Accelerating computation on compressed data via Context-Free Grammars.

FRANCISCO OLIVARES, University of Chile, Chile

Nowadays, the magnitude of the size of the data to be handled has grown widely. In many areas, it is necessary to work with data sets that are too big to fit in main memory. This situation has led to an increasing interest in efficiently compressing the data. In many situations, just compressing is not enough. It is also necessary to represent data in a way that can be queried without needing to decompress it. Among others, a compression method that permits working in compressed form is grammar compression. By using grammar compression, exponential compression rates can be achieved. On the other hand, direct access can take logarithmic time. Unlike other compression methods, grammar compression can achieve larger space reductions in either, repetitive and non-repetitive data. Recently published papers have shown that grammar compression can be achieved efficiently through a locally consistent parsing, which refers to a parsing where identical elements are parsed identically, with the possible exception of their extremes.

In this research, we propose to improve the knowledge and applications of grammar-based compression. We will study an efficient method of grammar compression constructed through locally consistent parsing. Additionally, the research will focus on improving queries over repetitive data while representing it by a grammar. Finally, we will research a grammar-based compression method to improve algorithmic efficiency on abstract data types.

Additional Key Words and Phrases: Compressed Data; Grammar compression; Repetitive sequences; Locally Consistent Parsing

1 INTRODUCTION

Over recent years, a large increase in the size of the data with which to work has been observed [24]. Many areas where the amount of data has grown faster are considered of vital importance for the modern society: astronomical data, genome collections, versioned document collection, software repositories, and other sources of data consisting of sequences of larger size than what could have seen a couple years ago [27]. Due to this accelerate growth, the interest for compressed representation of data has increased considerably. But just representing data in compressed form is not enough. We need compressed representations of data that permit working without needing to decompress it anymore [34]. Representations satisfying these constraints are said to be *compact* representations [24]. In that scenario, the problem has two dimensions: compressing efficiently and working over compressed form [21].

Interestingly, a significant fraction of growing-faster data is often highly repetitive, such as genome collections or astronomical data. To compress efficiently, we usually resort to statistical compression. However, statistical compression is not able to capture repetitiveness as a compression factor [17]. To achieve space reductions on repetitive data, other kinds of compressors are often used, such as Lempel-Ziv [18] or the run-length-compressed Burrows-Wheeler [23].

Author's address: Francisco Olivares, University of Chile, Computer Science Department – DCC, Santiago, Chile, folivares@uchile.cl.

A question that still has no answer is *how to lower bound compressibility on repetitive sequences*. For statistical compression the answer is clear: the tightest lower bound is the statistical entropy [33]. Unfortunately, it has not been possible to find similar measures capturing repetitiveness. Instead, repetitiveness is measured in ad-hoc terms, as a result of how much each compressor can capture it. For a given length-n string S, some of those measures are: the number z of phrases produced by Lempel-Ziv compressor [18], the number $b \leq z = O(b \log \frac{n}{b})$ of phrases of the smallest parsing produced by bidirectional macro schemes [34], the size $g = O(z \log \frac{n}{z})$ of the smallest context-free grammar that generates only S [13], the size $g_{rl} \leq g$ of the smallest run-length grammar that generates only S [28], the number $r = O(b \log^2 n)$ of the *runs* in the Burrows-Wheeler transform [3], and the size $\gamma = O(b)$ of the smallest attractor [12], where γ asymptotically lower bounds all the others measures [12].

Although many of those measures are promising for lower bounding compressibility on repetitive sequences, most of them are NP-hard to compute, are non monotonic upon symbols append, or are sensitive to simple string transformations [4, 12, 24, 34]. Recently, the measure δ was proposed for this task, which better captures the concept of compressibility in repetitive strings and is more manageable to deal with: it can be computed in linear time, is monotone when the string changes in the extremes, and is insensitive to string reversals or to alphabet permutations [4]. Since $\delta \leq \gamma$ [14], δ lower bounds all measures showed above [15].

