CREMSA: Compressed indexing of (ultra) large alignments

Mikaël Salson, Thomas Baudeau, Arthur Boddaert, Awa Bousso Gueye, Laurent Bulteau, Yohan Hernandez–Courbevoie, Camille Marchet, Nan Pan, Sebastian Will, Yann Ponty

Journal Title Here, 2024, 1–9
doi: DOI HERE
Advance Access Publication Date: Day Month Year
Paper

CREMSA: Compressed indexing of (ultra) large alignments

Mikaël Salson[1,*] **Thomas Baudeau**[1] **Arthur Boddaert**[2] **Awa Bousso Gueye**[2] **Laurent Bulteau**[3] **Yohan Hernandez--Courbevoie**[2] **Camille Marchet**[1] **Nan Pan**[4] **Sebastian Will**[4] **and Yann Ponty**[4,*]

[1]Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France, [2]Département d'Informatique, Lille University, France, [3]CNRS UMR 8049 LIGM, Gustave Eiffel University, Street, Postcode, State, France and [4]CNRS UMR 7161 LIX, Ecole Polytechnique, Institut Polytechnique de Paris, Street, Postcode, State, France

*Corresponding authors. mikael.salson@univ-lille.fr; yann.ponty@lix.polytechnique.fr

FOR PUBLISHER ONLY Received on Date Month Year; revised on Date Month Year; accepted on Date Month Year

Abstract

Recent viral outbreaks motivate the systematic collection of pathogenic genomesin order to accelerate their study and monitor the apparition/spread of variants. Due to their limited length and temporal proximity of their sequencing, viral genomes are usually organized, and analyzed as oversized Multiple Sequence Alignments (MSAs). Such MSAs are largely ungapped, and mostly homogeneous on a column-wise level but not at a sequential level due to local variations, hindering the performances of sequential compression algorithms.

In order to enable an efficient handling of MSAs, including subsequent statistical analyses, we introduce `CREMSA` (Column-wise Run-length Encoding for Multiple Sequence Alignments), a new index that builds on sparse bitvector representations to compress an existing or streamed MSA, all the while allowing for an expressive set of accelerated requests to query the alignment without prior decompression.

Using `CREMSA`, a 65GB MSA consisting of 1.9M SARS-CoV 2 genomes could be compressed into 22MB using less than half a gigabyte of main memory, while executing access requests in the order of 100ns. Such a speed up enables a comprehensive analysis of covariation over this very large MSA. We further assess the impact of the sequence ordering on the compressibility of MSAs and propose a resorting strategy that, despite the proven NP-hardness of an optimal sort, induces greatly increased compression ratios at a marginal computational cost.

Key words: Indexing, Multiple sequence alignment, Compression

Introduction

The analysis of MSAs enables an evaluation of key metrics to understand molecular evolution, including conservation, coevolution, evolutionary distances, and other higher-order statistics. For instance, in viruses whose genetic material consists of single-stranded nucleic acids (ssRNA viruses), evolutionary constraints at the structural level can be revealed by covariation analysis. Such analyses motivate the analysis of the joint content of columns pairs, to assess the propensity of genomic positions to form a base pair mediated by hydrogen bonds. Ultimately, they enable a reconstruction of RNA architecture(s), potentially revealing targets for future drugs [Triebel et al., 2024].

Gene or genome-based alignments may feature extreme level of conservation at the column level, reflecting compact genomes undergoing pervasive evolutionary pressure. Such is the case of the genomic material of pathogens, collected upon an outbreak to monitor their evolution. This results in the presence of multiple near-identical sequences within alignments, featuring highly-homogeneous column contents. Compressing such alignments, especially at a column-wise level, can lead to spectacular compression ratios [Deorowicz et al., 2018]. However, such prior representations are mainly static: they require the MSA to be fully loaded in memory prior to their creation; they cannot be conveniently updated upon insertion of a new sequence; and most analyses will require a full uncompression of the alignment.

In this work, we introduce a new compressed index, called CREMSA (Column-wise Run-length Encoding for Multiple Sequence Alignments) which greatly reduces the storage required to store column-wise redundant MSAs. Contrasting with earlier efforts, solely focusing on the file-level compression of an MSA [Deorowicz et al., 2018], our index enabling direct and efficient access to column-wise statistics (no full decompression needed).

Our main contributions are the following:

- We introduce CREMSA, a novel compressed index for MSAs which supports optimized row and column-wise analyses;
- We investigate the impact of the genome order on the compressibility of MSAs. We formalize the problem and demonstrate its computational intractability
- We propose and benchmark several heuristic strategies for reordering MSAs. In particular, a lexicographic sort, based on a subset of lower-sequence identity columns, is seen to greatly optimize the compressibility of MSAs, even outperforming phylogenetic-based reordering. Moreover, this result does not depend on the compression algorithm being used;
- Beyond the reduced storage, CREMSA enables sigificantly faster access to column-wise statistics, enabling a covariation analysis of MSAs representing the evolution of large viral RNAs.

Materials and methods

Datasets

Hunt et al. [2024] have re-processed millions of SARS-CoV-2 sequencing runs in order to provide accurate and reliable assemblies. They made 4.5 million SARS-CoV-2 assemblies available. From those sequences, we selected the most precise ones by removing all the sequences with at least 100 indeterminate nucleotides (any IUPAC nucleotide different from the canonical ones). Sequences which had at least 2 consecutive Ns, except at their ends, were also removed. This filtering was achieved to prevent too many gaps to be introduced by the multiple sequence alignment. In the end, we obtained 1,870,492 SARS-CoV-2 sequences that were aligned using Halign3 [Tang et al., 2022], the only multiple sequence aligner to run on a cluster node under 1.5TB RAM. The final alignment consists of 34,830 columns, totaling 65GB of data.

The CREMSA index

The central data-structure in our work is the *bit vector*: a $\{0,1\}$-array with fast *rank* and *select* queries. For a length-n bit vector B, $rank_1(B,i)$ if the number of 1s in the length-i prefix of B and $select_1(B,i)$ is the position of the i-th one in B, or $n+1$ whenever i is greater than the number of ones in B.

Index definition

Our compressed self-index, termed CREMSA, exploits the succession of identical letters (called *runs*) in the columns of the MSA. CREMSA indexes each column of the MSA independently to provide efficient column-wise queries. Each column C_j of length s of a multiple sequence alignment is stored using an approach close to run-length encoding by decomposing the column in two components: 1) a (compressed) bit vector B_j that identifies the start of each run in the column C_j and 2) a string that stores the repeated nucleotide of each run.

