# Report from Docker Container Reproducing Experiments from paper: Universal Indexes for Highly Repetitive Document Collections☆, by:

Antonio Fariña[*,b], Miguel A. Martínez-Prieto[c], Francisco Claude[a], Gonzalo Navarro[d]

[a] *Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Chile.*
[b] *Database Laboratory, University of A Coruña, Spain.*
[c] *DataWeb Research, Department of Computer Science, University of Valladolid, Spain.*
[d] *Department of Computer Science, University of Chile, Chile.*

**Abstract**

We include in this report the main figures reproduced for paper [1] from the experimental data and source codes obtained by running our build, search, and report scripts from our *parent* paper, and described in detail in our reproducibility paper: "*On the Reproducibility of Experiments of Indexing Repetitive DocumentCollections*".

**Keywords:** *experiments, reproducibility.*

## 1. Experimental Framework and Results

We experimentally study the space/time tradeoffs obtained with the described inverted list representations, in both the non-positional and positional scenarios. In the positional scenario we also add a comparison with the self-indexes proposed.

### *1.1. Test Data*

Within `uiHRDC` we provide in `data` directory, both the document collections and the query sets used in the experimental setup of our parent paper. They are described below.

#### *1.1.1. Document Collections*

Our document collections were created from the 108.5 GB Wikipedia collection described by He et al. [3], which contained 10% of the complete English Wikipedia from 2001 to mid 2008. It contains 240,179 articles, and each of them has a number of versions. Its statistics are shown in Table 1. Note that we had not the

☆

*Corresponding author
*Email addresses:* `fari@udc.es` (Antonio Fariña), `migumar2@infor.uva.es` (Miguel A. Martínez-Prieto), `fclaude@recoded.cl` (Francisco Claude), `gnavarro@dcc.uchile.cl` (Gonzalo Navarro)

original 108.5 GB collection, but a filtered (tag-free) version of it whose size totaled 85.58 GB. Therefore, the original collection was 1.27 times larger than ours.

| Subset | Size (GB) | Articles | Number of versions | Versions / article | Filename (within `uiHRDC`) | Filesize (GB) |
|---|---|---|---|---|---|---|
| Full | 108.50 | 240,179 | 8,467,927 | 35.26 | -- | 85.58 |
| Non-pos | 24.77 | 2,203 | 881,802 | 400.27 | text20gb.txt | 19.53 |
| Pos | 1.94 | 4,327 | 149,761 | 34.61 | wiki_src2gb.txt | 1.94 |

Table 1: Detailed statistics of the document collections used.

From the `Full` collection of articles, we chose two subsets of the articles, and collected all the versions of the chosen articles. For the non-positional setting our subset (`Non-pos`) contains a prefix of 19.53 GB of the full collection, whereas for positional indexes we chose 1.94 GB of random articles. However, for a fair comparison with the techniques from [3], in the non-positional scenario we scaled the size of the `Non-pos` subset using the above 1.27 factor when referring to its space requirements. Additional statistics of our two subsets are also included in Table 1.

### *1.1.2. Query sets*

Since the experiments target at providing space/time for the different indexing alternatives when performing `locate(pattern)` and `extract(interval)` queries we provided two types of query sets for each document subcollection.

- Query sets for `locate`: We provide four query sets, each of them containing 1,000 queries, which include: *i)* two query sets composed of one-word patterns chosen at random from the vocabulary of the indexed subcollection. In the first case ($W_a$), it includes low-frequency words occurring less than 1,000 times. In the second case, the query set ($W_b$) includes high-frequency words occurring more than 1,000 times; *and ii)* two query sets with 1,000 phrases composed of 2 and 5 words that were chosen randomly from the text of the subcollection (with no restrictions on its frequency).

- Interval sets for `extract`: Aiming at measuring extraction time when recovering snippets of length 80 (around one line) and 13,000 (around one document, in our collection) characters, we generated: *i)* a set of 10,000 intervals of width 13,000 characters from the `POS` text collection, and *ii)* a set containing 100,000 intervals of width 80 characters. Since these intervals are not suitable for our word-based self-indexes (`WCSA` and `WSLP`), and assuming that the average word length is around 4 in our datasets, we also generated two additional sets composed respectively of 10,000 intervals containing $3,000$ words each, and and 100,000 intervals containing 20 words each.

## 2. Techniques used: Inverted indexes and self-indexes

The techniques included in this Report are summarized in Table 2. Recall that we include, compressed inverted indexes for the non-positional scenario, whereas for the positional scenario we include the most promising compressed inverted indexes and also self-indexes.