In the last years, research on representing the data in compressed form through straight-line programs (SLP) has increased heavily, given that this kind of compression method can achieve exponential size reduction, works well with repetitive and no repetitive data, and can be constructed in linear time [13, 14]. Additionally, direct access to any element takes logarithmic time [24]. An SLP is a context-free grammar that generates only one string [21]. The size g of the smallest such grammar is NP-HARD to compute, but there are certain schemes to approximate quite closely the smallest grammar, such as grammar constructions showed by Rytter [30] and Jeż [10, 11]. Although these schemes have good theoretical bounds, in practice, RePair algorithm gets better performance constructing a grammar representing a given string [32], but its approximation ratio from g is not known.

If in an SLP we allow rules of the form $A \rightarrow B^n$, with $n \geq 2$, the resulting grammar is called *run-length straight-line program* (RLSLP) [28]. These grammars encompass regular context-free grammars but are more powerful, given that an SLP is a particular case of an RLSLP [4]. There exist string families where the smallest SLP for representing them has size $O(\log n)$, while the smallest RLSLP has size $O(1)$ [15]. The size g_{rl} of the smallest RLSLP is also NP-HARD to compute [28].

1.1 The measure δ.

The measure δ was originally introduced by Raskhodnikova et al. [29] in a stringology context, but it was formally defined recently by Christiansen et al. [4] as a way to construct a grammar of size $O(\gamma \log \frac{n}{\gamma})$ without knowing the size of the smallest string attractor γ, which is NP-HARD to compute [12].

For a given length-n string S, let $d_k(S)$ the number of distinct length-k substrings in S, where the set of $\{d_k(S) : k \in [1..n]\}$ is known as the *substring complexity* of S. Then, δ is defined [4] as:

$$\delta = \max\{d_k(S)/k : k \in [1..n]\}.$$

Although is not possible to represent any length-n string in $O(\delta)$ space (i.e., it is not reachable space) [14], Kociumaka et al. [15] proved that $O(\delta \log \frac{n}{\delta})$ is worst-case optimal space.

1.2 Grammar compression and Locally Consistent Parsing

Locally consistent parsing (LCP) is a way of partitioning a string into non-overlapping blocks [2]. Consistency means that identical substrings are partitioned identically with the possible exception of their extremes [1].

Recently published papers have shown a relationship between better performance of RLSLPs and their construction through an LCP, while building a data structure for matching a pattern over a given string [4, 11, 14–16, 29]. The data structures offering $n^{o(1)}$-time string matching over a collection of strings are known by the name of *indexes*. Grammar-based indexes have been very successful for getting good bounds for time construction, query time, and space.

Christiansen et al. [4] built a grammar-based index of size $O(\gamma \log \frac{n}{\gamma})$ that counts the occurrences occ of P in S in $O(m + \log^{2+\epsilon} n)$ time, and locates them in $O(m + (occ + 1) \log^{\epsilon} n)$ time, where ϵ is a small constant fixed at construction time. Even by increasing the space to $O(\gamma \log(n/\gamma) \log^{\epsilon} n)$, they reduced the locating time to the optimal $O(m + occ)$, and within space $O(\gamma \log(n/\gamma) \log n)$ they reduced counting time to the optimal $O(m)$. Their indexes can be constructed in $O(n)$ space and $O(n \log n)$ expected time through consecutive applications of LCP. Those times for finding and locating are the best known to date.

Kociumaka et al. [15] built a grammar-based index of size $O(\delta \log \frac{n}{\delta})$ that counts the occurrences of P in S in $O(m \log^{2+\epsilon} n)$ time and locates them in $O(m \log n + occ \log^{\epsilon} n)$ time. The space $O(\delta \log \frac{n}{\delta})$ is worst-case optimal in terms of δ [14] and, since $\delta \leq \gamma$, this space improves the previous one. They construct their grammar through consecutive applications of LCP.

