Practical and Flexible Indexes on Repetitive String Collections

DUSTIN COBAS BATISTA, University of Chile, Chile

Nowadays, we face an overwhelming problem: to store and access the massive amount of data generated every day. Fortunately, many of the fastest-growing string collections are composed of very similar documents, such as versioned code and document collections, genome repositories, etc. A lot of interest in this type of collections has lately arisen, allowing the emergence of many pattern-matching indexes that exploit the string repetitions in different ways: since the use of the efficient and well-known Lempel-Ziv parser to the novel and promising string attractor proposal. Despite the amount of previous work in this area, recent researches have triggered several lines of work, from which some questions remain open. On the other hand, document retrieval techniques are less developed on generic and repetitive string collections. For both fields, there are very few practical indexes implemented. Our main goal is to develop practical and flexible succinct indexes to support pattern matching and document retrieval operations on repetitive string collections. Our indexes must efficiently handle different repetitiveness scenarios.

Additional Key Words and Phrases: Pattern matching, document retrieval, repetitive collections

1 INTRODUCTION

The large amount of digital information that is generated by different human activities in a daily basis, needs to be efficiently stored and accessed. It constitutes a generalized problem in most organizations aimed at data processing. Handling of *string data* is a well-explored area. It is formed by collections of symbol sequences — such as natural language text collections, DNA and protein sequences, source code repositories, etc.

The sharp growth of text collections is a concern in many recent applications. This phenomenon causes Moore's Law to be outperformed in some cases [30]. Fortunately, many of the fastest-growing text collections are *highly repetitive*: each document can be obtained from a few large blocks of other documents. These collections arise in different areas, such as repositories of genomes of the same species (which differ from each other by a small percentage only) like the 100K-genome project[1]; software repositories that store all the versions of the code arranged in a tree or acyclic graph like GitHub[2]; versioned document repositories where each document has a timeline of versions, like Wikipedia[3], etc. On such text collections, statistical compression is ineffective [20] and even $O(n)$ bits of extra space can be unaffordable.

Repetitiveness is the key to tackle the fast growth of these collections: their amount of *unique* material grows much slower than their size. For example, version control systems provide efficient storage and access to the documents of a versioned collection. For each version, they store the list of edits regarding some reference document that is stored in plain form, and reconstruct it by applying the edits to the reference version.

However, it is much more challenging to *index* those collections in a small space so as to support more advanced functionalities such as fast *pattern matching* or *document retrieval* tasks. Given a *collection of string documents* $\mathcal{D} = \{T_1, \ldots, T_d\}$ and a *query pattern* P, the pattern matching problem consists in *counting* or *listing* all the *occurrences* of the string P in the collection $\mathcal{D}$. Instead, document retrieval is a family of problems aimed at retrieving *documents* from a set that are relevant to a query pattern. *Document listing* is the simplest and most basic task of the document retrieval family, its goal is to return all the documents in which P is present, while the *document counting*

[1]https://www.genomicsengland.co.uk/about-genomics-england/the-100000-genomes-project
[2]https://github.com/search?q=is:public
[3]https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

task only returns the number of such documents. The *top-k retrieval* task lists the k documents where P appears most often.

Compressed suffix arrays (CSAs) [26] are space-efficient representations of the suffix array (SA). Due to the fact that they can find the interval $[sp..ep]$ corresponding to $P[1..m]$ and access any cell $SA[i]$, they are commonly used as pattern matching indexes. The range $[sp..ep]$ is sufficient to *count* the number of occurrences of pattern P. Accessing $SA[sp], \ldots, SA[ep]$, we can locate the positions where the pattern occurs. Most CSAs need to store sampled SA values to support the access operation. This can represent a bottleneck in terms of required space .

Exploiting the close link between the *Burrows-Wheeler Transform* [4], $BWT[1..n]$, and the $SA[1..n]$ of $\mathcal{T}[1..n]$, the `FM-index` [10, 11] can represent a collection within its *statistical entropy*. For counting, the `FM-index` resorts to *backward search*. For locating, the `FM-index` stores sampled values of SA at regularly spaced text positions, say multiples of s. Statistical entropy is a measure that is insensitive to repetitiveness[20], so the FM-index is not an adequate index for repetitive datasets.

The *Run-Length FM-index* or `RLFM-index` [23, 24] is a variant of the `FM-index` that takes advantage of the compressibility of $BWT[1..n]$, which is formed by r runs of equal symbols. Besides, r is usually a relatively small number in repetitive collections ($r \ll n$). `RLFM-index` supports the count operation in $O(r)$ space. However, due to the required sampling, it needs a much larger $O(n/s)$ space to support locating in time proportional to s.