Table 2: Techniques included in this report.

| NON POSITIONAL INV. INDEXES | POSITIONAL INV. INDEXES | SELF-INDEXES |
|---|---|---|
| RICE | RICE | WCSA |
| RICE-B | | RLCSA |
| VBYTE | VBYTE | SLP |
| VBYTE-CM | VBYTE-CM | WSLP |
| VBYTE-ST | VBYTE-ST | LZ77-Index |
| VBYTE-B | | LZEnd-Index |
| VBYTE-CM-B | | |
| VBYTE-ST-B | | |
| SIMPLE-9 | SIMPLE-9e | |
| PFORDELTA | | |
| QMX-SIMD | QMX-SIMD | |
| *ELIAS-FANO-OPT | *ELIAS-FANO-OPT | |
| *OPT-PFD | *OPT-PFD | |
| *INTERPOLATIVE | *INTERPOLATIVE | |
| *VARINT-G8IU | *VARINT-G8IU | |
| RICE-RLE | | |
| VBYTE-LZMA | VBYTE-LZMA | |
| VBYTE-LZEND | | |
| REPAIR | REPAIR | |
| REPAIR-SKIPPING | REPAIR-SKIPPING | |
| REPAIR-SKIPPING-CM | REPAIR-SKIPPING-CM | |
| REPAIR-SKIPPING-ST | REPAIR-SKIPPING-ST | |
| *(text not included, only intersections)* | *(text compressed with Re-pair)* | |

### *2.1. Results for Non-positional inverted indexes*

Our experiments compare the space/time tradeoffs of several variants of non-positional inverted indexes over the highly repetitive 24.77 GB subcollection described above. We include in this comparison some of the best classical encodings to represent d-gaps, such as `Rice`, `Simple9`, `PforDelta`, and `Vbyte` with no sampling to speed up intersections (thus only merge-wise intersections are feasible). We also include two alternatives using `Vbyte` coupled with list sampling [2] (`Vbyte-CM`) with $k = \{4, 32\}$, or domain sampling [9] (`Vbyte-ST`) with $B = \{16, 128\}$. In addition, we include the hybrid variant of `Vbyte-CM` that uses bitmaps to represent the largest inverted lists (`Vbyte-CMB`) [2]. For completeness we used the same approach on `Vbyte-ST`, to build `Vbyte-STB`, and also included variants `VbyteB` and `RiceB` with no sampling. We also tested the novel

QMX[1] technique [10] that uses SIMD-instructions to boost decoding of large lists, and coupled it with an intersection algorithm [5] that also benefits from SIMD-instructions.[2]

We also tested the recent Partitioned Elias-Fano indexes [7], and used the best/optimized variant from that paper (EF-opt). The source code is available at authors' website.[3] From the same authors [7], we also included in our experiments the variants OPT-PFD (optimized PForDelta variant [11]), Interpolative (Binary Interpolative Coding [6]), and varintG8IU (SIMD-optimized Variable Byte code [8]). Since we used the implementations from [7] and we adapted our query patterns to match their needs, or even measured partial times (parsing and map-to-documents) separately, we marked with an '*' these techniques in the figures.

We compare all those techniques in Section 2.1.1, to determine which are the most successful among the traditional techniques in the repetitive scenario. Then, in Section 2.1.2 we compare them with the new variants designed to deal with repetitive data we proposed. In particular, we include Rice-Runs, Vbyte-LZMA, Vbyte-Lzend, RePair, RePair-Skip. We add no sampling to them. Therefore, only RePair-Skip can benefit of additional data to boost the intersections (which are performed sequentially). In addition, we show the Re-Pair variants using sampling, RePair-Skip-CM (with $k = \{1, 64\}$) and RePair-Skip-ST (with $B = \{1024\}$). For Vbyte-Lzend we will obtain different space/time tradeoffs by tuning its *delta-codes-sampling* parameter $ds$ (see [4] for details) to $ds = \{4, 16, 64, 256\}$.

Finally, note that the space results reported for the indexes are shown as a percentage of the index size with respect to the size of the original [sub]collection in plain text ($index\_size/original\_size \times 100$). Note that we are not considering in this experiment the compressed representation of the text. Times are shown in microseconds per occurrence.

### 2.1.1. *Using traditional techniques in a repetitive scenario*

We include a comparison of well-known techniques that were initially developed for non-repetitive scenarios, now operating on repetitive collections. Figure 1 shows the space/time tradeoffs for non-positional indexes using those techniques to represent posting lists.

### 2.1.2. *Comparison with our proposals*

Our next experiments compare our proposals with the best counterparts from the previous section. Figure 2 shows the space/time tradeoffs for all the resulting non-positional indexes.

[1] http://www.cs.otago.ac.nz/homepages/andrew/papers/QMX.zip.
[2] https://github.com/lemire/SIMDCompressionAndIntersection.
[3] https://github.com/ot/partitioned_elias_fano.