1.3 Algorithmics on RLSLP – Compressed Sequences

Algorithmics on compressed strings (ACS) has been studied mainly for three scenarios. One of them refer to a large data that have to be not just stored in compressed form, but the original data has to be queried in that representation, given that it is not feasible to decompress it to analyze its contents, for example, large genome collections [7].

Another relevant case of ACS appears when large strings (generally highly compressible) are produced by an algorithm as an intermediary representation. Compressing it may lead to improved efficiency of the algorithm, for example, problems related to combinatorial group theory [9].

Finally, ACS has been relevant for making explicit regularity in a given string. One of the most remarkable examples of this situation can be seen in highly repetitive data [23].

For all situations above, there are experiences compressing data through regular context-free grammars [4, 10, 13, 20]. However, it has not been studied deeply for run-length grammar compression.

2 RESEARCH GOALS

2.1 General goal

The main goal of this proposal is to develop new grammar-based compressed representation of data, using LCP as a construction tool, which improves current solutions. In that way that they are faster and support more elaborate queries than actual ones.

2.2 Specific goals

In the search for improving knowledge about grammar compression and its applications for working over compressed data, the following objectives will be considered:

- Improving index query-time efficiency by constructing it through LCP [1].
- Giving new functionalities to indexes, as can be seen in other data structures:
  - Functionalities more similar to those that can be seen in a suffix tree [6, 8, 31].
  - Document retrieval [25].
  - Queries specifically though over repetitive sequences [26].
  - Efficient counting of occurrences of a given pattern [4].
- Improving the understanding about grammar compressors with better performance: searching for an approximation ratio from g of RePair [10, 11, 30, 32].
- Designing grammar-based compression for other kind of data:
  - Tree compression [22].
  - Graph compression [35].
  - Sparse matrix compression [5].
- Designing algorithms to speed-up solutions of abstract algebra problems using grammar-based compression:
  - Sparse matrix multiplication [5].
  - Word problem for automorphism group and subgroups [20].
  - Word problem for graph product of groups [19].

3 RESEARCH HYPOTHESIS

Based on previous work about compression through RLSLP representation of data and based on the objectives fixed for this research, the following hypothesis are formulated:

- A linearly-constructed RLSLP-based index of size $\delta \log \frac{n}{\delta}$ can solve count and locate in optimal time [4, 14, 15].
- An index constructed through an RLSLP can search for a regular expression in expected sublinear time [6, 8, 31].
- Word problem for automorphism group and subgroups [9] can be solved in polynomial time by representing group generators set in compressed form through an RLSLP [19–21].
- Multiplication of sparse squared matrix of n rows with a length-n vector can be done in $o(n^2)$ by representing the matrix through an RLSLP [5].
- There exists a measure describing a $(\log n)$-approximation ratio of RePair from g [10, 11, 30, 32].

4 METHODOLOGY AND WORK PLAN

4.1 Methodology

The general methodology consists of studying the theoretical notions of the problem with formal space and time bounds guarantees.

4.2 Work Plan

According to research goals and hypothesis, the following work plan will be considered:

- **Second semester of 2021**.
  - **Improving grammar-based index efficiency**: we will study a grammar-based index of size $O(\delta \log \frac{n}{\delta})$, constructed in worst-case linear-time, which solves counting and locating in optimal time [24]. Research will focus on using ideas by Kociumaka et al. [15] and Christiansen et al. [4] for constructing such an index.
- **First semester of 2022**.
  - **Improving the understanding about grammar compressors with better performance**: we will study a measure that can describe how close the size of resulting grammar is given by RePair algorithm from the smallest size g [10, 11, 13, 30, 32].
  - **Giving new functionalities to indexes, as can be seen in other data structures**: we will study a mechanism for providing to indexes the ability of answer queries which are usually made in a suffix tree [6, 8, 31]. In addition, we will research a way of improving functionality of document retrieval in indexes; in particular, we will focus on using ideas by Navarro [25], but constructing the index through LCP. Finally, we will research a method that can allow indexes to count on the skill of answer queries specifically though over repetitive sequences [26] and efficient counting [4].
- **Second semester of 2022**.
  - **Designing grammar-based compression for other kind of data**: based on previous work, we will study grammar-based compression on trees [22], on graphs [35], and matrix [5]; and its applications for querying and operating over compressed form.
  - **Designing algorithms to speed-up solutions of abstract algebra problems using grammar - based compression**: we will study a grammar-based compression method to improve previous results for the word problem for automorphism groups [20] and the multiplication of sparse matrix with vectors [5].