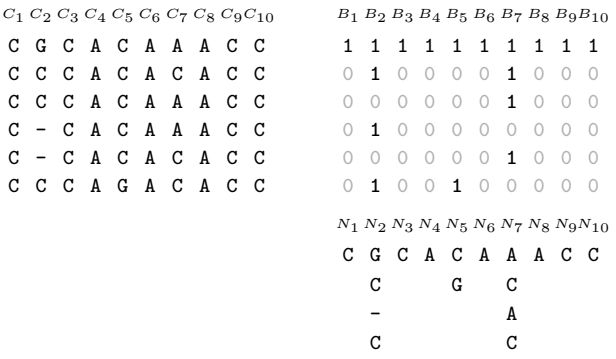

Fig. 1: Left: a multiple sequence alignment of $s = 6$ sequences of length $n = 10$. Right: the representation of this MSA with the CREMSA index. The number of runs in each column C_j can be identified by the number of 1s in B_j. For instance, in column 2, the number of runs $r_2 = 4$. The strings N_j are represented vertically, *eg.* $N_7 =$ ACAC. The last nucleotide in N_7 is a C because the last 1 in B_7 corresponds to the position in C_7 where a C is stored.

More formally, we define B_j of length n, as follows: $B_j[i] = 1$ iff $i = 1$ or $C_j[i] \neq C_j[i-1]$, for $1 \leq i \leq s$. The string N_j has a length of r_j, the number of runs in C_j, and is defined with: $N_j[i] = C_j[select_1(B_j, i)]$, for $1 \leq i \leq r_j$. An example is given in Figure 1.

Supported queries

Using *rank* and *select* queries on the bit vectors, the index supports the following basic queries:

- *access*(i,j) which returns the j-th nucleotide from the i-th sequence or, otherwise stated, the i-th nucleotide from C_j;
- *count-consecutive*(i,j) which counts the number of consecutive occurrences of $C_j[i]$ in C_j, *ie.* the length of the run of $C_j[i]$ in C_j at position i.

The *access*(i,j) query is solved using $N_j[rank_1(B_j,i)]$. The *count-consecutive*(i,j) query first identifies the first and final positions of the run, respectively b and e. The first position of the run is obtained with $b = select_1(B_j, rank_1(B_j,i))$ and the final position with $e = select_1(B_j, rank_1(B_j,i)+1) - 1$, therefore, *count-consecutive*$(i,j) = select_1(B_j, rank_1(B_j,i)+1) - select_1(B_j, rank_1(B_j,i))$.

Using those queries, one can devise more sophisticated queries in order to recover a complete sequence, or to compute the counts of each nucleotide in a column, the empirical entropy of a column, the empirical entropy of a pair of columns, the mutual information of a pair of columns, the G-test as promoted by R-scape [Rivas et al., 2017], the conservation score of RNAalifold [Bernhart et al., 2008], etc.

Counting nucleotides. The counts of each nucleotide in a column j are obtained by iterating on each 1 in B_j (using $select_1$) and by computing *count-consecutive* for the position of each 1 as shown in Algorithm 1. Once the counts have been obtained, computing the empirical entropy and other site-level metrics is straightforward.

Those queries can easily be generalized to more than one column in order to study the covariations between columns. For instance, *count-consecutive*(i, j_1, j_2) would return the length of the

Algorithm 1 Counting occurrences of all nucleotides/gaps in column j

```
function COUNT(j)
    occurrences = {}
    s ← |B_j|
    i ← 1
    b ← 1
    while i ≤ s do
        m ←count-consecutive(i, j)
        Increment occurrences[access(i, j)] by m
        i ← select_1(B_j, b + 1)
        b ← b + 1
    end while
    return occurrences
end function
```

run of $C_{j_1}[i] \cdot C_{j_2}[i]$ in the pair of columns C_{j_1}, C_{j_2} at position i. This can be achieved by identifying the start position b of the run at position i in B_{j_1} and B_{j_2}, and by keeping the highest one. Conversely, for the end position e, we keep the lowest one among the ones in B_{j_1} and in B_{j_2}. We can compute this with $b = \max(select_1(B_{j_1}, rank_1(B_{j_1}, i)), select_1(B_{j_2}, rank_1(B_{j_2}, i)))$ and $e = \min(select_1(B_{j_1}, rank_1(B_{j_1}, i)), select_1(B_{j_2}, rank_1(B_{j_2}, i)))$. Then, $count\text{-}consecutive(i, j_1, j_2) = e - b + 1$. We can generalize the approach for $count\text{-}consecutive(i, j_1, j_2, \ldots, j_k)$. Using those counts, one can then compute the empirical entropy for a pair of columns, or for k columns.

Index construction

To construct `CREMSA`, from a multiple sequence alignment with s rows and n columns, a naive solution consists in initializing n length-s bit vectors. Then by comparing each sequence in the MSA to the next one, it is straightforward to identify the start of the runs in each bit vector.

Offline construction. However, from a practical point of view, this requires to store $n \times s$ bits in memory during the construction, before the bit vectors are actually compressed, which would be several orders of magnitude larger than the final index. One solution would be to use dynamic bit vectors [Prezza, 2017]. They would always be compressed, preventing the issue of storing $n \times s$ plain bits, while allowing one to update them to set bits at 1 during the construction. This advantage however comes with a logarithmic penalty at query time. Another solution is to build columns by chunks instead of building them altogether. Let c be the size of the chunk, thus when reading the multiple sequence alignment for the first time, only the c first columns of `CREMSA` are built. With this solution, the multiple sequence alignment has to be read $\lceil \frac{n}{c} \rceil$ times, which prevents an online construction.

Online construction. To overcome this limitation, we also propose an algorithm for processing large multiple sequence alignment in a streaming manner. The goal is to split the MSA into manageable bundles of genomes (entire rows), compress and store the data efficiently, and later reconstruct the complete MSA. The process can therefore construct an MSA in different parts, and can also be used to handle large-scale datasets in a partitioned way to optimize memory and disk usage.

The input MSA is read line-by-line (genome-by-genome), and the general structure remains: each column is stored in a bitmap and a character vector. We expect to read a certain amount of lines, that we call bundle, each bundle is then written to disk when it reaches a limit size, to ensure memory efficiency.

