing universal tables used in other solutions with bit manipulations on the RAM model. Compression is achieved by combining the succinct representation with existing compression boosters.
4. We derive succinct and compressed representations of sequences over larger alphabets, which achieve time $O(\lg \frac{\lg \sigma}{\lg w})$ for *rank*, which is optimal, and almost-constant time for *access* and *select*. The result improves upon almost all succinct and compressed representations proposed so far [16, 2, 1, 19]. This is achieved by plugging our $O(n \lg \sigma)$-bit solutions into existing succinct and compressed data structures.

Our results assume a RAM model where bit shifts, bitwise logical operations, and arithmetic operations (including multiplication) are permitted. Otherwise we can simulate them with universal tables within $o(n)$ extra bits of space, but all $\lg w$ in our upper bounds become $\lg \lg n$.

## 2 Lower Bound for *rank*

Our technique is to reduce from a predecessor problem and apply the density-aware lower bounds of Patrascu and Thorup [26]. Assume that we have $n$ keys from a universe of size $u = n\sigma$, then the keys are of length $\ell = \lg u = \lg n + \lg \sigma$. According to branch 2 of Patrascu and Thorup's result, the time for predecessor queries in this setting is lower bounded by $\Omega\left(\lg\left(\frac{\ell - \lg n}{a}\right)\right)$, where $a = \lg(s/n) + \lg w$ and $s$ is the space in words of our representation (the lower bound is in the cell probe model for word length $w$, so the space is always expressed in number of cells). The lower bounds holds even for a more restricted version of the predecessor problem in which one of two colors is associated with each element and the query only needs to return the color of the predecessor. We assume $\sigma = O(n)$; the other case will be considered at the end of the section.

The reduction is as follows. We divide the universe $n \cdot \sigma$ into $\sigma$ intervals, each of size $n$. This division can be viewed as a binary matrix of $n$ columns by $\sigma$ rows, where we set a 1 at row $r$ and column $c$ iff element $(r-1) \cdot n + c$ belongs to the set. We will use four data structures.

1. A plain bitvector $L[1, n]$ which stores the color associated with each element. The array is indexed by the original ranks of the elements.
2. A partial sums structure $R$ stores the number of elements in each row. It is a bitmap concatenating the $\sigma$ unary representations, $1^{n_r}0$, of the number of 1s in each row $r \in [1, \sigma]$. Thus $R$ is of length $n + \sigma$ and can give in constant time the number of 1s up to (and including) any row $r$, $\mathit{count}(r) = \mathit{rank}_1(R, \mathit{select}_0(R, r)) = \mathit{select}_0(R, r) - r$, in constant time and $O(n + \sigma) = O(n)$ bits of space [23, 6].
3. A column mapping data structure $C$ that maps the original columns into a set of columns where $(i)$ empty columns are eliminated, and $(ii)$ new columns are created when two or more 1s fall in the same column. $C$ is a bitmap concatenating the $n$ unary representations, $1^{n_c}0$, of the numbers $n_c$ of 1s in each column $c \in [1, n]$. So $C$ is of length $2n$. Note that the new matrix of mapped columns has also $n$ columns (one per element in the set) and exactly one 1 per column. The original column $c$ is then mapped to $\mathit{col}(c) = \mathit{rank}_1(C, \mathit{select}_0(C, c)) = \mathit{select}_0(C, c) - c$, using constant time and $O(n)$ bits. Note that $\mathit{col}(c)$ is the last of the columns to which the original column $c$ might have been expanded.
4. A string $S[1, n]$ over alphabet $[1, \sigma]$, so that $S[c] = r$ iff the only 1 at column $c$ (after column remapping) is at row $r$. Over this string we build a data structure able to answer queries $\mathit{rank}_r(S, c)$.

