Indexing Highly Repetitive Collections via Grammar Compression

ALEJANDRO PACHECO MORALES, Department of Computer Science, University of Chile

The deluge of data in our society has made compression techniques play a fundamental role in data storage and processing tasks. Despite the amount of data generated every day, the information that the data contains does not grow at the same rate. For this reason, the collections that have a fast growth are highly repetitive. Some examples of these are, DNA sequences, repositories of codes, Wikipedia, among others. The algorithms based on dictionary compression schemes have been the most successful ones in these collections. The most popular of them is the Lempel Ziv (LZ77) parse, however, it has a fundamental disadvantage, which is that to access random positions in the text, you basically need to decompress the entire text. A second approach that presents an alternative to this problem is the Context-Free Grammar (CFG). CFGs have proven to be an elegant and effective compression model. These have been applied to various problems in the literature. In this proposal, we will focus on the main drawbacks of Grammar-based compressors and self-indexes in repetitive collections.

Additional Key Words and Phrases: Context-Free Grammar, repetitive collections, compression, self-index

INTRODUCTION

At present, the amount of data that people generate is stunning. Every day, $2.5x10^{30}$ bytes are created. The 90% of the data that exists in the world was created between 2016-2017 [1]. Consider that, currently, Google processes more than 40,000 searches per second. This massive creation of data has brought along the appearance of new challenges for the collection, storage, and processing of these large amounts of data.

In this data deluge, compression techniques are fundamental for disk storage and data transfer. There are two fundamental techniques of compression: lossy (inaccurate) or lossless (exact). Lossless compression can be reverted to get the original data, while lossy compression loses details or introduces small errors when it tries to obtain the original data. Lossy compression can be used for images, videos, voice, etc. Lossless compression is used mostly in texts (strings), where each character matters. Currently, there are large text databases which must be stored and processed efficiently. The text collections that grow fast are highly repetitive. Some examples of these are natural language databases such as Wikipedia; DNA or protein sequences and source code repositories.

The principal feature of highly repetitive collections is that most of the substrings in the collection can be obtained from another one by performing a small number of edit operations. For example, collections of DNA sequences store many genomes of the same species, and in many cases, these sequences are very similar. Two human sequences differ only by 0.1% [9]. Another example is Wikipedia, the giant encyclopedia presented a report where it was shown that there were 20 reviews per article in its content of 10 TB. They also showed that the tendency was for revisions to grow faster than articles, which increases the repeatability. In this thesis project, we will focus on collections of highly repetitive strings and therefore lossless compression.

Two fundamental techniques for achieving lossless compression are statistical compression and compression based on dictionaries. The statistical compression attempts to exploit the frequency of the characters in the text to obtain a compression close to the entropy of the text [8]. However, it has been demonstrated that methods based on statistical compression do not perform adequately in repetitive collections [21]. On the other hand, the compression based on dictionaries tries to exploit the redundancy of the text by creating a dictionary of phrases and representing the text as a sequence of these phrases. The most successful compressors in this family are based on the Lempel-Ziv (LZ77) [24] factoring (z size); Context-Free Grammar(CFG) [19] (in size g); Bidirectional Macro Schemes(MBS) [33] (of size b); and Collage Systems(CS) [18] (of size c).

[1]https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#11b2464760ba

Author's address: Alejandro Pacheco Morales, Department of Computer Science, University of Chile, alpachec@dcc.uchile.cl.

Lempel-Ziv is, as far as we know, the most popular and effective compression method for repetitive texts. In addition, it can be executed in linear time [15] and even in external memory [16]. LZ77 has an important drawback in order to access random positions of the compressed text, it is necessary to unzip it from the beginning. Conversely, compression based on CFG [19] is an alternative that offers better guarantees in this regard.

CFGs have proven to be an elegant and efficient model for data compression. The idea of grammar-based compression is well-known since the seventies. Given a text $T[1, n]$, we can construct a CFG Gr that only generate $T[1, n]$. Then, instead of storing $T[1, n]$, it is possible to store Gr, and consequently, it may be possible to achieve compression. Although, finding the smallest grammar for a text is NP-complete [4], there are several grammar construction algorithms that guarantee, at most, a logarithmic blowup on the LZ77 parse [4, 13, 14, 31, 32]. Some of the best known compression algorithms based on grammar are Re-Pair [23], Sequitur [27] and LZ78 [34]. In practice, however, Re-Pair has the best performance. Re-Pair algorithm is a heuristic method that runs in linear time and obtains, in most cases, grammars of size similar to that of the LZ77 parse.

For this reason, RePair has become the compressor of choice to build grammar-based compressed data structures [1, 3, 7]. However,a critical problem with RePair, is that, despite operating in linear time and space, in practice, it can be built only on texts that are about one-tenth of the main memory. This impairs significantly its performance on large datasets.