From the disk, bundles are merged column by column. The very first chunk serves for the initialisation of the MSA, then the next bundle is merged to it to become the current_MSA. The current_MSA representation remains the similar than the main construction algorithm, where compressed bit vectors are associated to strings for columns representation. Each bundles are merged to the current_MSA until there are no more bundles.

During merge, the last character in each column of the current_MSA (or of the initial MSA at the first iteration) is compared to the first character of the corresponding column in the next chunk to be merged. If the two characters are equal, the first value of the bundle to be merged can be skipped when editing the current_MSA's bit vector and strings.

Complexities

When constructing CREMSA offline with chunks of c columns ($1 \leq c \leq n$), the construction needs to keep in memory c columns, hence $\Theta(sc)$ space, and the input of size sn has to be read $\lceil \frac{n}{c} \rceil$ times, hence $\Theta\left(\frac{sn^2}{c}\right)$ time. This strategy offers a trade-off between time and space at construction. However, in that case an online construction of the structure comes at the cost of a prohibitive space consumption.

As the *rank* and *select* operations can be computed in constant time on compressed bit vectors [Navarro and Mäkinen, 2007], the *access* and *count-consecutive* queries are performed in constant time. Any genome of length n can thus be retrieved in $\Theta(n)$ time, making CREMSA a self-index as it can recover the input data. Counting the occurrences of each nucleotide or computing the empirical entropy of a column j is obtained in $\Theta(r_j)$ as the while loop in Algorithm 1 iterates on each 1 in B_j. Counting pair of nucleotides in pair of columns j_1, j_2 follows the same principle, hence the time complexity in $\Theta(\max(r_{j_1}, r_{j_2}))$.

Since our genomes are highly redundant, the bit vectors B_j are expected to be very sparse. In such cases, the compressed bit vectors only consist in storing the relative positions of the 1s in each bit vector, which leads to a $\Theta(r_j)$ space complexity. When B_j is dense, the space complexity is $O(H_0(B_j))$, where H_0 is the zero-th order empirical entropy of B_j.

Both our time and space complexities depend on r_j the number of runs in column j, meaning that decreasing this value makes the index more time and space efficient.

Improving compressibility through sequence reordering

The number of runs in a column of the MSA critically and directly depends on how the genomes are ordered in the MSA. Ideally, we would like to find an ordering which maximizes the average lengths of runs across columns, *i.e.* minimizes the total numbers of runs.

Unfortunately, finding an ordering of the sequences minimizing the total number of runs is NP-complete, as can be seen with the following simple reduction from Hamiltonian Path. Given a graph $G = (V, E)$ where all vertices have degree 3, build $|V|$ length-$|E|$ sequences over alphabet $\{\mathtt{A}, \mathtt{U}\}$, such that the jth character of sequence i is `U` iff edge j is incident to vertex i. Then each column has 5 runs in the worst case: two length-1 runs of `U` separating three runs of `A`. The number of runs decreases by 1 or 2 if the column starts and/or ends with `U`; independently of the order of the rows, this amounts to exactly 6 missing runs in total since

each vertex has degree 3. Also, the number of runs in column j decreases by 2 if both occurrences of `U` are consecutive, in which case we say that edge j is *realized*. The overall number of runs is thus $5|E|-6-2\times$number of realized edges. Each pair of consecutive rows can realize at most one edge (if the corresponding vertices are adjacent in G), so the number of runs is at least $5|E|-6-2(|V|-1)$, and this bound is attained if and only if the ordering of the vertices chosen for each row forms an hamiltonian path in G.

Even though finding an optimal order is computationally intractable, simple heuristics may still help find better orders than a random one. Břinda et al. [2024] addressed a similar problem: they used phylogenetic information in order to improve compression ratios of microbial genomic sequences. In particular, they reordered 590,779 SARS-CoV-2 genomes using GISAID phylogeny. They achieved a 1,647 compression ratio sorting sequences by their phylogenetic order and compressing them with `xz`. The same order can also benefit `CREMSA`: clustering the most similar sequences together should produce longer runs.

We also introduce a new ordering, specific to multiple sequence alignments that does not depend on an external (or heavy to compute) information, such as the phylogeny of the sequences. Let $G_1,\ldots,G_s$ the s genomes in the MSA. Let occ_j be an array of the number of occurrences of each character in column C_j, then the percentage of identity id_j in C_j is $id_j = \frac{\max(occ_j)}{s}$. Using the COUNT function on `CREMSA`, one can quickly compute the percentage of identity in C_j. Thus, we will extract the d positions $p_1,\ldots,p_d$ of the columns having the lowest percentage of identity, with $id_{p_1} < id_{p_2} < \cdots < id_{p_d}$. Then, for each genome G_i, we extract a word w_i of d characters, such that $w_i = G_i[p_1]\cdot G_i[p_2]\cdots G_i[p_d]$, this is a shuffled subword of G_i. We call those d-length shuffled subwords d-discriminative subwords. The genomes $G_1,\ldots,G_s$ are eventually sorted according to the lexicographic order of the d-discriminative subwords. Once the positions $p_1,\ldots,p_d$ are identified, the d-discriminative subwords can be sorted in linear time $\Theta(sd)$ and space $\Theta(s)$ using a bucket sort.

Entropy and covariation metrics for RNA comparative analysis

Columnwise entropy is a common metric of the local diversity of MSA. It is defined for a column i as

$$H_i = \sum_x p_x^i \log(p_x^i)$$

where p_x^i denotes the frequency of character x in the i-th column.

Various covariation measures have been discussed in the literature; among them several variations of the general theme of mutual information [Gutell et al., 1994]. In this work, we provide data for two of these scores. First, we study the G-test [Woolf, 1957], which was found to have best performance in `R-scape` [Rivas et al., 2017]. in their comparison of eight different covariation scores. The G-test score $\mathrm{GT}(i,j)$ for two alignment columns i and j is defined as

$$GT(i,j) = 2\sum_{x,y} N_{ij} p_{x,y}^{i,j} \log \frac{p_{x,y}^{i,j}}{p_x^i p_y^j},$$

where for nucleotides x and y, as well as columns i and j, $N_{i,j}$, $p_{x,y}^{i,j}$, p_x^i, p_y^j respectively denote the number of base pairs; the probability of the nucleotide pair x,y; the probability of x in i and the probability of y in j. In this score we consider only determined nucleotides A, C, T/U, or G and their pairings, but consider all kinds of canonical pairings, i.e. AU, GC, or GU, and non-canonical pairings.