Queries are done in the following way. Given an element $x \in [1, u]$, we first deompose it into a pair $(r, c)$ where $x = (r-1) \cdot n + c$. In a first step, we compute $\mathit{count}(r-1)$ in constant time. This gives us the count of elements up to point $(r-1) \cdot n$. Next we must compute the count of elements in the range $[(r-1) \cdot n + 1, (r-1) \cdot n + c]$. For doing that we first remap the column to $c' = \mathit{col}(c)$ in constant time, and finally compute $\mathit{rank}_r(S, c')$, which gives the number of 1s in row $r$ up to column $c'$. Note that if column $c$ was expanded to several ones, we are counting the 1s up to the last of the expanded columns, so that all the original 1s at column $c$ are counted at their respective rows. Then the rank of the predecessor of $x$ is $p = count(r-1) + rank_r(S, col(c))$. Finally, the color associated with $x$ is given by $L[p]$.

**Theorem 1.** *Given a data structure that supports rank queries on strings of length $n$ over alphabet $[1,\sigma]$ in time $t(n,\sigma)$ and using $s(n,\sigma)$ bits of space, we can solve the colored predecessor problem for $n$ integers from universe $[1,n\sigma]$ in time $t(n,\sigma)+O(1)$ using a data structure that occupies $s(n,\sigma)+O(n)$ bits.*

By the reduction above we get that any lower bound for predecessor search for $n$ keys over a universe of size $n\sigma$ must also apply to *rank* queries on sequences of length $n$ over alphabet of size $\sigma$. In our case, if we aim at using $O(n \cdot w^{O(1)})$ bits of space, this lower bound (branch 2 [26]) is $\Omega\left(\lg \frac{\ell - \lg n}{\lg(s/n)+\lg w}\right) = \Omega\left(\lg \frac{\lg \sigma}{\lg w}\right)$.

For $\sigma = \Theta(n)$ and $w = \Theta(\lg n)$, the bound is simply $\Omega(\lg \lg \sigma)$. In case $\sigma = \omega(n)$, $\Omega(\lg \frac{\lg \sigma}{\lg w})$ must still be a lower bound, as otherwise we could break it in the case $\sigma = O(n)$ by just declaring $\sigma$ artificially larger.

**Theorem 2.** *Any data structure that uses space $O(n \cdot w^{O(1)})$ bits to represent a sequence of length $n$ over alphabet $[1,\sigma]$, must use time $\Omega(\lg \frac{\lg \sigma}{\lg w})$ to answer rank queries.*

For simplicity, assume $w = \Theta(\lg n)$. This lower bound is trivial for small $\lg \sigma = O(\lg \lg n)$ (i.e., $\sigma = O(\text{polylog } n)$), where constant-time solutions for *rank* exist that require only $nH_0(S) + o(n)$ bits [10]. On the other hand, if $\sigma$ is sufficiently large, $\lg \sigma = \Omega((\lg \lg n)^{1+\epsilon})$ for any constant $\epsilon > 0$, the lower bound becomes simply $\Omega(\lg \lg \sigma)$, where it is matched by known compact and compressed solutions [16, 1, 19] requiring as little as $nH_0(S) + o(nH_0(S)) + o(n)$ or $nH_k(S) + o(n \lg \sigma)$ bits.

The only range where this lower bound has not yet been matched is $\omega(\lg \lg n) = \lg \sigma = o((\lg \lg n)^{1+\epsilon})$, for any constant $\epsilon > 0$. The next section presents a new matching upper bound.

## 3 Optimal Upper Bound for *rank*

We now show a matching upper bound with optimal time and space $O(n \lg \sigma)$ bits. In the next sections we make the space succinct and even compressed.

We reduce the problem to predecessor search and then use an existing solution for that problem. The idea is simply to represent the string $S[1,n]$ over alphabet $[1,\sigma]$ as a matrix of $\sigma$ rows and $n$ columns, and regard the matrix as the set of $n$ points $\{(S[c]-1) \cdot n + c,\ c \in [1,n]\}$ over the universe $[1,n\sigma]$. Then we store an array of $n$ cells containing $\langle r, rank_r(S,c)\rangle$, where $r = S[c]$, for the point corresponding to column $c$ in the set.

To query $rank_r(S,c)$ we compute the predecessor of $(r-1) \cdot n + c$. If it is a pair $\langle r, v\rangle$, for some $v$, then the answer is $v$. Else the answer is zero.