Oriented to solve this problem, Gagie et al. [12] presented an algorithm that responds well to very large and very repetitive collections. This algorithm processes a file of 55GB in less than one hour and uses only 650 MB of main memory. Nevertheless, the performance of the method deteriorates when the collection has low repetitiveness, taking 38h of time and using 15GB of RAM for another collection of similar size. On the other hand, Nunes et al. [30] presented a new grammar compressor based on the Induced-sorting algorithm of Nong et al. [29] which presents a competitive compression rate with Re-Pair, compresses much faster, uses much less memory during the construction and is slower in decompression. The performance of this proposal in large collections has not been studied, which is interesting because it could offer advantages over the algorithm of Gagie et al. [12] and RePair.

The CFG has also been used to develop compact data structures capable of extracting information from compressed texts, these structures are called self-indexes. The self-indexes are Compact Data Structures [25] that represent the text in a compressed way and allows to count the occurrences of a pattern (count), return the positions of the original text where these occurrences appear (locate) and extract any substring of the text (display).

There are few indexes based on grammars [5, 6, 25] and only one of these has a practical implementation [7]. These mentioned indexes present two fundamental limitations: none of them is capable of counting the number of occurrences without reporting their positions, and all of them are penalized at the time of the locate option by a factor of m^2 [11], where m is the length of the pattern. Recently, Navarro [26] presented the first index capable of counting the occurrences of a pattern of length m in $O(m^2 + m \log^{2+\epsilon} g)$. The optimal time for counting is $O(m)$, consequently, this solution is distant from the optimal one, however, it is the first approach to a solution.

On the other hand Kociumaka et al. [20] built an index using a Run-Length Context-Free Grammar (RLCFG) [28] with properties of local consistency in a space $O(\gamma \log n/\gamma)$ where γ is the size of the string attractor. n string attractor of a string $T[1, ..., n]$ is a set of positions of T such that, any substring $T[i, ..., j]$ has at least one occurrence that crosses one of these positions [17]. We say that a grammar that only generates a text $T[1, .., n]$ is locally consistent if, given two equal substrings $T[i, ..., j] = T'[i', ..., j']$, the sequences S and S' of labels taken from the nodes in preorder of their corresponding parser trees, can be divided in three subsequences $S = BNE$ and $S' = B'NE'$, and $|B|$,$|E|$,$|B'|$,$|E'| \leq c$, where c is a constant [20].

This index counts the occurrences of a pattern of length m in $O(m + \log^{2+\epsilon} n)$ time and locate them in $O(m + (occ + 1)\ log^{\epsilon} n)$. These bounds beat the existing state of the art. Furthermore, by increasing the space by a factor of $\log^{\epsilon} n$ they can obtain optimal times in both operations. This index, however, has not been implemented yet, therefore, its practical performance is unknown.

PROBLEM STATEMENT

Nowadays, repetitive collections constitute one of the types of collections with one of the fastest growths. The grammar-based compressor with the best practical performance of the state of the art (Repair) cannot be applied to very large repetitive collections due to its large memory consumption. In addition, to storing these data collections, we want to be able to process them in compact space in order to retrieve relevant information. Self-indexes are compact data structures that make this possible. Although grammars have proven to be excellent compression models, there are not many approaches in grammar-based self-indexes for repetitive collections, and only one of them has a practical implementation. Furthermore, these have some deficiencies. There is only one index that supports the count operation with time very far from the optimum and the time of the operation locate of all these self-indexes is affected by a factor of m^2. Recently, new brands of grammar have been developed in the state of the art. These ones have interesting properties that can be exploited in order to build self-indexes with better theoretical properties.

RESEARCH GOALS AND CONTRIBUTION

The objective of this thesis is to develop indexes and compression algorithms based on grammars capable of storing and processing large highly repetitive collections of strings. In order to conduct this research we will focus on the following topics:

(1) *Building grammar compressors for large text:* Designing variants of grammar-based compressors with practical implementations that perform adequately in large highly repetitive collections. Our specific objectives are listed below.
  - To improve the proposal of Gagie et al. [12] in order to make it less sensitive to lower repetitiveness.
  - To adapt the grammar compressor proposed by Nunes et al. [30] to use it on highly repetitive collections with large size

(2) *Grammar self-indexes :* Designing grammar-based indexes with practical implementations and theoretical improvements in locating, counting and displaying bounds. Our specific objectives are listed below.
  - To improve the existing grammar-based index in order to obtain better times for the locate and count operations, eliminating the factor m^2 that affects them. In addition, we propose to study the impact on time and space complexity of this improvement in practical performance.
  - To extend the current results of the state of the art in grammar-based indexes to Run-Length Grammars, which are more powerful than Context-Free Grammars.
  - To design practical schemes in order to support the operation count in the grammar-based indexes of the literature.
  - To evaluate the practical performance of the grammar-based self-indexes created on other types of grammar (ex. Induced Sorting) and compare their performance against Re-Pair self-indexes.
  - To design a self-index based in a locally consistent grammar that has space bounded by the size of the grammar and that exploits the local consistency properties to improve the times of the locate, count and display operations.