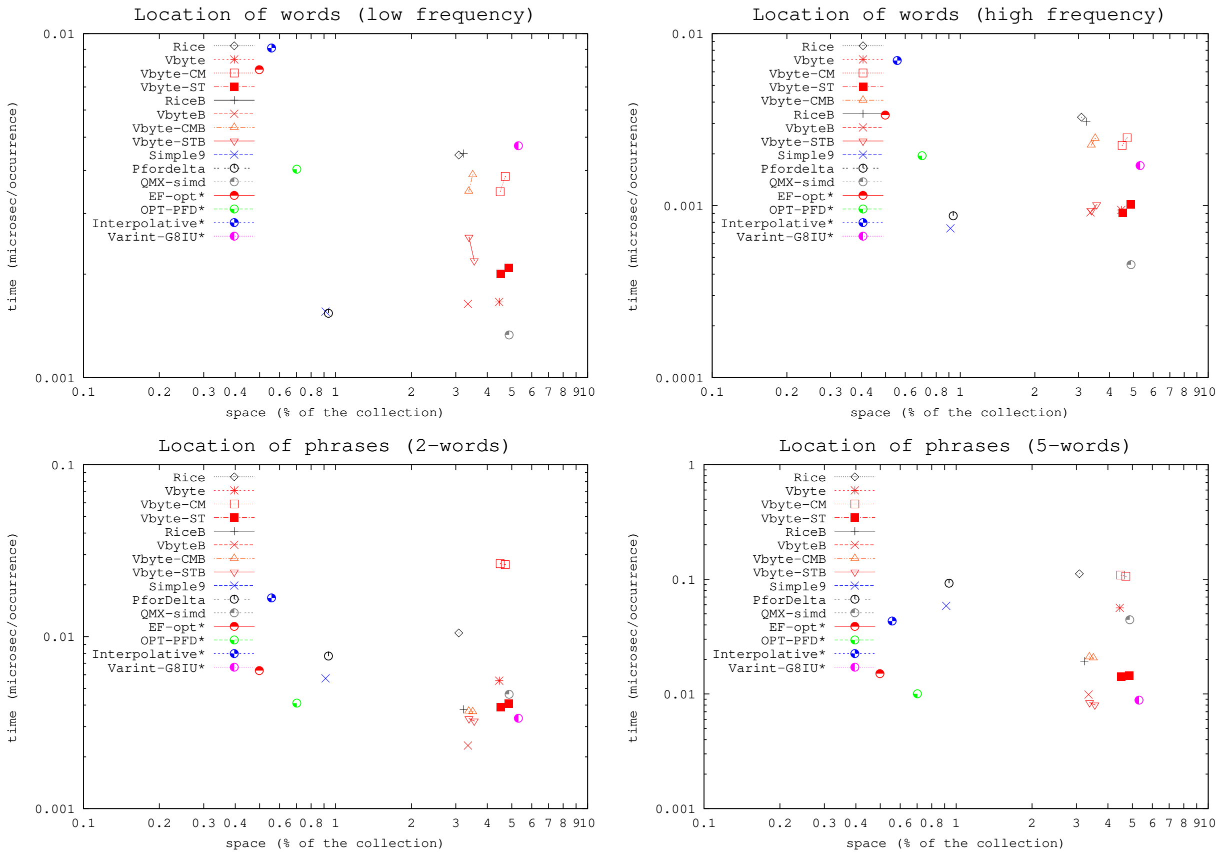


Figure 1: Space/time tradeoffs for non-positional indexes using traditional techniques. Logscale. This figure corresponds to Figure 3 in the parent paper [1].

### *2.2. Results for Positional indexes*

For testing the positional indexes we used the 1.94 GB subcollection because several self-index implementations are unable to handle texts larger than $2^{31}$ bytes.

Since self-indexes must reproduce the precise text, we cannot apply case folding nor any kind of filtering in this scenario. We index the original text as is. As explained, word-based self-indexes will regard (and index) the text as a sequence of words and separators. For fairness, the positional inverted indexes will index separators as valid words as well, and phrase queries will choose sequences of tokens (either words or separators). Still, we note that character-based self-indexes will return more occurrences than word-based self-indexes (or than inverted indexes), as they also report the non-word-aligned occurrences. Times per occurrence still seem comparable, yet they slightly favor character-based self-indexes since the time per occurrence drops as more occurrences are reported (there is a fixed time cost per query).

We consider most of the techniques of the non-positional setting, now operating on position lists. Yet, for `Rice` and `Vbyte` we do not include the hybrid variants using bitmaps as they obtain no space/time