This solution requires $n \lg \sigma + n \lg n$ bits for the pairs, on top of the space of the predecessor query. If $\sigma \le n$ we can reduce this extra space to $2n \lg \sigma$ by storing the pairs $\langle r, rank_r(S,c)\rangle$ in a different way. We virtually cut the string into chunks of length $\sigma$, and store the pair as $\langle r, rank_r(S,c) - rank_r(S, c - (c \bmod \sigma))\rangle$. The rest of the $rank_r$ information is obtained in constant time and $O(n)$ bits using Golynski et al.'s [16] reduction to chunks: They store a bitmap $A[1,2n]$ where the matrix is traversed row-wise and we append to $A$ a 1 for each 1 found in the matrix and a 0 each time we move to the next chunk (so we append $n/\sigma$ 0s per row). Then the remaining information for $rank_r(S,c)$ is $rank_r(S, c - (c \bmod \sigma)) = select_0(A, p_1) - select_0(A, p_0) - (c \operatorname{div} \sigma)$, where $p_0 = (r-1) \cdot n/\sigma$ and $p_1 = p_0 + (c \operatorname{div} \sigma)$ (we have simplified the formulas by assuming $\sigma$ divides $n$).

**Theorem 3.** *Given a solution for predecessor search on a set of $n$ keys chosen from a universe of size $u$, that occupies space $s(n,u)$ and answers in time $t(n,u)$, there exists a solution for rank queries on a sequence of length $n$ over an alphabet $[1,\sigma]$ that runs in time $t(n, n\sigma) + O(1)$ and occupies $s(n, n\sigma) + O(n \lg \sigma)$ bits.*

In the extended version of their article, Patrascu and Thorup [27] give an upper bound matching the lower bound of branch 2 and using $O(n \lg u)$ bits for $n$ elements over a universe $[1,u]$, and give hints to reduce the space to $O(n \lg(u/n))$. For completeness, we do this explicitly in an extended version of the present paper [5, App. A]. By using this predecessor data structure, the following result on *rank* is immediate.

**Theorem 4.** *A string $S[1,n]$ over alphabet $[1,\sigma]$ can be represented using $O(n \lg \sigma)$ bits, so that operation rank is solved in time $O(\lg \frac{\lg \sigma}{\lg w})$.*

Note that, within this space, operations *access* and *select* can also be solved in constant time.

## 4 Optimal-time *rank* in Succinct and Compressed Space

We start with a sequence representation using $n \lg \sigma + o(n \lg \sigma)$ bits (i.e., succinct) that answers *access* and *select* queries in almost-constant time, and *rank* in time $O(\lg \frac{\lg \sigma}{\lg w})$. This is done in two phases: a constant-time solution for $\sigma = w^{O(1)}$, and then a solution for general alphabets. Then we address compression.

### 4.1 Succinct Representation for Small Alphabets

Using multiary wavelet trees [10] we can obtain succinct space and $O(\frac{\lg \sigma}{\lg \lg n})$ time for *access*, *select* and *rank*. This is constant for $\lg \sigma = O(\lg \lg n)$. We start by extending this result to the case $\lg \sigma = O(\lg w)$, as a base case for handling larger alphabets thereafter. More precisely, we prove the following result.

**Theorem 5.** *A string $S[1,n]$ over alphabet $[1,\sigma]$ can be represented using $n \lg \sigma + o(n \lg \sigma)$ bits so that operations access, select and rank can be solved in time $O(\frac{\lg \sigma}{\lg w})$. If $\sigma = w^{O(1)}$, the space is $n \lceil \lg \sigma \rceil + o(n)$ bits and the times are $O(1)$.*

A multiary wavelet tree for $S[1,n]$ divides, at the root node $v$, the alphabet $[1,\sigma]$ into $r$ contiguous regions of the same size. A sequence $R_v[1,n]$ recording the region each symbol belongs to is stored at the root node (note $R_v$ is a sequence over alphabet $[1,r]$). This node has $r$ children, each handling the subsequence of $S$ formed by the symbols belonging to a given region. The children are decomposed recursively, thus the wavelet tree has height $O(\lg_r \sigma)$. Queries *access*, *select* and *rank* on sequence $S[1,n]$ are carried out via $O(\lg_r \sigma)$ similar queries on the sequences $R_v$ stored at wavelet tree nodes [18]. By choosing $r$ such that $\lg r = \Theta(\lg \lg n)$, it turns out that the operations on the sequences $R_v$ can be carried out in constant time, and thus the cost of the operations on the original sequence $S$ is $O(\frac{\lg \sigma}{\lg \lg n})$ [10].