HYPOTHESES

We hold two main hypotheses, which are:

(1) It is possible to improve the grammar-based compressors of the state of the art in order to build a compressor that performs adequately in large repetitive collections.

(2) It is possible to improve the Grammar-based self-indexes focusing in the following aspects:
- Memory and time improvement in the building phase.
- Count operation support.
- Locate time improvement.
- Index designing based on context-free grammars and run-length grammars with locally consistent properties.

METHODOLOGY AND WORK PLAN

Below, we describe our work plan in order to validate our hypotheses and accomplish our objectives.

Grammar Compressors for large collections

In order to achieve our first goal of designing variants for grammar-based compressors with practical implementations that work adequately on large repetitive collections, we draw the following work plan:

(1) To conduct a review of the literature about the different approaches that have been used for the compression of large repetitive collections.

(2) To design a new compressor based on the proposal made by Gagie et al. [12] called Big Re-Pair. This method divides the original text $T[1, n]$ into s_i phrases using a hash function. These phrases are indexed in a dictionary, and a second sequence S is formed by the concatenation of indexes corresponding to the s_i in the same order of the original text.
Then, the grammar is built applying Re-Pair to each phrase s_i and to the new sequence S and then combining both grammars. The problem with this method is that unlike the s_i for which we can control its size, the sequence S can be not very repetitive. Therefore, applying directly Re-Pair is not adequately if S is very large. It is possible to improve this proposal by recursively applying the same scheme to the indexed sequence S until we obtain a sequence to which we can apply Re-Pair directly.

(3) To evaluate the performance of the compressor proposed by Nunes et al. [30] in large repetitive collections and compare it against the state of the art. As we mentioned above in our problem statement, the compressors based on grammar have practical drawbacks in large repetitive collections. The proposal of Nunes et al. [30] makes better use of memory and is faster than Re-Pair when it compresses the text. In addition, it reaches a compression ratio close to Re-Pair. These are characteristics that make us expect a good performance in large repetitive collections.

Grammar Indexes

Our second goal is oriented to designing grammar-based indexes with practical implementations and theoretical improvements in locating, counting and displaying bounds. For achieving this objective we establish the following guidelines:

(1) To implement the grammar-based indexes of the state of the art [6, 25] and evaluate their performance compared to the state of the art of self-indexes based on compression by dictionaries in repetitive collections [5, 11, 22].

(2) To improve the proposal of Claude and Navarro [6] in order to obtain better times for the operation locate. The z-fast-trie data structure and the Karp-Rabin string signature have been combined in other indexes [2, 10, 11] for reducing the prefix search time of the m possible partitions of the pattern. The index of Claude and Navarro [6] can be adapted to use these structures. In this way, it is possible to reduce the m^2 factor that affects its locate time. In addition, to evaluate the impact of this improvement in the practical performance of the index.

(3) To evaluate the performance of the grammar-based indexes in the state of the art [5, 6, 25] on a grammar constructed with the parser proposed by [30] and to compare them to the results obtained on a Re-Pair grammar. As we mentioned above, this new parser uses less space during compression than Re-Pair, builds the grammar faster than Re-Pair and has a very competitive compression rate. These features are very suitable for the construction of self-indexes. Therefore, it is possible to obtain competitive indexes with better construction time and better memory usage.
(4) To design new schemes to support the count operation in grammar-based self-indexes and propose practical adaptations for the existing indexes if it is possible. As far as we know, the only grammar-based self-index that supports the count operation is [26], therefore the rest of the indexes that we had mentioned can be improved in this regard.
(5) To extend the obtained results for locating, counting and displaying in Context-Free Grammars to Run-Length Grammars. The Run-Length Grammars include Context-Free Grammars and are more powerful, for example for the strings family $S = a^n$ the smallest CFG takes $\Theta(\log n)$, while the size of an RLG is $O(1)$. Therefore, it is possible to improve the results of the state of the art by extending them from CFG to RLG.
(6) To conduct a survey to identify the existent approaches to build grammars with local consistent properties. By means of the study of local consistent properties in grammar, we will be able to design indexes that take advantage of these properties.
(7) To design and implement a new self-index based on locally consistent grammar, with better theoretical bounds than the grammar-based indexes of the state of the art for the operations locate, count and display, and space bounded by the number of rules of the grammar. As we mentioned before the approach of [20] uses a Run-Length Grammar with local consistent properties, however, the space of this index is $O(\gamma \log n/\gamma)$ bounded by the size of the smallest string attractor γ, not by the size of the grammar. It were proved that $g \leq \gamma$, but it is possible to built an index with space $O(g) \leq O(\gamma \log n/\gamma)$.