Second, we consider the RNAAlifold score as introduced by Bernhart et al. [2008] to predict canonical RNA secondary structures based on MSAs. This score distinguishes between canonical pairing and non-canonical pairing, which is penalized as incompatibility. The score is defined as sum of two components, a covariation score $g_{\mathrm{cov}}(i,j)$ and an incompatibility score $g_{\mathrm{inc}}(i,j)$. The covariation score is a sum-of-pairs score over the Hamming distance of canonical base pairs occurring in columns i and j in all pairs of alignment rows. The incompatibility score penalizes gap symbols in columns i and j as well as non-canonical base pairs.

Results

`CREMSA` is implemented in C++ using the sdsl-lite v3 library[1] and is available at `https://gitlab.univ-lille.fr/cremsa/cremsa`, or as a Docker image at `https://hub.docker.com/repository/docker/mikaels/cremsa/` under a GNU GPL v3 license. The online algorithm's implementation is availablee at `https://gitlab.cristal.univ-lille.fr/bonsai/msa_streaming`. `CREMSA` uses two types of bit vectors in sdsl-lite depending on the number of runs in the bit vector: either SD bit vector or RRR bit vector. A SD bit vector is intended to support sparse bit vectors, thus we use them in `CREMSA` for any bit vector B_j with $r_j/s < .1$. In the other cases, a RRR bit vector is used because it is less space consuming with denser bit vectors. In practice, the *rank* operation is in $O(\log(\frac{s}{r_j}))$ in the SD vector and $O(1)$ for the RRR vector. The *select* operation is in $O(1)$ for SD vectors and $O(\log s)$ for RRR vector. The offline construction of the index is currently implemented in `CREMSA`. As of now, the online construction is a proof-of-concept in a standalone implementation. Sorting the sequences by their d-disciminative subwords is not included in `CREMSA` yet either, but it is managed by an external Python script which sorts the sequences on disk.

We built our `CREMSA` index on the 1,870,492 SARS-CoV-2 genomes. When storing the `CREMSA` index on disk, the compressed bit vectors as well as the nucleotides stored in each N_j string are compressed using gzip. When performing the queries, the whole index is loaded in memory and both the bit vectors and the N_j strings are un-gzipped. Note that the bit vectors are still compressed by the scheme used in the SDSL library. As far as we know, there exists no other index structure for multiple alignments. However we compare our compression results to other compressors.

All experiments were performed on the single thread of a computer with a 12-core Intel i7-12700, 64GB of memory and a 1TB hard drive.

Compression ratios

`CREMSA` stores a multiple sequence alignment in a compressed representation that also allows efficient queries. However `CREMSA` does not store the identifiers of the sequences it represents. For the sake of comparison, the compressors were assessed on the basis of the sequences only, the sequence identifiers were not taken into account. In Table 1 we present the compression ratios of `CREMSA`, of two generic compressors (`gzip` and `xz`) and one

[1] `https://github.com/xxsds/sdsl-lite`

	Compression ratio	Runtime (s)	Memory (MB)
`gzip`	3	9,742	2
`xz`	2,087	7,974	97
`CoMSA`	5,285	5,196	64,400
`CREMSA`	2,922	1,365	326

Table 1. Compression ratios of various compressors on the reordered multiple sequence alignment of 1,870,492 SARS-CoV-2 genomes (65GB uncompressed). The higher the compression ratio, the better the compression. `gzip` and `xz` were launched with the default option and with the best compression level (`-9`). The results with the latter were similar or even worse, thus only the default option is shown.

specialized compressor dedicated to multiple sequence alignments (`CoMSA` [Deorowicz et al., 2018]) on the 1.9M genomes reordered according to our heuristic based on 5,000-discriminative subwords. Note that CoMSA was launched on another computer (with 128GB RAM) as its memory usage exceeded the memory available on the computer used for our experiments.

Unsurprisingly, `CoMSA` is the best compressor, however, this comes at the cost of prohibitive memory usage. `CREMSA` is very competitive in terms of compression ratio compared to `xz`. In the reordered version, `CREMSA` even compresses better than `xz` while being almost 9 times faster. `gzip` performances are poor because of its look-back buffer whose size is only 32,768 characters long, slightly too small to contain more than a single genome (the aligned genomes are 34,830 characters long).

As introduced previously, the order of the sequences matters for `CREMSA` in order to minimize the number of runs in each column. Reordering the sequences also benefits dictionary-based compressors since similar sequences are closer. We assessed the impact of the following orders on the compression:

1. by increasing length of the genomes;
2. by phylogenetic order (*ie.* the lexicographical order of their pango lineage);
3. by phylogenetic order and length of the genomes;
4. by the 5,000-discriminative subwords
5. by 5,000-random subwords (same idea as before but the 5000 columns are chosen randomly instead of using the percentage of identity)

In Table 2 we show how the order influences the size of the compressed representation of the sequences in the MSA. For the d-discriminative subwords (and the random ones), the 1.9M genomes were reordered on disk in 10 minutes on a SSD drive. As mentioned previously this could be implemented more efficiently with a bucket sort.

The phylogenetic order improves compression in all the cases, confirming the result by Břinda et al. [2024]. Adding information about the length of the sequences improves the compression for `CREMSA` but decreases it for `xz`. This is probably due to the nature of the compression methods. While `CREMSA` compresses column-wise, where it is an advantage to have gaps at the ends clustered together, `xz` compresses sequence-wise, where there is no such advantage. Finally, the discriminative order allows the best compression ratios for all the methods (except `gzip` that cannot benefit from the redundancy for reasons explained previously). Under that order, adding phylogenetic information does not bring any improvement.

The performance of the discriminative subwords increases with longer subwords and then plateau at 5,000 (data not shown).