REFERENCES

[1] Tuğkan Batu and S. Cenk Sahinalp. 2005. Locally Consistent Parsing and Applications to Approximate String Comparisons. In *Developments in Language Theory*, Clelia De Felice and Antonio Restivo (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 22–35.

[2] Or Birenzwige, Shay Golan, and Ely Porat. 2020. Locally Consistent Parsing for Text Indexing in Small Space. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '20)*. Society for Industrial and Applied Mathematics, USA, 607–626.

[3] M. Burrows and D. J. Wheeler. 1994. *A block-sorting lossless data compression algorithm.* Technical Report 124. Digital Equipment Corporation.

[4] Anders Christiansen, Mikko Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. 2020. Optimal-Time Dictionary-Compressed Indexes. *ACM Transactions on Algorithms* 17 (12 2020), 1–39. https://doi.org/10.1145/3426473

[5] A. Francisco, T. Gagie, S. Ladra, and G. Navarro. 2018. Exploiting computation-friendly graph compression methods for adjacency-matrix multiplication. *Data Compression Conference Proceedings* 2018-March (2018), 307–314.

[6] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. 2020. Fully Functional Suffix Trees and Optimal Text Searching in BWT-Runs Bounded Space. *J. ACM* 67, 1, Article 2 (Jan. 2020), 54 pages. https://doi.org/10.1145/3375890

[7] Travis Gagie and Simon J. Puglisi. 2015. Searching and Indexing Genomic Databases via Kernelization. *Frontiers in Bioengineering and Biotechnology* 3 (2015), 12. https://doi.org/10.3389/fbioe.2015.00012

[8] Roberto Grossi and Jeffrey Vitter. 2000. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching (Extended Abstract). *SIAM J. Comput.* 35 (03 2000). https://doi.org/10.1145/335305.335351

[9] Niko Haubold and Markus Lohrey. 2009. Compressed Word Problems in HNN-Extensions and Amalgamated Products. In *Computer Science - Theory and Applications*, Anna Frid, Andrey Morozov, Andrey Rybalchenko, and Klaus W. Wagner

(Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 237–249.

[10] Artur Jeż. 2015. Approximation of grammar-based compression via recompression. *Theoretical Computer Science* 592 (2015), 115–134. https://doi.org/10.1016/j.tcs.2015.05.027

[11] Artur Jeż. 2016. A really simple approximation of smallest grammar. *Theoretical Computer Science* 616 (2016), 141–150. https://doi.org/10.1016/j.tcs.2015.12.032

[12] Dominik Kempa and Nicola Prezza. 2018. At the Roots of Dictionary Compression: String Attractors. *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 827–840. https://doi.org/10.1145/3188745.3188814

[13] J.C. Kieffer and En-Hui Yang. 2000. Grammar Based Codes: A New Class of Universal Lossless Source Codes. *Information Theory, IEEE Transactions on* 46 (06 2000), 737 – 754. https://doi.org/10.1109/18.841160

[14] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. 2020. Towards a Definitive Measure of Repetitiveness. In *LATIN 2020: Theoretical Informatics*, Yoshiharu Kohayakawa and Flávio Keidi Miyazawa (Eds.). Springer International Publishing, Cham, 207–219.

[15] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. 2021. Towards a Definitive Compressibility Measure for Repetitive Sequences.

[16] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. 2021. Internal pattern matching queries in text and applications. (2021).