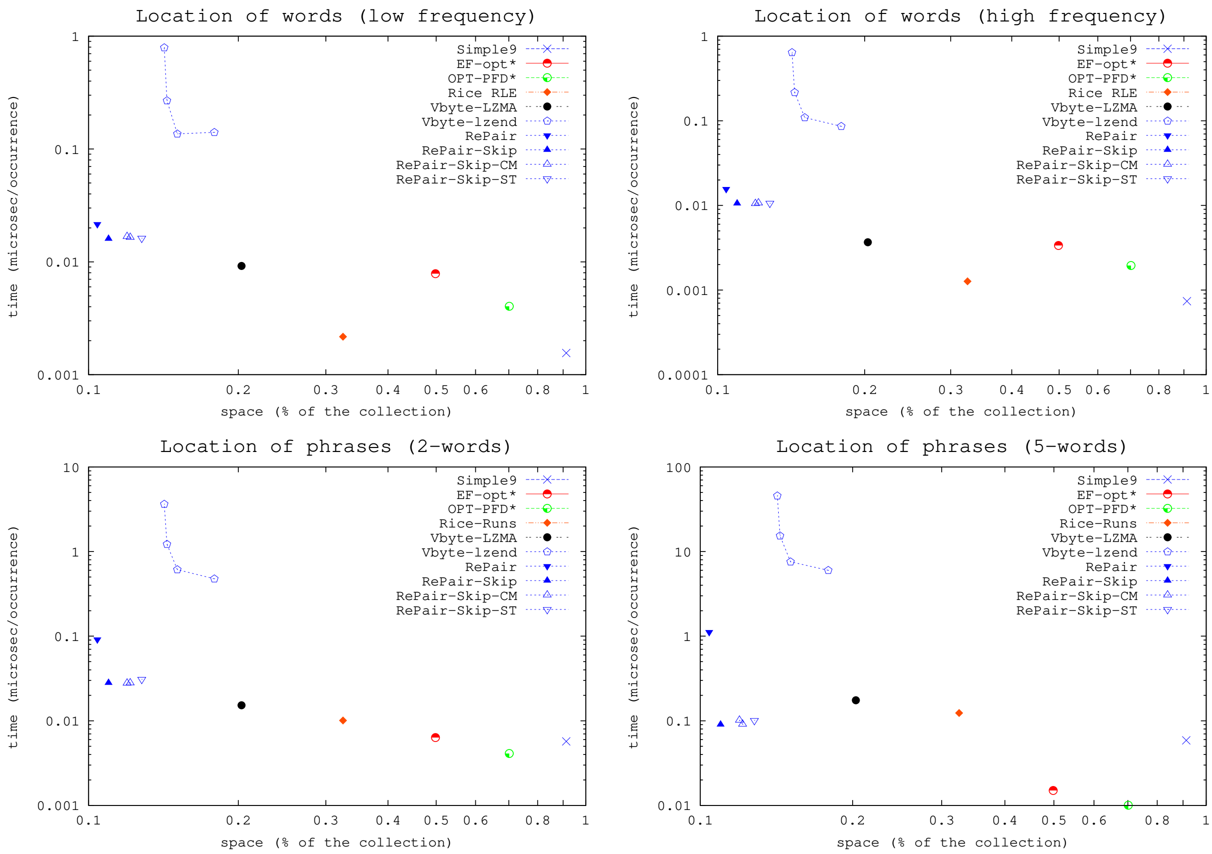


Figure 2: Space/time tradeoffs for non-positional indexes, comparing the best classical techniques with our new ones. Logscale. This figure corresponds to Figure 4 in the parent paper [1].

improvements in the positional scenario. For the `Vbyte` counterparts using sampling, we set the same sampling parameters as in the previous section: `Vbyte-CM` with $k = \{4, 32\}$ and `Vbyte-ST` with $B = \{16, 128\}$. We had to adapt `Simple9` because it is unable to represent gaps longer than $2^{28}$. While such gaps do not arise on document lists, they do occur in position lists. We use the gap $2^{28} - 1$ as an escape code and then the next 32 bits represent the real gap. We exclude `PforDelta` because it has the same limitation, fixing it is more cumbersome, and its performance is not very different from that of `Simple9`. We also exclude `Rice-Runs`, as runs do not arise in the positional setting.

We did not include `Vbyte-Lzend`, as it was clearly overcome by `Vbyte-LZMA` and our Re-Pair variants. For the variants of Re-Pair using sampling, we set the sampling parameters to $k = \{1, 64\}$ for `RePair-Skip-CM` and $B = \{256\}$ for `RePair-Skip-ST`.

To compete in similar conditions with self-indexes, positional inverted indexes must be enhanced with an efficient extraction/decompression mechanism that allows any portion of the source text to be efficiently reproduced; i.e. we need a representation of the source text. We choose Re-Pair for this purpose because it is well-suited for highly repetitive collections and supports fast direct access to the text. Because the text in this way represents a very small fraction of the total space, we represent the rules as pairs of integers to speed up text extraction, instead of the slower $R_B$ and $R_S$ based implementation. This adds up to 1.21% of the original text size. To further improve extraction performance, we add a regular sampling of the array $C$, which increases space up to 1.3% for the densest sampling. As a comparison, `p7zip` (from `www.7-zip.org`), the best compressor for this type for repetitive texts, achieved 0.52% space on this subcollection (albeit not providing direct access).

We add to both the inverted indexes and self-indexes the time and space required for converting absolute positions to (document,offset) pairs (see `merge-occs-to-docs` procedure in the reproducibility companion paper). The extra space added by the corresponding mapping structure is just 0.03%.

In Section 2.2.1 we compare positional inverted indexes using state-of-the-art representations for posting lists. Then, in Section 2.2.2 we compare the best state-of-the art representations and our new representations of positional inverted lists, plus the tuned self-indexing alternatives. Our final experiments, in Section 2.2.3, study the performance when we extract snippets from the original document collection.

### *2.2.1. Traditional inverted list representations*

Figure 3 shows the space/time tradeoffs achieved, for the four types of queries, with traditional inverted index representations. All classical inverted indexes achieve similar space, ranging from 30% to 40% of the text size. From those, `OPT-PFD`[4] obtains the best compression, with a slight gap over `EF-opt` and `Interpolative`.

### *2.2.2. Comparing positional inverted indexes with self-indexes*

We compare the best traditional inverted indexes with the variants we developed to exploit repetitiveness. In addition, we include the self-indexes (tuned as shown in the companion reproducibility paper) in the comparison. Figure 4 shows the results.

`RePair` and `RePair-Skip` achieve almost the same space, close to 20%, and the latter is always faster for the same reasons as in non-positional indexes. While for words, `RePair-Skip` is slower than the classical methods, its times become similar to those of `Simple9` on phrases. Adding sampling on top of `RePair-Skip` clearly outperforms `Simple9` on phrases. Yet, `RePair-Skip-ST` and `RePair-Skip-CM` are still clearly slower than the `EF-opt`, `OPT-PFD`, and `varintG8IU`, which obtain the best performance at phrase queries.