In order to achieve time $O(\frac{\lg \sigma}{\lg w})$, we need to handle in constant time the operations over alphabets of size $r = w^\beta$, for some $0 < \beta < 1$, so that $\lg r = \Theta(\lg w)$. This time we cannot resort to universal tables of size $o(n)$, but rather must use bit manipulation on the RAM model.

The sequence $R_v[1,n]$ is stored as the concatenation of $n$ fields of length $\lg r$, into consecutive machine words. Thus achieving constant-time *access* is trivial: To access $R_v[i]$ we simply extract the corresponding bits, from the $(1+(i-1)\cdot \lg r)$-th to the $(i \cdot \lg r)$-th, from one or two consecutive machine words, using bit shifts and masking.

Operations *rank* and *select* are more complex. We will proceed by cutting the sequence $R_v$ into blocks of length $b = w^\alpha$ symbols, for some $\beta < \alpha < 1$. First we show how, given a block number $i$ and a symbol $a$, we extract from $R[1,b] = R_v[(i-1)\cdot b+1, i\cdot b]$ a bitmap $B[1,b]$ such that $B[j]=1$ iff $R[j]=a$. Then we use this result to achieve constant-time *rank* queries. Next, we show how to solve predecessor queries in constant time, for several fields of length $\lg w$ bits fitting in a machine word. Finally, we use this result to obtain constant-time *select* queries.

*Projecting a Block.* Given sequence $R[1,b] = R_v[1+(i-1)\cdot b, i\cdot b]$, which is of length $b \cdot \ell = w^\alpha \lg r < w^\alpha \lg w = o(w)$ bits, where $\ell = \lg r$, and given $a \in [1,r]$, we extract $B[1, b\cdot \ell]$ such that $B[j\cdot \ell] = 1$ iff $R[j]=a$.

To do so, we first compute $X = a \cdot (0^{\ell-1}1)^b$. This creates $b$ copies of $a$ within $\ell$-bit long fields. Second, we compute $Y = R \text{ XOR } X$, which will have zeroed fields at the positions $j$ where $R[j]=a$. To identify those fields, we compute $Z = (10^{\ell-1})^b - Y$, which will have a 1 at the highest bit of the zeroed fields in $Y$. Now $W = Z \text{ AND } (10^{\ell-1})^b$ isolates those leading bits.

*Constant-time rank Queries.* We now describe how we can do rank queries in constant time for $R_v[1,n]$. Our solution follows that of Jacobson [21]. We choose a superblock size $s = w^2$ and a block size $b = (\sqrt{w}-1)/\lg r$. For each $a \in [1,r]$, we store the accumulated values per superblock, $rank_a(R_v, i\cdot s)$ for all $1 \le i \le n/s$. We also store the within-superblock accumulated values per block, $rank_a(R_v, i\cdot b) - rank_a(R_v, \lfloor (i\cdot b)/s \rfloor \cdot s)$, for $1 \le i \le n/b$. Both arrays of counters require, over all symbols, $r((n/s)\cdot w + (n/b)\cdot \lg s) = O(nw^\beta (\lg w)^2/\sqrt{w})$ bits. Added over the $O(\frac{\lg\sigma}{\lg w})$ wavelet tree levels, the space required is $O(n\lg\sigma\lg w/w^{1/2-\beta})$ bits. This is $o(n\lg\sigma)$ for any $\beta < 1/2$, and furthermore it is $o(n)$ if $\sigma = w^{O(1)}$.

To solve a query $rank_a(R_v, i)$, we need to add up three values: $(i)$ the superblock accumulator at position $\lfloor i/s \rfloor$, $(ii)$ the block accumulator at position $\lfloor i/b \rfloor$, $(iii)$, the bits set at $B[1,(i \bmod b)\cdot\ell]$, where $B$ corresponds to the values equal to $a$ in $R_v[\lfloor i/b\rfloor\cdot b+1, \lfloor i/b\rfloor\cdot b+b]$. We have shown above how to extract $B[1, b\cdot\ell]$, so we count the number of bits set in $C = B$ AND $1^{(i \bmod b)\cdot\ell}$.