[17] Sebastian Kreft and Gonzalo Navarro. 2013. On compressing and indexing repetitive sequences. *Theoretical Computer Science* 483 (2013), 115–133. https://doi.org/10.1016/j.tcs.2012.02.006 Special Issue Combinatorial Pattern Matching 2011.

[18] A. Lempel and J. Ziv. 1976. On the Complexity of Finite Sequences. *IEEE Transactions on Information Theory* 22, 1 (1976), 75–81. https://doi.org/10.1109/TIT.1976.1055501

[19] Markus Lohrey. 2010. Compressed Membership Problems for Regular Expressions and Hierarchical Automata. *Int. J. Found. Comput. Sci.* 21 (10 2010), 817–841. https://doi.org/10.1142/S012905411000757X

[20] Markus Lohrey. 2011. Compressed Word Problems for Inverse Monoids. In *Mathematical Foundations of Computer Science 2011*, Filip Murlak and Piotr Sankowski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 448–459.

[21] Markus Lohrey. 2012. Algorithmics on SLP-compressed strings: A survey. *Groups - Complexity - Cryptology* 4, 2 (2012), 241–299. https://doi.org/doi:10.1515/gcc-2012-0016

[22] Markus Lohrey, Sebastian Maneth, and Carl Philipp Reh. 2016. Traversing Grammar-Compressed Trees with Constant Delay. *2016 Data Compression Conference (DCC)* (2016), 546–555.

[23] Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. 2010. Storage and Retrieval of Highly Repetitive Sequence Collections. *Journal of computational biology : a journal of computational molecular cell biology* 17 (03 2010), 281–308. https://doi.org/10.1089/cmb.2009.0169

[24] Gonzalo Navarro. 2016. *Compact Data Structures – A practical approach*. Cambridge University Press. ISBN 978-1-107-15238-0. 536 pages.

[25] Gonzalo Navarro. 2019. Document listing on repetitive collections with guaranteed performance. *Theoretical Computer Science* 772 (2019), 58–72. https://doi.org/10.1016/j.tcs.2018.11.022

[26] Gonzalo Navarro. 2020. Contextual Pattern Matching. In *String Processing and Information Retrieval*, Christina Boucher and Sharma V. Thankachan (Eds.). Springer International Publishing, Cham, 3–10.

[27] Gonzalo Navarro. 2021. Indexing Highly Repetitive String Collections, Part I: Repetitiveness Measures. *ACM Comput. Surv.* 54, 2, Article 29 (March 2021), 31 pages. https://doi.org/10.1145/3434399

[28] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. 2016. Fully Dynamic Data Structure for LCE Queries in Compressed Space. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier (Eds.), Vol. 58. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 72:1–72:15. https://doi.org/10.4230/LIPIcs.MFCS.2016.72

[29] Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam Smith. 2007. Sublinear Algorithms for Approximating String Compressibility. *CoRR* abs/0706.1084 (06 2007). https://doi.org/10.1007/s00453-012-9618-6

[30] Wojciech Rytter. 2003. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science* 302 (06 2003), 211–222. https://doi.org/10.1016/S0304-3975(02)00777-6

[31] Kunihiko Sadakane. 2006. Compressed Suffix Trees with Full Functionality. *Theory of Computing Systems* 41 (2006), 589–607.

[32] Kensuke Sakai, Tatsuya Ohno, Keisuke Goto, Yoshimasa Takabatake, I. Tomohiro, and Hiroshi Sakamoto. 2019. RePair in Compressed Space and Time. In *2019 Data Compression Conference (DCC)*. 518–527. https://doi.org/10.1109/DCC.2019.00060

[33] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[34] James A. Storer and Thomas G. Szymanski. 1982. Data Compression via Textual Substitution. *J. ACM* 29, 4 (Oct. 1982), 928–951. https://doi.org/10.1145/322344.322346

[35] Takeshi Takeda, Kenji Hashimoto, and Hiroyuki Seki. 2019. Graph Compression by Tree Grammars and Direct Evaluation of Regular Path Query. 257–262. https://doi.org/10.1109/CCOMS.2019.8821730