Order	`gzip`	`xz`	`CoMSA`	`CREMSA`
Initial	3	1,211	4,628	965
Length	3	1,265	4,770	1,135
Phylogenetic	3	1,537	4,891	1,316
Phylo+length	3	957	4,962	1,741
Random sample	3	1,006±5	5,035±12	1,204±103
Discriminative	3	2,087	5,285	2,922
Discriminative+phylo	3	2,087	5,285	2,925

Table 2. Compression ratios on the multiple sequence alignment of 1,870,492 SARS-CoV-2 genomes (65GB uncompressed) reordered in various ways. The higher the compression ratio, the better the compression. The order are the following ones. Length: by genome length, phylogenetic: by pango lineage, phylo+length: by pango lineage and genome length, random sample: by 5,000 random subwords (the process was repeated 50 times, the average and standard deviation are shown), discriminative: by the 5,000-discriminative subwords, discriminative+phylo: by the latter and by the pango lineage.

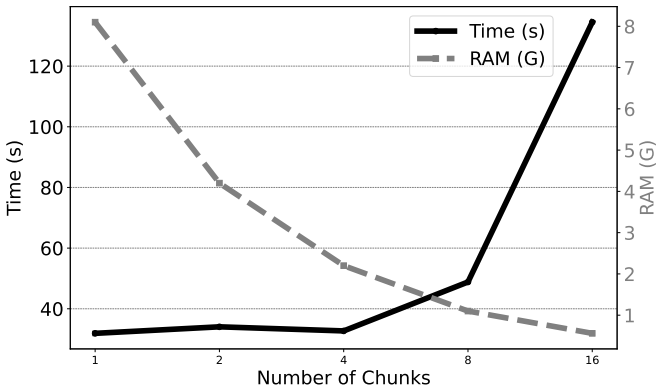

Fig. 2: Time and RAM, observed for different chunk sizes, for the online index construction of on 100,000 SARS-CoV-2 genomes.

Surprisingly the performances of the discriminative order is not impacted when shuffling the disciminative subwords used.

Under the best order, in `CREMSA` 1,473 bit vectors are stored using a RRR bit vector, this number increases to 4,214 for the initial order.

Runtime analysis

Construction time. The construction of the `CREMSA` index offers a space-time trade-off. By default, `CREMSA` loads 1,000 columns in memory, builds the bit vectors and the N_j strings and compresses the bit vectors. Once all the columns have been processed, the bit vectors and the N_j strings are gzipped and the index is written on disk. The time and memory consumption of `CREMSA` construction on 1.9M genomes is shown in Table 1. Obviously, loading more columns reduces the construction time and increases the memory consumption. When the construction is performed by loading 10,000 columns in memory, the time consumption is reduced 5 fold and the space consumption increased 7 fold (resp. 271s and 2,369MB). In comparison to 10,000 columns, loading all the alignment in memory only marginally improves times consumption but increases space consumption 4 fold (resp. 168s and 9,378MB).

We benchmarked the online construction algorithm for construction time and RAM. The online construction never reaches

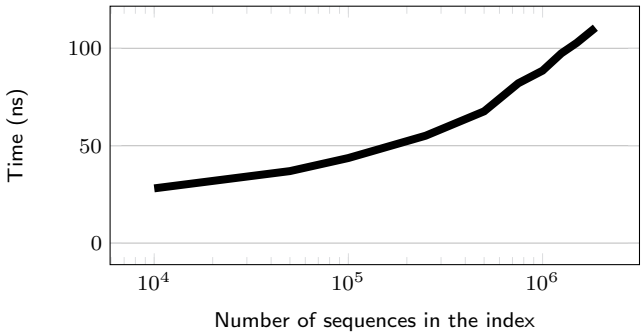

Fig. 3: Average time per nucleotide to perform the *access* query on an index of 10,000 to 1.9M SARS-CoV-2 genomes.

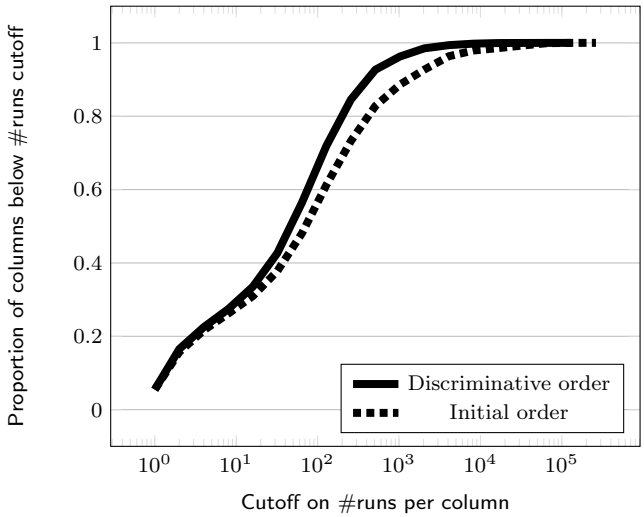

Fig. 4: Detailed impact of reordering. Cumulative distribution of the number of runs in each column, as induced by the initial and optimized row ordering. Our optimized *discriminative* order greatly increases the proportion of columns with #runs in the $[10^2, 10^3]$ range, while greatly depleting columns with #runs greater than 5.10^3, resulting in an MSA with improved compressibility.

the entire size of the original MSA in RAM thanks to the use of buffers in the implementation. The offline method, based on chunks of columns, processes 1.9M genomes in 1,365 seconds (1,392 genomes per second), whereas this second approach processes 100,000 genomes in just 30 to 135 seconds (740 to 3,333 genomes per second), depending on the sequence chunk size (Figure 2). Note that the bit vectors used in the protoype for the latter approach is less time-efficient that the one used in the offline method implemented in `CREMSA`.

Access to a random nucleotide. We recall that the purpose of `CREMSA` is not only to compress a multiple sequence alignment but also to provide efficient queries on it. For instance, `CREMSA` provides random access to any nucleotide in the multiple sequence alignment, something that competing compressors cannot achieve without first decompressing the whole content of a column (or alignment).

We assessed the *access* query by retrieving 10,000 sequences randomly chosen from `CREMSA` indices containing from 10,000 to 1.9M sequences. The experiments were repeated 10 times. The mean time for the *access* query is shown for each experiment in Fig. 3.

Our experiments show that the *access* query is not performed in constant time. This is due to the implementation of the sparse bit vector we chose, which doesn't have a constant time complexity for *rank* queries, as explained previously. However, when the number of sequences increases ∼190 fold, the access time increases less than 4 fold, showing that the *access* query scales very well, at about 100ns for an index of 1.9M genomes.