`r-index`[15] closed the long-standing problem of efficiently locating the occurrences of a pattern in a text using $O(r)$ space. The experiments show that the `r-index` outperforms all the other implemented indexes by orders of magnitude in space or in time to locate pattern occurrences on highly repetitive datasets.

Several other indexes are proposed based on other compression schemes that perform well on repetitive texts: indexes based on the *Lempel-Ziv parse* [22] of $\mathcal{T}$, with size bounded in terms of the number z of phrases [20, 12, 2]; indexes based on a approximation of the smallest *context-free grammar* that generates $\mathcal{T}$ and only $\mathcal{T}$ [5], with size bounded in terms of the size g of the grammar [8, 25]; indexes based on the size e of the smallest automaton (CDAWG) [3] recognizing the substrings of $\mathcal{T}$ [2]; index based on the *string attractors* [19] of $\mathcal{T}$, with size bounded in terms of the size γ of the smallest string attractor [27]. Many others are referenced by Gagie et al. [15] and Navarro and Prezza [27].

For the repetitiveness measures r, z, g, and e exist few known asymptotic bounds: $z \leq g = O(z \log(n/z))$ [29, 5, 18] and $e = \Omega(\max(r, z, g))$ [2]. The measure r is not comparable with z and g [2, 28]. However, theoretical and experimental results [24, 20, 2, 19] suggest that $\gamma < z < r \approx g \ll e$ on repetitive datasets.

Despite the abundance of indexes for pattern matching for repetitive collections, there are hardly any solutions for document retrieval tasks on repetitiveness scenarios[7, 13, 25, 9].

2 PROBLEM STATEMENT

Recent works have triggered several lines of research, from which some questions remain open. It forces us to revisit several pattern matching and document retrieval problems using the most recent advances that have been made in indexes for repetitive collections.

This thesis is focused on answering some of the fundamental questions: What other pattern matching operations can be supported within $O(r)$ space? How can we use the repetitiveness in the suffix array to improve document retrieval methods? What new queries make the most sense in an environment of repetitiveness, considering that many of the traditional queries lose relevance in this case?

3 RESEARCH GOALS

The goal of this thesis is to develop succinct indexes and compression-aware algorithms to support pattern matching and document retrieval tasks on repetitive string collections. To accomplish this goal, we will research the following topics:

- Designing practical indexes in terms of space that efficiently exploit the intrinsic repetitiveness in the collections. These indexes must support at least the most basic pattern matching operations. We will focus mainly in the following problems:
  - Designing more stable and resistant indexes to different repetitive scenarios. `r-index`[15] is an excellent solution for highly repetitive collections, but it has a problem: when repetitiveness decreases, the required space degrades quickly.
  - Supporting pattern matching operations within $O(r)$ space. A recent $O(r)$-space index[15] can efficiently count and locate the occurrences of a pattern. Extending these results to other operations is a challenge. Our initial aim is to study the task of extracting a substring from the text collection.
  - Improving suffix tree representation in terms of space. We will study the feasibility to represent the suffix tree of the collection $\mathcal{T}$ within $O(r)$ space.
- Designing flexible indexes to support more useful and relevant queries on repetitive scenarios. Document retrieval tasks include, perhaps, more natural queries for a collection of documents. Despite this, this area has been less developed over repetitive collections. For these reasons, we will research the following document retrieval problems:
  - Designing indexes that exploit the repetitiveness of the suffix array. Our recent work[9] proposes a competitive index based on grammar-compression to solve document listing task. We will work in reducing, even more, the space required by this solution. Also, we will study how to extend this index to support other document retrieval tasks.
  - Studying new problems and queries in the areas of pattern matching and document retrieval suited to the features of current repetitive collections. Traditional queries may lose relevance in an environment of repeatability: listing only the highest-level versions of the documents where a pattern appears in a collection of versioned documents versus listing all the documents. Very few works have addressed this kind of problems.

4 HYPOTHESES

It is possible to improve the space or time required by pattern matching operations on repetitive datasets taking advantage of the different kinds of repetitiveness scenarios. Combining recent results, it is possible to create indexes with better theoretical bounds in terms of space for document retrieval tasks as document listing.

On the other hand, competitive and efficient indexes can be implemented to support classical and novel problems in these fields, thus, augmenting the available practical solutions.