The best space of inverted indexes is achieved by `Vbyte-LZMA`, which reaches a compression ratio near 10% (half the space of `RePair-Skip` variants). This represents a significant improvement upon the state of

[4]Recall that in the companion reproducibility paper (Section 2.2.1) we indicated that there was a fix here, because in the parent paper, the space/time values for `OPT-PFD` were wrong.

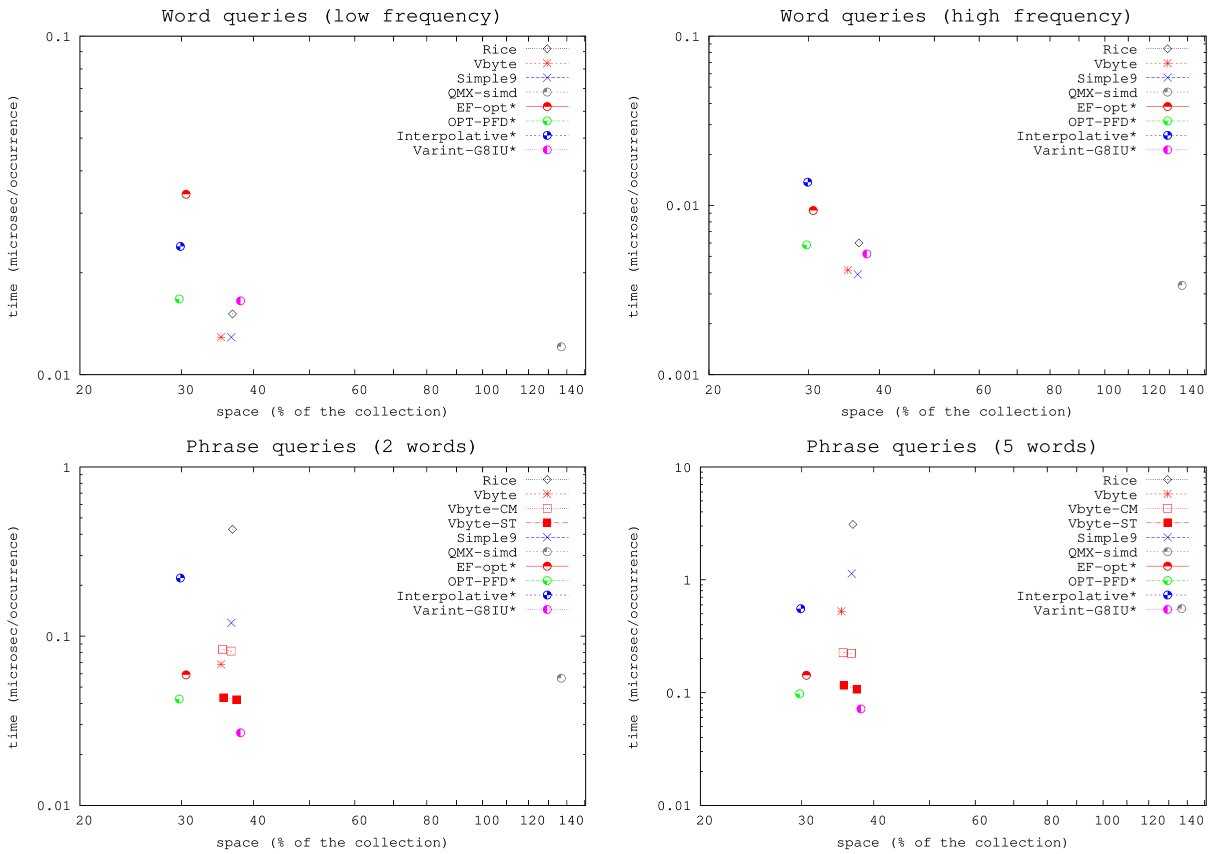


Figure 3: Space/time tradeoffs for traditional representations of positional indexes. Logscale. This figure corresponds to Figure 6 in the parent paper [1].

the art. Moreover, for single-word queries its times are only slightly worse than those of `RePair-Skip`, yet on phrase queries its need to fully decompress the list makes it clearly slower (among the inverted indexes, only `RePair` performs worse than `Vbyte-LZMA` in this scenario).

Self-indexes are able to use much less space. First, note that `WSLP` is only slightly smaller than `SLP`. This shows that grammar-based compressors do not gain much from handling words instead of characters. They achieve around 3% compression ratio. This important reduction in space compared to the 10% of `Vbyte-LZMA` is paid with a sharp increase in search times. On words, they are up to two orders of magnitude slower than `Vbyte-LZMA`. However, this gap decreases as longer patterns are used. Actually, they could obtain comparable times to `Vbyte-LZMA` on 5-word queries. This is because, these self-indexes are mostly insensitive to the number of words in the query, whereas inverted indexes become much slower when looking for long phrases. Thus, for long queries, `SLP` and `WSLP` are very attractive alternatives.