Impact of the number of runs per column. The time complexities directly depend on the number of runs in each column of the MSA. As expected, from the compression ratios, the number of runs is lower in the discriminative order compared to the initial order. More precisely, there are more than .1% of runs (*ie.* more than 1,870) in less than 5% of the columns in the discriminative order, and in 12% of the columns in the initial order (see Figure 4). The median number of runs is 95 (0.005%) with the discriminative order and 141 (0.008%) in the initial order.

Counting nucleotides in a column. The runtime of queries on the columns depend on their number of runs. We compared the runtime for the COUNT function in columns with varying number of runs, from 1 to almost 10,000. For each $i \in \{1, \ldots, 4\}$, 500 columns were randomly chosen with a number of runs between $.5 \cdot 10^i$ and $2 \cdot 10^i$. For each selected columns, the COUNT function is run 10 times. The results are shown in Figure 5.

We remind that the time complexity of the COUNT query is linear in the number of runs ($\Theta(r_j)$). In practice, other considerations come into play such as cache efficiency making COUNT more efficient on columns with few runs than those with many runs. For instance, the time consumption increases by a factor less than 2 between $r_j = 1$ and $r_j = 9$, while it increases 7 fold between $r_j = 993$ and $r_j = 8917$.

Queries on multiple columns. It could be argued that an index is not required to compute counts on each column, since such statistics could be precomputed in a single run, and subsequently queried directly. However, this is not the case anymore when one wants

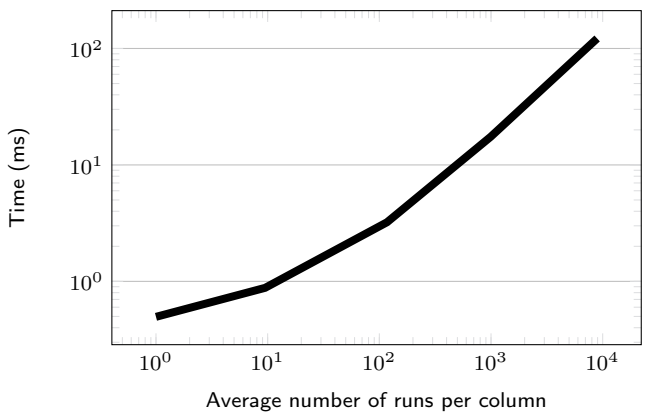

Fig. 5: Average time to count occurrences of nucleotides in a single column of 1.9M genomes, depending on its number of runs.

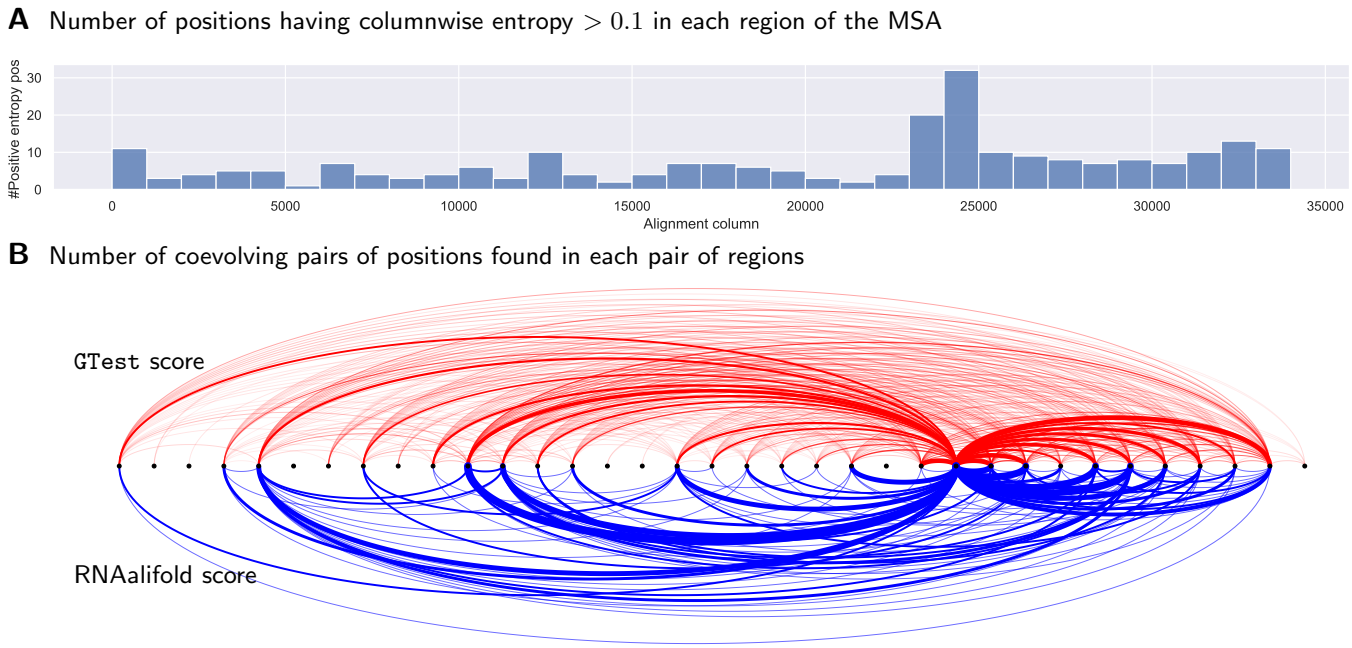

Fig. 6: Main coevolving regions and entropic positions derived from an ultra-large MSA consisting of 1,870,492 SARS-CoV 2 genomes. The top panel (A) represents as a barplot the number of positions, in each region (1k nts slice) of the genome, associated with a columnwise entropy greater than 0.1. The bottom panel (B) shows main coevolving regions, each dot representing 1,000 consecutive nucleotides, with respect to two classic metrics (top–`GTest`, bottom–`RNAalifold` conservation score). For a given metrics and pair of region, the strength of the interaction (line thickness) indicates the number of column pairs which exceeding a predefined cutoff.

to query counts on multiple columns. For instance, for pair count queries, each column is queried on average $O(s)$ times.

Optimizing genome order is all the more important for pair count queries since the time complexity depends on the largest number of runs among the two columns. In practice, the optimized order is 4 times quicker to compute pair count queries than the initial order on the 1.9M genomes: the counts for the 607M pair of columns are computed in 5h and 81MB of RAM under the 5000-discordant subword order and in 22h and 197MB of RAM under the initial order.