5 METHODOLOGY AND WORK PLAN

Our main goal is to design succinct data structures capable of representing collections in a small space taking advantage of their repetitiveness, as well as efficient algorithms capable of handling this representation of the collections to answer pattern matching and document retrieval queries.

To achieve our research goals and validate our hypotheses, we propose the following milestones:

5.1 Practical indexes

We will focus on indexes based on *Burrows-Wheeler Transform* (*BWT*). Despite extensive research on this area in the last years and the widespread use given to these indexes in fields such as biotechnology, the recently emerged results open new lines of future work.

5.1.1 Making r-index less sensitive to lower repetitiveness. The repetitiveness measure r (and thus `r-index`) is more sensitive than g and z to the decrease in repetitiveness. In particular, g and z are always $O(n/\log_\sigma n)$, and thus, the related indexes always use $O(n \log \sigma)$ bits. Instead, r can be as large as n [28], so in the worst case `r-index` can use $\Theta(n \log n)$ bits.

A challenge is making the `r-index` less sensitive to lower repetitiveness scenarios.

To locating occurrences of P in $O(r)$ space, `r-index` proposes a new sampling scheme. It samples the text character $\mathcal{T}[i]$ if and only if $\mathcal{T}[i]$ is the first or last character in its *BWT* run. This sampling can be inadequate in areas where the *BWT* runs are very short because it can produce oversampled areas on the collection $\mathcal{T}$.

We are going to work in designing sampling mechanisms to overcome this problem. Our first approach will be to create a kind of hybrid sampling scheme that uses the r-index's sampling over large BWT runs but, in oversampled areas of the text, it samples at regularly spaced text positions (like classic FM-index). Thus, we can handle the areas with higher and lower repetitiveness in different ways.

5.1.2 Self-index within $O(r)$ space. A *self-index* is a data structure built on $\mathcal{T}[1..n]$ that, in addition to supporting the count and locate operations, can efficiently extract any substring $\mathcal{T}[i..i+\ell]$. With the extract operation, a self-index can be a replacement of $\mathcal{T}$, avoiding its expensive storage.

The `r-index` is the first structure built on *BWT* runs that replaces $\mathcal{T}$ while retaining direct access. Gagie et al. [15] showed how `r-index` supports efficiently count and locate pattern occurrences. However, it requires $O(r \log(n/r))$ space to provide random access to the text or to extract any substring of length ℓ.

We are going to study whether efficient random access to the text is possible within $O(r)$ space. To achieve this, we only need to know the suffix array (*SA*) position p that points to the desired text position $\mathcal{T}[i]$. Once we get p, we can get $\mathcal{T}[i]$ using a backward search over the sampling scheme of the `r-index` (similar to locate operation). Since $p = SA^{-1}[i]$, our main concern is how to represent SA^{-1} within $O(r)$ space.

5.1.3 Suffix tree within $O(r)$ space. *Compressed suffix trees* provide much more complete functionality than self-indexes since they support the operations of a classical suffix tree using much less space. Generally, they are built using a compact representation of the *topology* of the suffix tree, a compressed suffix array (CSA), and other data structures to support some basic operations such as the *longest common prefix* (LCP).

Gagie et al. [14] proposed the first compressed suffix tree whose space is bound in terms of r, $O(r \log(n/r))$ words. `r-index` can return the *SA* cells that result from a pattern search within $O(r)$ space, but accessing an arbitrary *SA* cell requires $O(r \log(n/r))$ space.

We are going to study whether efficient random access to the suffix array is possible within $O(r)$ space. Compute $p = SA[i]$ is the dual problem of $i = SA^{-1}[p]$, so this is a very close problem to provide random access to the text.

5.2 Flexible indexes

We will focus on exploiting the repetitiveness of the suffix array (*SA*) and the *document array* (*DA*) to build document retrieval indexes.

5.2.1 Document retrieval indexes. Recently, we presented a simple and efficient *SA*-based index for document listing on repetitive string collections[9]. It uses a grammar-compressed document array (*GCDA*), and it obtains a better performance as the texts are more repetitive. Also, we adapted other previous solutions to run on our grammar-compressed document array, obtaining unprecedented performance on repetitive texts.

We are going to further reduce the space of *GCDA* and our index variants that use the grammar-compressed document array, by using a more clever encoding of the grammars that may nearly halve their space at a modest increase in time[16].