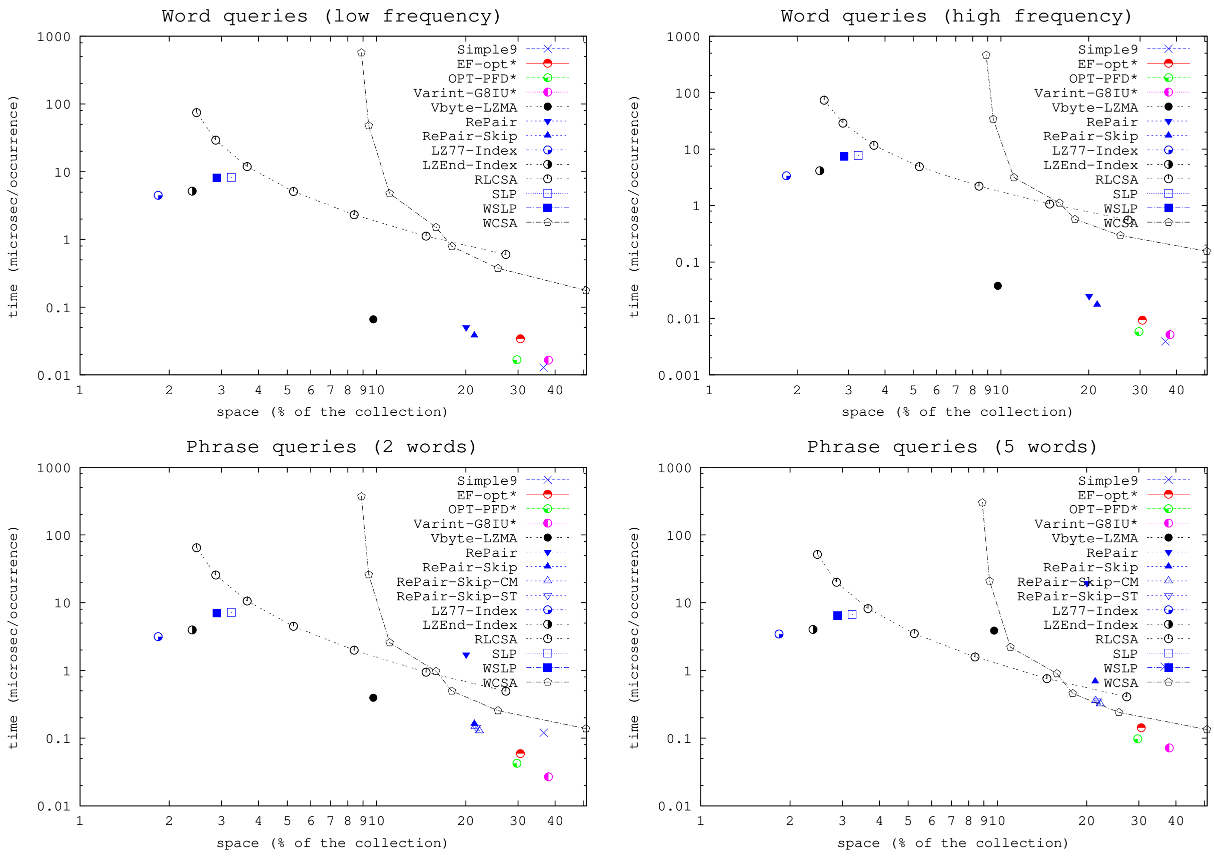


Figure 4: Space/time tradeoffs for positional indexes. Logscale. This figure corresponds to Figure 9 in the parent paper [1].

### 2.2.3. *Text extraction*

Since self-indexes represent the text as a part of the index, it is relevant to measure how fast they are at extracting an arbitrary text snippet. For fairness we have added to our inverted indexes a Re-Pair-compressed version of the text. In order to support snippet extraction, we added a regular sampling over the final Re-pair sequence ($C$), which indicates the text position where the corresponding symbol starts. For decompressing an arbitrary snippet we binary search the rightmost preceding sample and decompress from there on. This induces a space/time tradeoff regarding the sampling step.

Figure 5 shows the results obtained when we extract random snippets of length 80 and 13,000 characters. Word-based indexes `WCSA` and `WSLP` extract a number of words equal to 80 or 13,000 divided by the average word length, to provide a roughly comparable result.

## 3. Conclusions

We have left our codes and experimental testbeds available at `https://github.com/migumar2/uiHRDC`. Please refer to our Reproducibility paper "*On the Reproducibility of Experiments of Indexing Repetitive*