Covariation analysis of SARS CoV-2 genomes

In order to demonstrate the usability of our index in the context of very large viral alignments, we analyzed a 65GB large MSA consisting of 1,870,492 SARS-CoV2 genomes, further described in the Datasets subsection. More specifically, we strived to identify sites and regions undergoing evolutionary pressure.

More specifically, we analyzed the columnwise entropy as a measure of the diversity of nucleotides observed within a given column. Pair statistics were also considered to capture a notion of coevolution, and we considered the `GTest` statistics used at the core of the popular `RScape` method [Rivas et al., 2017]. This score is closely related to Mutual Information, and does not specifically focus on the propensity of sites to form a base pair, rather rewarding an observed bias of evolution away from the uniform distribution. Towards structural analysis, it is sometimes beneficial to complement such metrics with scores that explicitly reward compatible nucleotide pairs, more likely to indicate compensatory mutations. We thus turn to a conservation score introduced by `RNAalifold` [Bernhart et al., 2008] as a pseudoenergy term to complement the classic Turner energy model with evolutionary information.

We compute the site-level entropy, for each of the columns in the alignment, and the two-sites `GTest` and `RNAalifold` scores for each pair of columns. As mentioned in the Runtime analysis Section, an optimized ordering of genomes allows performing the whole computation in little over 22h on a single CPU, an impressive feat given the large number of column pairs (606,547,035), each requiring an iteration over 1,870,492 rows in an uncompressed setting.

For the sake of readability, and given the length of the alignment, we present in Figure 6 a coarse-grained visualization of the results. Namely, the alignment is broken up into 1,000nts non-overlapping regions, and we report the number of sites and pairs having value greater than a metrics-specific cutoff (entropy > 0.1, `GTest` $>$ 250k, `RNAAlifold` > 0.75).

On a single-site level, our analysis shows that only a marginal fraction (1-30‰) of the positions feature some level of entropy (above 0.1). Such a perceived conservation likely reflects strong evolutionary constraints, coupled with a close temporal proximity of collection for the majority of genomes, in the initial stages of the COVID 19 outbreak. Interestingly, the largest values observed for the columnwise entropy are found in the 22k-26k region of the alignment, encoding the *spike* glycoprotein. Since this protein has been targeted by vaccines, higher mutation rates in this genomic region are expected and consistent with documented escape strategies (along with the position of variants of concern). Probably more surprising is the presence of a steady, relatively high, level of mutations in the terminal 25k-35k region, despite

the presence in this region of multiple, sometimes overlapping, open reading frames.

Our two chosen metrics are in general agreement in the region-level coevolutionary analyses, but also feature notable discrepancies. On a broad level, our analyses reveal the existence of a hub in the 24k-25k region of the MSA, associated with the ORF encoding the spike protein. According to both metrics, this region seemingly coevolves with 10 out of 23 regions in 5', and with all of the terminal regions (25k-35k). Both metrics also support a strong and pervasive coevolution with the 25k-35k region, both on an information-theoretic and structural level. As expected, the `RNAalifold` score appears more selective than the `GTest`, leaving some regions entirely devoid of coevolving partners, while highlighting a potential of the 4k-5k region to region with downstream regions which are not visible from the `GTest`. This is likely due to the *transitive* nature of scores akin to a Mutual Information (if both pairs (a, b) and (a, c) have large MI, then (b, c) typically has large MI). Conversely, the large number of interactions having good `RNAalifold` score between the 21k-22k and 24k-25k regions, not present in the `GTest` data, suggests the presence of compensatory mutations, possibly indicating evolutionarily-conserved structural elements.

While those promising preliminary results would certainly warrant further refinements (*e.g.* phylogenetic subsampling, comparison against available probing data [Manfredonia et al., 2020]), we must stress that the mere production of the metrics would not have been realistically feasible if not for the availability and efficiency of `CREMSA`, thus demonstrating its value and relevance.

Discussion

We have introduced a new compressed index for highly redundant multiple sequence alignments, as well as a new ordering of mutiple sequence alignment that benefits other compressors. This new order significantly improves compression. It is yet to be determined whether other simple orders could go beyond the compression ratios we achieved. It is surprising that an order as simple as the d-discriminative subword works so well. It is also surprising that the order of the letters within the subword has no significant impact on the compression ratios. This suggests that some information used for the ordering is redundant and that a more careful selection of the columns to be selected to determine the order could achieve similar results.

The wealth of data used for this kind of analysis also comes with its own limitations. Among millions of genomes, it is to be expected that some of them are of poorer qualities, despite the filters we have applied. Those genomes will tend to fragment the MSA by introducing gaps within it. Spurious gaps could prevent the identification of compensatory mutations, when those mutations end up being spread on different columns because of gaps. Among millions of genomes, it is usually difficult to identify the tiny fraction of them which is problematic. Using `CREMSA` one could easily identify the genomes which often introduce gaps in the alignment. Thus, `CREMSA` could help improve the public datasets of millions of genomes and it could eventually help to identify meaningful covariations.

`CREMSA` could easily be turned into a fully dynamic structure, allowing the insertion of new sequences to an existing index *on the fly*. However in such a case, the computation of the multiple sequence alignment would remain the bottleneck, as the MSA would require complete recomputation. As mentioned previously the computation of the MSA on 1.9M SARS-CoV-2 genomes required 1.2TB of RAM with Halign3. This makes such computation not readily accessible. Thus the availability of `CREMSA` motivates the development of more frugal approaches to multiple sequence alignment, tailored for highly similar sequences.

Conclusion

`CREMSA` is a frugal compressed index for highly redundant multiple sequence alignments achieving compression ratios close to the state-of-the-art specific compressors, but taking a fraction of the time they needed. Additionally, `CREMSA` provides random access to any nucleotide and greatly eases the computation of multi-columns/sites statistics in the MSA, in order to determine valuable covariation statistics.

We also introduced a new ordering on multiple sequence alignments that also benefits other compressors. Such an optimized ordering is purely beneficial for `CREMSA` (as well as other compressors), as they both reduce space and time consumption of our structure, and enable faster queries. Namely, in our running example focusing on SARS-CoV 2, our new order reduces the index size by a factor of three and the computation time for pair count queries by a factor of four.

We also compute an index on 1,870,492 SARS-CoV-2 genomes aligned with Halign3 (65GB of data) in an index compressed to 22MB on disk and taking 81MB of RAM. Then, in a few hours `CREMSA` could compute pair counts for any pair of columns, that could then be used to derive conservation scores.