As the grammar compressor, *GCDA* uses `Re-Pair`[21] since it performs very well in practice. Despite this, it may be advisable to use another compressor for the set of inverted lists. We will experiment with a promising choice: the `Web graph compressor`[17], successfully used in another document listing solution, a variant of the `PDL` algorithm[13].

Another line of work is to extend the *GCDA* index to support *top-k document retrieval*, that is, find the k documents where P appears most often. For example, following previous ideas[13], we can store the list of documents where each nonterminal appears in decreasing order of frequency, and use algorithms developed for inverted indexes [1] on the $O(\log n)$ lists involved in a query. The frequency of the candidates can be efficiently counted on repetitive collections [13].

The *grammar-based indexes* grammar-compress the text collections. Grammar-compressing $\mathcal{T}$ means finding a *context-free grammar* that generates $\mathcal{T}$ and only $\mathcal{T}$. These indexes use the grammar as a substitute for $\mathcal{T}$, which provides good compression when $\mathcal{T}$ is repetitive. To search for a pattern $P[1..m]$, some grammar-based indexes[8, 7, 25] first find the *primary occurrences*, that is, those that appear when B is concatenated with C in a rule $A \longrightarrow BC$. They cut P into two non-empty parts $P = P_1P_2$, in the $m-1$ possible ways, to find the rules $A \longrightarrow BC$ such that P_1 is a suffix of the expansion of B and P_2 is a prefix of the expansion of C. The primary occurrences appear at these rules.

The *locally consistent grammars* guarantee that for equal substrings in the text $\mathcal{T}$, their subtrees in the grammar derivation tree are identical except, maybe, the external nodes in the coverture. We will use this interesting property of this kind of grammar that avoids cutting the query pattern in its $m-1$ partitions. We only need to cut the pattern in $O(\log m)$ positions, the ones that coincide with the external nodes of the corresponding parser subtree. Our initial research will try to combine locally-consistent grammars like the ones proposed by Christiansen et al. [6] with the grammar-based index for document listing proposed by Navarro [25]. This work can be an important enhancement to grammar-based indexes that use primary occurrence searching.

5.2.2 New pattern matching and document retrieval problems. Among the most repetitive datasets nowadays, we have the versioned document repositories like the ones formed by natural language documents or source code. In this kind of collections, we are not usually interested in obtaining all the versions of the same document where a pattern appears. However, there are some practical queries that we can handle.

Retrieving only documents in a given range of the collection is an interesting query. This range can be a time interval in datasets with a linear structure or a version subtree in hierarchical datasets. We will extend *GCDA* index[9] to support this query. We can store additional information in the nonterminals of the grammar. This facilitates the filtering of nonrelevant documents.

Given a hierarchical collection, with documents structured in a tree of versions, and different granularities (e.g. $x, x.y, x.y.z$) for each level of the tree, we can be interested in listing only the documents at a certain level of granularity, i.e., at a certain depth in the version tree. We will work in a solution to this problem handling the topology of the version tree with compact data structures.

REFERENCES

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 2011. *Modern Information Retrieval: The Concepts and Technology Behind Search.*

[2] Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. 2015. Composite Repetition-Aware Data Structures. In *Combinatorial Pattern Matching - 26th Annual Symposium, {CPM} 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings* (Lecture Notes in Computer Science). Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, (Eds.) Vol. 9133. Springer, Cham, 26–39.

[3] Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. 1987. Complete Inverted Files for Efficient Text Retrieval and Analysis. *Journal of the ACM (JACM)*, 34, 3, (July 1987), 578–595.

[4] Michael Burrows and David J. Wheeler. 1994. A block-sorting lossless data compression algorithm. Tech. rep. 124. Digital Equipment Corporation.

[5] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. 2005. The Smallest Grammar Problem. *{IEEE} Transactions on Information Theory*, 51, 7, (July 2005), 2554–2576.

[6] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. 2018. Optimal-Time Dictionary-Compressed Indexes. *arXiv e-prints*, (Nov. 2018), arXiv:1811.12779.

[7] Francisco Claude and J. Ian Munro. 2013. Document Listing on Versioned Documents. In *String Processing and Information Retrieval - 20th International Symposium, {SPIRE} 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings* (Lecture Notes in Computer Science). Oren Kurland, Moshe Lewenstein, and Ely Porat, (Eds.) Vol. 8214. Springer-Verlag, Berlin, Heidelberg, 72–83.