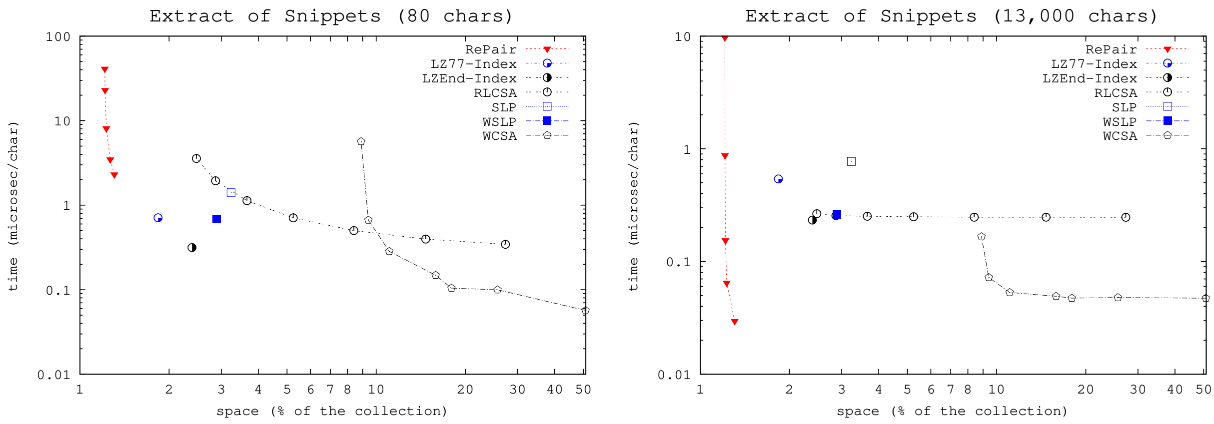


Figure 5: Space/time tradeoffs for extraction. Logscale. This figure corresponds to Figure 10 in the parent paper [1].

*DocumentCollections"* or to our parent paper [1] for more details.

## References


[1] F. Claude, A. Fariña, M. Martínez-Prieto, and G. Navarro. Universal indexes for highly repetitive document collections. *Information Systems*, 61:1–23, 2016.

[2] J. S. Culpepper and A. Moffat. Efficient set intersection for inverted indexing. *ACM Transactions on Information Systems*, 29(1):article 1, 2010.

[3] J. He, J. Zeng, and T. Suel. Improved index compression techniques for versioned document collections. In *Proc. 19th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1239–1248, 2010.

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

[5] D. Lemire, L. Boytsov, and N. Kurz. Simd compression and the intersection of sorted integers. *Software: Practice and Experience (published online)*, 2015.

[6] A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, 2000.

[7] G. Ottaviano and R. Venturini. Partitioned elias-fano indexes. In *Proc. 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 273–282, 2014.

[8] A. A. Stepanov, A. R. Gangolli, D. E. Rose, R. J. Ernst, and P. S. Oberoi. SIMD-based decoding of posting lists. In *Proc. 20th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 317–326, 2011.

[9] F. Transier and P. Sanders. Engineering basic algorithms of an in-memory text search engine. *ACM Transactions on Information Systems*, 29:article 2, 2010.

[10] A. Trotman. Compression, simd, and postings lists. In *Proc. 19th Australasian Document Computing Symposium (ADCS)*, pages 50–57. ACM, 2014.

[11] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *Proc. 18th International Conference on World Wide Web (WWW)*, pages 401–410, 2009.

## 4. Appendix 1: Details on the Computer used and summary of Results

The experiments were reproduced by the uiHRDC framework within a Docker instance in a computer which the following specifications:

- **CPU:** Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz (siblings[5]: 12)
- **RAM:** 65930372 kB (Swap: 59170208 kB)
- **OS:** Ubuntu 14.04.6 LTS (trusty). In particular: Linux version 4.4.0-142-generic (buildd@lgw01-amd64-033) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.10) ) #168-Ubuntu SMP Wed Jan 16 21:00:45 UTC 2019.

The experiments started on *2018-12-31 00:26:28.294789* and ended on *2019-01-01 16:09:34.907268*. The overall time to run the experiments was: **1 days, 15h43m06.61s**.

Below, we include a summary of the experiments performed for: *(a)* the non-positional inverted indexes (Table 3), *(b)* the positional inverted indexes (Table 4), and *(c)* the self-indexes (Table 5). In all those tables we show respectively for each technique: the name of the technique, the building time (considering all the variants built depending on the sampling parameter, if any) and the time spent to perform all the queries. Then, we include either a '✓' or a ' -- ' symbol to report wether the querying experiments either succeeded or failed. We consider the four scenarios for `locate` discussed in Section 1.1.2 (low frequency words, high frequency words, 2-words phrases, and 5-words phrases). In addition, in Tables 4 and 5 we also consider `extract` experiments considering snippets of both 80 and 13,000 chars. Finally, the last row of each table includes the overall time required to perform all the experiments that are summarized in each table.

[5]Only `Vbyte-LZMA` (at building time) takes advantage of using the available cores.