This counting is known as a popcount operation. Given a bit block of length $b\ell = \sqrt{w}-1$, with bits set at positions multiple of $\ell$, we popcount it using the following steps:

1. We first duplicate the block $b$ times into $b$ fields. That is, we compute $X = C\cdot(0^{b\ell-1}1)^b$.
2. We now isolate a different bit in each different field. This is done with $Y = X$ AND $(0^{b\ell}10^{\ell-1})^b$. This will isolate the $i$th aligned bit in field $i$.
3. We now sum up all those isolated bits using the multiplication $Z = Y\cdot(0^{b\ell+\ell-1}1)^b$. The end result of the popcount operation lies at the bits $Z[b^2\ell+1, b^2\ell+\lg b]$.
4. We finally extract the result as $c = (Z \gg b^2\ell)$ AND $(1^{\lg b})$.

*Constant-time select Queries.* The solution to *select* queries is similar but more technical. For lack of space we describe it in the extended version [5, Sec. 4.1.3].

### 4.2 Succinct Representation for Larger Alphabets

We assume now $\lg\sigma = \omega(\lg w)$; otherwise the previous section achieves succinctness and constant time for all operations.

We build on Golynski et al.'s solution [16]. They first cut $S$ into chunks of length $\sigma$. With bitvector $A[1,2n]$ described in Section 3 they reduce all the queries, in constant time, to within a chunk. For each chunk they store a bitmap $X[1,2\sigma]$ where the number of occurrences of each symbol $a\in[1,\sigma]$ in the chunk, $n_a$, is concatenated in unary, $X = 1^{n_1}01^{n_2}0\ldots1^{n_\sigma}0$. Now they introduce two complementary solutions.

*Constant-time Select.* The first one stores, for each consecutive symbol $a\in[1,\sigma]$, the chunk positions where it appears, in increasing order. Let $\pi$ be the resulting permutation, which is stored with the representation of Munro et al. [24]. This requires $\sigma\lg\sigma(1+1/f(n,\sigma))$ bits and computes any $\pi(i)$ in constant time and any $\pi^{-1}(j)$ in time $O(f(n,\sigma))$, for any $f(n,\sigma)\geq 1$. With this representation they solve, within the chunk, $select_a(i) = \pi(select_0(X,a-1)-(a-1)+i)$ in constant time and $access(i) = 1+rank_0(select_1(X,\pi^{-1}(i)))$ in time $O(f(n,\sigma))$.

For $rank_a(i)$, they basically carry out a predecessor search within the interval of $\pi$ that corresponds to $a$: $[select_0(X,a-1)-(a-1)+1, select_0(X,a)-a]$. They have a sampled predecessor structure with one value out of $\lg\sigma$, which takes just $O(\sigma)$ bits. With this structure they reduce the interval to size $\lg\sigma$, and a binary search completes the process, within overall time $O(\lg\lg\sigma)$.

To achieve optimal time, we sample one value out of $\frac{\lg \sigma}{\lg w}$ within chunks. We build the predecessor data structures of Patrascu and Thorup [27], mentioned in Section 3, over the sampled values. Added over all the chunks, these structures take $O((n/\frac{\lg \sigma}{\lg w}) \lg \sigma) = O(n \lg w) = o(n \lg \sigma)$ bits (as we assumed $\lg \sigma = \omega(\lg w)$). The predecessor structures take time $O(\lg \frac{\lg \sigma}{\lg w})$ (see Theorem 10 in the extended version [5, App. A]). The search is then completed with a binary search between two consecutive sampled values, which also takes time $O(\lg \frac{\lg \sigma}{\lg w})$.

*Constant-time Access.* This time we use the structure of Munro et al. on $\pi^{-1}$, so we compute any $\pi^{-1}(j)$ in constant time and any $\pi(i)$ in time $O(f(n,\sigma))$. Thus we get *access* in constant time and *select* in time $O(f(n,\sigma))$.