[8] Francisco Claude and Gonzalo Navarro. 2012. Improved Grammar-Based Compressed Indexes. In *String Processing and Information Retrieval - 19th International Symposium, {SPIRE} 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings* (Lecture Notes in Computer Science). Liliana Calderón-Benavides, Cristina González-Caro, Edgar Chávez, and Nivio Ziviani, (Eds.) Vol. 7608. Springer, Berlin, Heidelberg, 180–192.

[9] Dustin Cobas and Gonzalo Navarro. 2019. Fast, small, and simple document listing on repetitive text collections. *CoRR*, abs/1902.07599.

[10] Paolo Ferragina and Giovanni Manzini. 2005. Indexing Compressed Text. *Journal of the ACM*, 52, 4, (July 2005), 552–581.

[11] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. 2007. Compressed Representations of Sequences and Full-text Indexes. *ACM Transactions on Algorithms (TALG)*, 3, 2, (May 2007).

[12] Travis Gagie, Juha Gawrychowski Pawełand Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. 2014. LZ77-Based Self-indexing with Faster Pattern Matching. In *{LATIN} 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings* (Lecture Notes in Computer Science). Alberto Pardo and Alfredo Viola, (Eds.) Vol. 8392. Springer, Berlin, Heidelberg, 731–742.

[13] Travis Gagie, Aleksi Hartikainen, Kalle Karhu, Juha Kärkkäinen, Gonzalo Navarro, Simon J. Puglisi, and Jouni Sirén. 2017. Document Retrieval on Repetitive String Collections. *Information Retrieval Journal*, 20, 3, 253–291.

[14] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. 2018. Fully-Functional Suffix Trees and Optimal Text Searching in BWT-runs Bounded Space, (Sept. 2018).

[15] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. 2018. Optimal-Time Text Indexing in BWT-runs Bounded Space. In *Proceedings of the Twenty-Ninth Annual {ACM-SIAM} Symposium on Discrete Algorithms, {SODA} 2018, New Orleans, LA, USA, January 7-10, 2018* (SODA '18). Artur Czumaj, (Ed.) Society for Industrial and Applied Mathematics {SIAM}, Philadelphia, PA, USA, 1459–1477.

[16] Rodrigo González, Gonzalo Navarro, and Héctor Ferrada. 2014. Locally Compressed Suffix Arrays. *{ACM} Journal of Experimental Algorithmics ({JEA})*, 19, (Jan. 2014), 1.1:1–1.1:30.

[17] Cecilia Hernández and Gonzalo Navarro. 2014. Compressed Representations for Web and Social Graphs. *Knowledge and Information Systems*, 40, 2, (Aug. 2014), 279–313.

[18] Artur Jeż. 2016. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616, (Feb. 2016), 141–150.

[19] Dominik Kempa and Nicola Prezza. 2018. At the Roots of Dictionary Compression: String Attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing* (STOC 2018). ACM, New York, NY, USA, 827–840.

[20] Sebastian Kreft and Gonzalo Navarro. 2013. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483, 115–133.

[21] N. Jesper Larsson and Alistair Moffat. 2000. Off-line dictionary-based compression. *Proceedings of the {IEEE}*, 88, 11, (Nov. 2000), 1722–1732.

[22] Abraham Lempel and Jacob Ziv. 2006. On the Complexity of Finite Sequences. *IEEE Transactions on Information Theory*, 22, 1, (Sept. 2006), 75–81.

[23] Veli Mäkinen and Gonzalo Navarro. 2005. Succinct Suffix Arrays Based on Run-length Encoding. *Nordic Journal of Computing*, 12, 1, (Mar. 2005), 40–66.

[24] Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. 2010. Storage and Retrieval of Highly Repetitive Sequence Collections. *Journal of Computational Biology*, 17, 3, (Mar. 2010), 281–308.

[25] Gonzalo Navarro. 2019. Document listing on repetitive collections with guaranteed performance. *Theoretical Computer Science*, 772, 58–72.

[26] Gonzalo Navarro and Veli Mäkinen. 2007. Compressed Full-text Indexes. *{ACM} Computing Surveys*, 39, 1, (Apr. 2007).

[27] Gonzalo Navarro and Nicola Prezza. 2019. Universal Compressed Text Indexing. *Theoretical Computer Science*, 762, 41–50.

[28] Nicola Prezza. 2016. *Compressed Computation for Text Indexing*. Ph.D. Dissertation. University of Udine.

[29] Wojciech Rytter. 2003. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302, 1-3, (June 2003), 211–222.

[30] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. 2015. Big Data: Astronomical or Genomical? *PLOS Biology*, 13, 7, 1–11.