| | Overall Time | | Locate | | | |
|---|---|---|---|---|---|---|
| | | | Words | | Phrases | |
| | Building | Querying | Low freq | High freq | 2-words | 5-words |
| RICE | 00h06m12.16s | 00h05m16.62s | ✓ | ✓ | ✓ | ✓ |
| RICE-B | 00h05m20.43s | 00h01m39.72s | ✓ | ✓ | ✓ | ✓ |
| VBYTE | 00h05m07.31s | 00h02m45.73s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-CM | 00h10m39.11s | 00h15m17.61s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-ST | 00h10m21.40s | 00h02m44.35s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-B | 00h05m08.80s | 00h00m57.81s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-CM-B | 00h10m38.54s | 00h04m06.09s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-ST-B | 00h10m15.81s | 00h02m08.97s | ✓ | ✓ | ✓ | ✓ |
| SIMPLE-9 | 00h05m07.13s | 00h02m52.63s | ✓ | ✓ | ✓ | ✓ |
| PFORDELTA | 00h07m09.34s | 00h04m07.79s | ✓ | ✓ | ✓ | ✓ |
| QMX-SIMD | 00h05m11.40s | 00h06m31.33s | ✓ | ✓ | ✓ | ✓ |
| *ELIAS-FANO-OPT | 00h02m19.51s | 00h00m25.12s | ✓ | ✓ | ✓ | ✓ |
| *OPT-PFD | 00h00m42.14s | 00h00m16.81s | ✓ | ✓ | ✓ | ✓ |
| *INTERPOLATIVE | 00h00m51.56s | 00h01m08.28s | ✓ | ✓ | ✓ | ✓ |
| *VARINT-G8IU | 00h00m15.78s | 00h00m19.00s | ✓ | ✓ | ✓ | ✓ |
| RICE-RLE | 00h05m11.30s | 00h05m32.24s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-LZMA | 03h08m53.71s | 00h03m57.80s | ✓ | ✓ | ✓ | ✓ |
| VBYTE-LZEND | 03h47m46.22s | 01h12m03.54s | ✓ | ✓ | ✓ | ✓ |
| REPAIR | 00h08m12.84s | 00h25m06.67s | ✓ | ✓ | ✓ | ✓ |
| REPAIR-SKIPPING | 00h08m05.95s | 00h04m47.82s | ✓ | ✓ | ✓ | ✓ |
| REPAIR-SKIPPING-CM | 00h08m13.74s | 00h10m50.35s | ✓ | ✓ | ✓ | ✓ |
| REPAIR-SKIPPING-ST | 00h08m13.74s | 00h05m57.46s | ✓ | ✓ | ✓ | ✓ |
| **OVERALL TIME** | 12h22m19.43s | | | | | |

Table 3: Summary and state of the experiments run on the test machine: non-positional inverted indexes.

| | Overall Time | | Locate | | | | Extract | |
|---|---|---|---|---|---|---|---|---|
| | | | Words | | Phrases | | 80 | 13,000 |
| | Building | Querying | Low freq | High freq | 2-words | 5-words | chars | chars |
| RICE | 00h12m16.61s | 00h01m31.33s | ✓ | ✓ | ✓ | ✓ | | |
| VBYTE | 01h38m27.30s | 00h34m28.70s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| VBYTE-CM | 00h24m31.32s | 00h01m01.65s | ✓ | ✓ | ✓ | ✓ | | |
| VBYTE-ST | 00h24m11.16s | 00h00m56.62s | ✓ | ✓ | ✓ | ✓ | | |
| SIMPLE-9e | 00h12m15.59s | 00h01m15.65s | ✓ | ✓ | ✓ | ✓ | | |
| QMX-SIMD | 00h12m15.46s | 00h00m23.24s | ✓ | ✓ | ✓ | ✓ | | |
| *ELIAS-FANO-OPT | 00h01m00.81s | 00h00m14.59s | ✓ | ✓ | ✓ | ✓ | | |
| *OPT-PFD | 00h00m57.64s | 00h00m09.75s | ✓ | ✓ | ✓ | ✓ | | |
| *INTERPOLATIVE | 00h00m43.37s | 00h00m49.18s | ✓ | ✓ | ✓ | ✓ | | |
| *VARINT-G8IU | 00h00m25.22s | 00h00m07.73s | ✓ | ✓ | ✓ | ✓ | | |
| VBYTE-LZMA | 01h19m46.15s | 00h04m20.25s | ✓ | ✓ | ✓ | ✓ | | |
| REPAIR | 00h15m03.25s | 00h19m54.57s | ✓ | ✓ | ✓ | ✓ | | |
| REPAIR-SKIPPING | 00h15m17.66s | 00h01m15.13s | ✓ | ✓ | ✓ | ✓ | | |
| REPAIR-SKIPPING-CM | 00h16m54.98s | 00h10m00.58s | ✓ | ✓ | ✓ | ✓ | | |
| REPAIR-SKIPPING-ST | 00h17m00.10s | 00h05m36.71s | ✓ | ✓ | ✓ | ✓ | | |
| **OVERALL TIME** | 07h18m25.66s | | | | | | | |

Table 4: Summary and state of the experiments run on the test machine: positional inverted indexes. Note that Repair extraction times are identical for all the indexes (actually they do not depend on the type of index). Therefore, we only measured them once for `VBYTE` index.

| | Overall Time | | Locate | | | | Extract | |
|---|---|---|---|---|---|---|---|---|
| | | | Words | | Phrases | | 80 | 13,000 |
| | Building | Querying | Low freq | High freq | 2-words | 5-words | chars | chars |
| WCSA | 00h23m57.25s | 06h33m10.42s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| RLCSA | 00h19m01.55s | 05h39m49.84s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SLP | 01h24m57.17s | 01h44m45.96s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| WSLP | 00h14m36.89s | 01h02m16.33s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| LZ77-Index | 00h04m48.21s | 00h54m09.18s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| LZEnd-Index | 00h44m44.44s | 00h55m37.56s | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **OVERALL TIME** | 20h02m21.41s | | | | | | | |

Table 5: Summary and state of the experiments run on the test machine: self-indexes.