Now the binary search of *rank* needs to compute values of $\pi$, which is not anymore constant time. This is why Golynski et al. [16] obtained time slightly over $\lg \lg \sigma$ time for *rank* in this case. We instead set the sampling step to $(\frac{\lg \sigma}{\lg w})^{\frac{1}{f(n,\sigma)}}$. The predecessor structures on the sampled values still answer in time $O(\lg \frac{\lg \sigma}{\lg w})$, but they take $O((n/(\frac{\lg \sigma}{\lg w})^{\frac{1}{f(n,\sigma)}}) \lg \sigma)$ bits of space. This is $o(n \lg \sigma)$ provided $f(n,\sigma) = o(\lg \frac{\lg \sigma}{\lg w})$. On the other hand, the time for the binary search is $O(\frac{f(n,\sigma)}{f(n,\sigma)} \lg \frac{\lg \sigma}{\lg w})$, as desired.

The following theorem, which improves upon Golynski et al.'s [16] (not only as a consequence of a higher low-order space term), summarizes our result.

**Theorem 6.** *A string $S[1,n]$ over alphabet $[1,\sigma]$, $\sigma \leq n$, can be represented using $n \lg \sigma + o(n \lg \sigma)$ bits, so that, given any function $\omega(1) = f(n,\sigma) = o(\lg \frac{\lg \sigma}{\lg w})$, (i) operations access and select can be solved in time $O(1)$ and $O(f(n,\sigma))$, or vice versa, and (ii) rank can be solved in time $O(\lg \frac{\lg \sigma}{\lg w})$.*

For larger alphabets we must add a dictionary mapping $[1,\sigma]$ to the (at most) $n$ symbols actually occurring in $S$, in the standard way.

### 4.3 Zero-order Compression

Barbay et al. [1] showed how, given a representation $\mathcal{R}$ of a sequence in $n \lg \sigma + o(n \lg \sigma)$ bits, its times for *access*, *select* and *rank* can be maintained while reducing its space to $nH_0(S) + o(nH_0(S)) + o(n)$ bits. This can be done even if $\mathcal{R}$ works only for $\sigma \geq (\lg n)^c$ for some constant $c$. The technique separates the symbols according to their frequencies into $O(\lg n)$ classes. The sequence of classes is represented using a multiary wavelet tree [10], and the subsequences of the symbols of each class are represented with an instance of $\mathcal{R}$.

We can use this technique to compress the space of our succinct representations. By using Theorem 5 as our structure $\mathcal{R}$, we obtain the following result, which improves upon Ferragina et al. [10].

**Theorem 7.** *A string $S[1,n]$ over alphabet $[1,\sigma]$ can be represented using $nH_0(S)$ $+o(nH_0(S)) + o(n)$ bits so that operations access, select and rank can be solved* *in time $O(\frac{\lg\sigma}{\lg w})$. If $\sigma = w^{O(1)}$, the space is $nH_0(S) + o(n)$ and the operation times are $O(1)$.*

To handle larger alphabets, we use Theorem 6 as our structure $\mathcal{R}$. The only technical problem is that the subsequences range over a smaller alphabet $[1, \sigma']$, and Theorem 6 holds only for $\lg\sigma' = \omega(\lg w)$. In subsequences with smaller alphabets we can use Theorem 5, which give *access*, *select* and *rank* times $O(\frac{\lg\sigma'}{\lg w})$. More precisely, we use that structure for $\frac{\lg\sigma'}{\lg w} \leq f(n,\sigma)$, else use Theorem 6. This gives the following result, which improves upon Barbay et al.'s [1].

**Theorem 8.** *A string $S[1,n]$ over alphabet $[1,\sigma]$, $\sigma \leq n$, can be represented using $nH_0(S)+o(nH_0(S))+o(n)$ bits, so that, given any function $\omega(1) = f(n,\sigma) = o(\lg\frac{\lg\sigma}{\lg w})$, $(i)$ operations access and select can be solved in time $O(f(n,\sigma))$, and $(ii)$ rank can be solved in time $O(\lg\frac{\lg\sigma}{\lg w})$.*

## 4.4 High-order Compression

Ferragina and Venturini [11] showed how a string $S[1,n]$ over alphabet $[1,\sigma]$ can be stored within $nH_k(S) + o(n\lg\sigma)$ bits, for any $k = o(\lg_\sigma n)$, so that it offers constant-time *access* to any $O(\lg_\sigma n)$ consecutive symbols.