REFERENCES

[1] A. Abeliuk, R. Cánovas, and G. Navarro. Practical compressed suffix trees. *Algorithms*, 6(2):319–351, 2013.
[2] P. Bille, M. B. Ettienne, I. L. Gørtz, and H. W. Vildhøj. Time-space trade-offs for lempel-ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018.
[3] N. R. Brisaboa, A. Gómez-Brandón, G. Navarro, and J. R. Paramá. Gract: A grammar-based compressed index for trajectory data. *Inf. Sci.*, 483:106–135, 2019.
[4] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
[5] F. Claude and G. Navarro. Self-indexed grammar-based compression. *Fundam. Inform.*, 111(3):313–337, 2011.
[6] F. Claude and G. Navarro. 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*, pages 180–192, 2012.
[7] F. Claude, A. Fariña, M. A. Martínez-Prieto, and G. Navarro. Universal indexes for highly repetitive document collections. *Inf. Syst.*, 61: 1–23, 2016.
[8] T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.).* Wiley, 2006.
[9] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. Efficient storage of high throughput dna sequencing data using reference-based compression. *Genome research*, 2011.
[10] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi. 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*, pages 731–742, 2014.
[11] T. Gagie, G. Navarro, and N. Prezza. Fully-functional suffix trees and optimal text searching in bwt-runs bounded space. *CoRR*, abs/1809.02792, 2018.
[12] T. Gagie, T. I, G. Manzini, G. Navarro, H. Sakamoto, and Y. Takabatake. Rpair: Rescaling RePair with Rsync. *arXiv e-prints*, 2019.
[13] A. Jez. A really simple approximation of smallest grammar. *CoRR*, abs/1403.4445, 2014.
[14] A. Jez. Approximation of grammar-based compression via recompression. *Theor. Comput. Sci.*, 592:115–134, 2015.
[15] J. Kärkkäinen, D. Kempa, and S. J. Puglisi. Linear time lempel-ziv factorization: Simple, fast, small. In *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings*, pages 189–200, 2013.

[16] J. Kärkkäinen, D. Kempa, and S. J. Puglisi. Lempel-ziv parsing in external memory. In *Data Compression Conference, DCC 2014, Snowbird, UT, USA, 26-28 March, 2014*, pages 153–162, 2014.

[17] D. Kempa and N. Prezza. At the roots of dictionary compression: string attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840, 2018.

[18] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theor. Comput. Sci.*, 298(1):253–272, 2003.

[19] J. C. Kieffer and E. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Information Theory*, 46(3): 737–754, 2000.

[20] T. Kociumaka, G. Navarro, and N. Prezza. Optimal-time dictionary-compressed indexes. *arXiv preprint arXiv:1811.12779*, 2018.

[21] S. Kreft and G. Navarro. Lz77-like compression with fast random access. In *2010 Data Compression Conference (DCC 2010), 24-26 March 2010, Snowbird, UT, USA*, pages 239–248, 2010.

[22] S. Kreft and G. Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013.

[23] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Data Compression Conference, DCC 1999, Snowbird, Utah, USA, March 29-31, 1999.*, pages 296–305, 1999.

[24] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Information Theory*, 22(1):75–81, 1976.

[25] G. Navarro. *Compact Data Structures - A Practical Approach.* Cambridge University Press, 2016.

[26] G. Navarro. Document listing on repetitive collections with guaranteed performance. *Theor. Comput. Sci.*, 772:58–72, 2019.

[27] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7: 67–82, 1997.

[28] T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda. Fully dynamic data structure for LCE queries in compressed space. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 72:1–72:15, 2016.

[29] G. Nong, S. Zhang, and W. H. Chan. Linear suffix array construction by almost pure induced-sorting. In *2009 Data Compression Conference (DCC 2009), 16-18 March 2009, Snowbird, UT, USA*, pages 193–202, 2009.

[30] D. S. N. Nunes, F. A. Louza, S. Gog, M. Ayala-Rincón, and G. Navarro. A grammar compression algorithm based on induced suffix sorting. In *2018 Data Compression Conference, DCC 2018, Snowbird, UT, USA, March 27-30, 2018*, pages 42–51, 2018.

[31] W. Rytter. Application of lempel-ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3): 211–222, 2003.

[32] H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3(2-4):416–430, 2005.

[33] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982.

[34] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, 1978.