Data availability and Reproducibility

The multiple sequence alignment of 1,870,492 SARS-CoV-2 genomes is available on Zenodo at `https://zenodo.org/records/14698859`. A Snakefile is available to reproduce the benchmarks on `CREMSA` and the other compressors at `https://gitlab.univ-lille.fr/cremsa/bench`.

Acknowledgment

This work was funded by the French *Agence Nationale de la Recherche* (ANR) through the INSSANE project (ANR21-CE45-0034). Additional computing resources where provided by the *Institut Français de Bioinformatique* (IFB) and the IFB Core cluster, under the ANR-funded *Programme d'Investissements d'Avenir* (RENABI-IFB; ANR-11-INBS-0013 and MUDIS4LS ANR-21-ESRE-0048).

We are also grateful to Karel Břinda for pointing us to the Hunt et al. dataset.

References

S. H. Bernhart, I. L. Hofacker, S. Will, A. R. Gruber, and P. F. Stadler. RNAalifold: improved consensus structure prediction for RNA alignments. *BMC Bioinformatics*, 9(1), Nov. 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-474.

K. Břinda, L. Lima, S. Pignotti, N. Quinones-Olvera, K. Salikhov, R. Chikhi, G. Kucherov, Z. Iqbal, and M. Baym. Efficient and robust search of microbial genomes via phylogenetic compression. *bioRxiv*, 2024. doi: 10.1101/2023.04.15.

536996. URL https://www.biorxiv.org/content/early/2024/05/11/2023.04.15.536996.

S. Deorowicz, J. Walczyszyn, and A. Debudaj-Grabysz. CoMSA: compression of protein multiple sequence alignment files. *Bioinformatics*, 35(2):227–234, 07 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty619. URL https://doi.org/10.1093/bioinformatics/bty619.

R. R. Gutell, N. Larsen, and C. R. Woese. Lessons from an evolving rRNA: 16S and 23S rRNA structures from a comparative perspective. *Microbiol. Rev.*, Mar. 1994. doi: 10.1128/mr.58.1.10-26.1994.

M. Hunt, A. S. Hinrichs, D. Anderson, L. Karim, B. L. Dearlove, J. Knaggs, B. Constantinides, P. W. Fowler, G. Rodger, T. Street, S. Lumley, H. Webster, T. Sanderson, C. Ruis, B. Kotzen, N. de Maio, L. N. Amenga-Etego, D. S. Y. Amuzu, M. Avaro, G. A. Awandare, R. Ayivor-Djanie, T. Barkham, M. Bashton, E. M. Batty, Y. Bediako, D. D. Belder, E. Benedetti, A. Bergthaler, S. A. Boers, J. Campos, R. A. A. Carr, Y. Y. C. Chen, F. Cuba, M. E. Dattero, W. Dejnirattisai, A. Dilthey, K. O. Duedu, L. Endler, I. Engelmann, N. M. Francisco, J. Fuchs, E. Z. Gnimpieba, S. Groc, J. Gyamfi, D. Heemskerk, T. Houwaart, N.-y. Hsiao, M. Huska, M. Hölzer, A. Iranzadeh, H. Jarva, C. Jeewandara, B. Jolly, R. Joseph, R. Kant, K. K. K. Ki, S. Kurkela, M. Lappalainen, M. Lataretu, J. Lemieux, C. Liu, G. N. Malavige, T. Mashe, J. Mongkolsapaya, B. Montes, J. A. M. Mora, C. M. Morang'a, B. Mvula, N. Nagarajan, A. Nelson, J. M. Ngoi, J. P. da Paixão, M. Panning, T. Poklepovich, P. K. Quashie, D. Ranasinghe, M. Russo, J. E. San, N. D. Sanderson, V. Scaria, G. Screaton, O. M. Sessions, T. Sironen, A. Sisay, D. Smith, T. Smura, P. Supasa, C. Suphavilai, J. Swann, H. Tegally, B. Tegomoh, O. Vapalahti, A. Walker, R. J. Wilkinson, C. Williamson, X. Zair, I. L. N. Consortium, T. de Oliveira, T. E. Peto, D. Crook, R. Corbett-Detig, and Z. Iqbal. Addressing pandemic-wide systematic errors in the sars-cov-2 phylogeny. *bioRxiv*, 2024. doi: 10.1101/2024.04.29.591666.

I. Manfredonia, C. Nithin, A. Ponce-Salvatierra, P. Ghosh, T. K. Wirecki, T. Marinus, N. S. Ogando, E. Snijder, M. J. van Hemert, J. M. Bujnicki, and D. Incarnato. Genome-wide mapping of sars-cov-2 rna structures identifies therapeutically-relevant elements. *Nucleic Acids Research*, 48(22):12436–12452, 11 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa1053. URL https://doi.org/10.1093/nar/gkaa1053.

G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys (CSUR)*, 39(1):2–es, 2007.

N. Prezza. A framework of dynamic data structures for string processing. *arXiv*, 2017. doi: 10.48550/arXiv.1701.07238. URL https://arxiv.org/abs/1701.07238.

E. Rivas, J. Clements, and S. R. Eddy. A statistical test for conserved RNA structure shows lack of evidence for structure in lncRNAs. *Nat. Methods*, 14(1):45–48, Jan. 2017. ISSN 1548-7105. doi: 10.1038/nmeth.4066.

F. Tang, J. Chao, Y. Wei, F. Yang, Y. Zhai, L. Xu, and Q. Zou. Halign 3: fast multiple alignment of ultra-large numbers of similar dna/rna sequences. *Molecular Biology and Evolution*, 39(8):msac166, 2022.

S. Triebel, K. Lamkiewicz, N. Ontiveros, B. Sweeney, P. F. Stadler, A. I. Petrov, M. Niepmann, and M. Marz. Comprehensive survey of conserved rna secondary structures in full-genome alignment of hepatitis c virus. *Scientific Reports*, 14(1), July 2024. ISSN 2045-2322. doi: 10.1038/s41598-024-62897-0.

B. Woolf. THE LOG LIKELIHOOD RATIO TEST (THE G-TEST). *Ann. Hum. Genet.*, 21(4):397–409, May 1957. ISSN 0003-4800. doi: 10.1111/j.1469-1809.1972.tb00293.x.