We provide *select* and *rank* functionality on top of this representation by adding extra data structures that take $o(n\lg\sigma)$ bits, whenever $\lg\sigma = \omega(\lg w)$. The technique is similar to those used by Barbay et al. [2] and Grossi et al. [19]. We divide the text logically into chunks, as with Golynski et al. [16], and for each chunk we store a monotone minimum perfect hash function (mmphf) $f_a$ for each $a \in [1,\sigma]$. Each $f_a$ stores the positions where symbol $a$ occurs in the chunk, so that given the position $i$ of an occurrence of $a$, $f_a(i)$ gives $rank_a(i)$ within the chunk. All the mmphfs can be stored within $O(\sigma\lg\lg\sigma) = o(\sigma\lg\sigma)$ bits and can be queried in constant time [4]. With array $X$ we can know, given $a$, how many symbols smaller than $a$ are there in the chunk.

Now we have sufficient ingredients to compute $\pi^{-1}$ in constant time: Let $a$ be the $i$th symbol in the chunk (obtained in constant time using Ferragina and Venturini's structure), then $\pi^{-1}(i) = f_a(i) + select_0(X, a-1) - (a-1)$. Now we can compute *select* and *rank* just as done in the "constant-time *access*" branch of Section 4.2. The resulting theorem improves upon Barbay et al.'s results [2] (they did not use mmphfs).

**Theorem 9.** *A string $S[1,n]$ over alphabet $[1,\sigma]$, for $\sigma \leq n$ and $\lg\sigma = \omega(\lg w)$, can be represented using $nH_k(S) + o(n\lg\sigma)$ bits for any $k = o(\lg_\sigma n)$ so that, given any function $\omega(1) = f(n,\sigma) = o(\lg\frac{\lg\sigma}{\lg w})$, $(i)$ operation access can be solved in constant time, $(ii)$ operation select can be solved in time $O(f(n,\sigma))$, and $(ii)$ operation rank can be solved in time $O(\lg\frac{\lg\sigma}{\lg w})$.*

To compare with the corresponding result by Grossi et al. [19] (who do use mmphfs) we can fix the redundancy to $O(\frac{n\lg\sigma}{\lg\lg\sigma})$, where they obtain $O(\lg\lg\sigma)$ time for *select* and *rank*, whereas we obtain the same time for *select* and our improved time for *rank*, as long as $\lg\sigma = \Omega(\lg w\lg\lg w\lg\lg\lg w)$.

## 5 Conclusions

This paper considerably reduces the gap between upper and lower bounds for sequence representations providing *access*, *select* and *rank* queries. Most notably, we give matching lower and upper bounds $\Theta(\lg \frac{\lg \sigma}{\lg w})$ for operation *rank*, which was the least developed one in terms of lower bounds. The issue of the space related to this complexity is basically solved as well: we have shown it can be achieved even within compressed space, and it cannot be surpassed within space $O(n \cdot w^{O(1)})$. On the other hand, operations *access* and *select* can be solved, within the same compressed space, in almost constant time (i.e., as close to $O(1)$ as desired but not both reaching it, unless we double the space).

There are still some intriguing issues that remain unclear:

1. Golynski's lower bounds [15] leave open the door to achieving constant time for *access* and *select* simultaneously, with $O(n(\lg \sigma)^2/\lg n)$ bits of redundancy. However, this has not yet been achieved for the interesting case $\omega(\lg w) = \lg \sigma = o(\lg n)$. We conjecture that this is not possible and a stronger lower bound holds.
2. While we can achieve constant-time *select* and almost-constant time for *access* (or vice versa) within zero-order entropy space, we can achieve only the second combination within high-order entropy space. If simultaneous constant-time *access* and *select* is not possible, then no solution for the first combination can build over a compressed representation of $S$ giving constant-time access, as it has been the norm $[2, 1, 19]$.
3. We have achieved high-order compression with almost-constant *access* and *select* times, and optimal *rank* time, but on alphabets of size superpolynomial in $w$. By using one Golynski's binary *rank*/*select* index [14] per symbol over Ferragina and Venturini's representation [11], we get high-order compression and constant time for all the operations for any $\sigma = o(\lg n)$. This leaves open the interesting band of alphabet sizes $\Omega(\lg n) = \sigma = w^{O(1)}$.

## References

1. J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich. Alphabet partitioning for compressed rank/select and applications. In *Proc. 21st ISAAC*, LNCS 6507, pages 315–326, 2010. Part II.
2. J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proc. 18th SODA*, pages 680–689, 2007.
3. J. Barbay and G. Navarro. Compressed representations of permutations, and applications. In *Proc. 26th STACS*, pages 111–122, 2009.
4. D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Monotone minimal perfect hashing: searching a sorted table with $o(1)$ accesses. In *Proc. 20th SODA*, pages 785–794, 2009.
5. D. Belazzougui and G. Navarro. New lower and upper bounds for representing sequences. *CoRR*, arXiv:1111.26211v1, 2011. `http://arxiv.org/abs/1111.2621v1`.
6. D. Clark. *Compact Pat Trees.* PhD thesis, University of Waterloo, Canada, 1996.

7. F. Claude and G. Navarro. Extended compact web graph representations. In *Algorithms and Applications (Ukkonen Festschrift)*, LNCS 6060, pages 77–91, 2010.
8. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM*, 57(1), 2009.
9. P. Ferragina and G. Manzini. Indexing compressed texts. *Journal of the ACM*, 52(4):552–581, 2005.
10. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms*, 3(2):article 20, 2007.
11. P. Ferragina and R. Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, 2007.
12. T. Gagie. Large alphabets and incompressibility. *Information Processing Letters*, 99(6):246–251, 2006.
13. T. Gagie, G. Navarro, and S. Puglisi. Colored range queries and document retrieval. In *Proc. 17th SPIRE*, LNCS 6393, pages 67–81, 2010.
14. A. Golynski. Optimal lower bounds for rank and select indexes. *Theoretical Computer Science*, 387(3):348–359, 2007.
15. A. Golynski. Cell probe lower bounds for succinct data structures. In *Proc. 20th SODA*, pages 625–634, 2009.
16. A. Golynski, I. Munro, and S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proc. 17th SODA*, pages 368–373, 2006.
17. A. Golynski, R. Raman, and S. Rao. On the redundancy of succinct data structures. In *Proc. 11th SWAT*, LNCS 5124, pages 148–159, 2008.
18. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pages 841–850, 2003.
19. R. Grossi, A. Orlandi, and R. Raman. Optimal trade-offs for succinct string indexes. In *Proc. 37th ICALP*, pages 678–689, 2010.
20. A. Gupta, W.-K. Hon, R. Shah, and J. Vitter. Dynamic rank/select dictionaries with applications to XML indexing. Technical Report CSD TR #06-014, Purdue University, July 2006.
21. G. Jacobson. Space-efficient static trees and graphs. In *Proc. 30th FOCS*, pages 549–554, 1989.
22. G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.
23. I. Munro. Tables. In *Proc. 16th FSTTCS*, LNCS 1180, pages 37–42, 1996.
24. J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations. In *Proc. 30th ICALP*, pages 345–356, 2003.
25. M. Patrascu. Succincter. In *Proc. 49th FOCS*, pages 305–313, 2008.
26. M. Patrascu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th STOC*, pages 232–240, 2006.
27. M. Patrascu and M. Thorup. Time-space trade-offs for predecessor search. *CoRR*, cs/0603043v1, 2008. `http://arxiv.org/pdf/cs/0603043v1`.
28. M. Patrascu and E. Viola. Cell-probe lower bounds for succinct partial sums. In *Proc. 21st SODA*, pages 117–122, 2010.
29. N. Välimäki and V. Mäkinen. Space-efficient algorithms for document retrieval. In *Proc. 18th CPM*, LNCS 4580, pages 205–215, 2007.