ADAPTIVE AUTOMATA FOR GRAMMAR BASED TEXT COMPRESSION

Newton Kiyotaka Miura[1] and João José Neto[1]

[1]Escola Politécnica da Universidade de São Paulo, São Paulo, Brazil

ABSTRACT

The Internet and the ubiquitous presence of computing devices anywhere is generating a continuously growing amount of information. However, the information entropy is not uniform. It allows the use of data compression algorithms to reduce the demand for more powerful processors and larger data storage equipment. This paper presents an adaptive rule-driven device - the adaptive automata - as the device to identify repetitive patterns to be compressed in a grammar based lossless data compression scheme.

KEYWORDS

Adaptive Automata, Grammar Based Data Compression

1. INTRODUCTION

The amount of data processed by the computers has grown by the increase in hardware computing power and the data storage capacity. New applications are built to take advantage of this, reinforcing the need for more capacity in data processing. Currently, analysis of data generated by social media Internet applications and genome database processing are examples of applications the requires handling of huge amount of data. In this scenario, the necessity for optimizing the use of the finite amount of data storage available in computers is economically justifiable.

Grammar-based text compression is a method for representing a sequence of symbols using a context-free grammar (CFG) as defined in Chomsky's hierarchy [1], which generates a single sequence identical to the original one. It is based on the idea that a CFG can compactly represent the repetitive structures within a text. Intuitively, greater compression would be obtained for input strings that contain a greater number of repeated substrings that will consequently be represented by the same grammatical production rule. Examples of data with these characteristics are the sequences of genes of the same species, texts with version control systems.

The naturally hierarchical definition of a CFG allows string-manipulation algorithms to perform operations directly on their compressed representations, without the need for a prior decompression [2] [3] [4] [5]. It potentially brings the advantage of decreasing the temporary storage space required for data manipulation, in addition to opening the possibility of the algorithms present shorter execution times by decreasing the data to be processed [2]. These features are attractive for improving the efficiency in processing large amounts of data whose demand has grown mainly in Big Data applications and manipulation of genomes.

Operations in grammatically compressed texts [6] include string search, edit distance calculation, string or character repeating frequency calculation, access to a specific position of the original string, obtaining the first occurrence of a character and indexing for random access. Examples of

Dhinaharan Nagamalai et al. (Eds) : CoSIT, SIGL, AIAPP, CYBI, CRIS, SEC, DMA - 2017
pp. 173– 183, 2017. © CS & IT-CSCP 2017 DOI : 10.5121/csit.2017.70417

grammar-based compression applications such as repetitive structure mining, pattern recognition, data mining using are cited in [7], [3] and [2].

The process of obtaining this type of grammar with specific characteristics is a grammatical inference process, which is studied within the computer sciences since the sixties [8].

This work focuses on adopting an adaptive device guided by rules [9] for the identification of repetitive data for grammar-based text compression.

An adaptive device [9] has the ability of self-modification, that is, it changes the rules of its operation at execution time without the need for external intervention. An example is the adaptive automaton, which is a state machine with computational power equivalent to the Turing machine [10]. It might change its configuration based on the input string.

In the Section 2 the basic concept of this type of compression and some algorithms found in the literature is presented. In the Section 3 an adaptive automaton-based implementation of the repetitive pattern searching is presented.

2. BACKGROUND

2.1. Grammar based compression

Text compression is performed by a CFG $G = (\Sigma, V, D, X_s)$, where Σ is the finite set of terminal symbols. V is the set of nonterminal (or variable) symbols with $\Sigma \cap V = \emptyset$. $D \subset V \times (V \cup \Sigma)^*$ is the finite set of rules of production with size $n = | V |$. $X_s \in V$ is the non-terminal that represents the initial nonterminal symbol of the grammar.

The syntax tree of G is an ordered binary tree in which the inner nodes are labelled with the non-terminal symbols of V and the leaves with the terminals of Σ, that is, the sequence of labels in the sheets Corresponds to the input string S.

Each internal node Z corresponds to a production rule $Z \rightarrow XY$ with the child nodes X on the right and Y on the left. As G can be expressed in the normal form of Chomsky [1] any compression grammar is a Straight Line Program (SLP) [11] [12] [2] which is defined as a grammatical compression on $\Sigma \cup V$ and production rules in the form $X_k \rightarrow X_i X_j$ where $X_k, X_i, X_j \in \Sigma \cup V$ and $1 \leq I, j < k \leq n + \sigma$.

The compression of a string S of the size $\sigma = | S |$ is achieved when the σ is greater than the sum of the size of the grammar G that generates it and the size of the compressed sequence.

Figure 1 presents an example of the compression of the string $s = (a, b, a, a, b, c, a, b, a, a)$, resulting in the compressed sequence $S = \{X_4, c, X_3\}$ and the derivation dictionary $D = \{X_1 \rightarrow ab, X_2 \rightarrow aa, X_3 \rightarrow X_1 X_2, X_4 \rightarrow X_3 X_1\}$, corresponding to the forest of syntactic trees. The dashed lines of the same colour identify portions of the tree that have parent nodes with the same label.

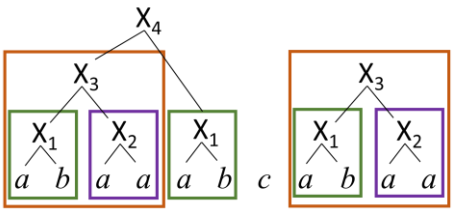

Figure 1 Example of grammar based compression.

The main challenge of grammar-based compression is to find the smallest CFG that generates the original string to maximize the compression rate. This problem has been shown to be intractable. Storer and Szymanski demonstrated [7] that given a string S and a constant k, obtaining a CFG of size k that generates S is an NP-complete problem. Furthermore, Charikar et al. [13] demonstrated that the minimum CFG can be approximated by a logarithmic rate calculating that *8569/8568* is the limit of this approximation rate of the algorithms to obtain the lowest value of k, assuming that $P \neq NP$.

Much research effort has focused in finding approximate minimal grammar inference algorithms, considering not only the compression but also searching grammars and data structures to represent them with characteristics suitable to support operations in the compressed text [2] [3] [4] [5]. Regarding the dictionary representation, as discussed by [3] several initial algorithms have adopted Huffman coding to compact it although it does not allow random access of the strings, and more recently the succinct data structure method has been used. A brief description of some researches are presented below.

2.2. Compression algorithms

In the descriptions below N is the size of the input string, g is the minimum CFG size, and $\log$ refers to $\log_2$.

The Sequitur algorithm proposed by Nevill-Manning and Witten [14] operates incrementally in relation to the input string with the restriction that each bigram is present only once in a derivation rule of the inferred grammar and that each rule is used more than once. Sequitur operates in a linear space and execution time relative to the size of the input string.

The RePair algorithm developed by Larsson and Moffat [15] constructs a grammar by iteratively replacing pairs of symbols, either terminal or non-terminal, with a non-terminal symbol by doing an off-line processing of the complete text, or long phrases, and adopting a compact representation of the dictionary. It has the following simple heuristic to process a sequence S:

1. Identify the most frequent pair ab in S.
2. Add a rule $X \rightarrow ab$ to the dictionary of productions, where X is a new symbol that is not present in S.
3. Replace every occurrence of the pair ab in S by the new symbol X.
4. Repeat this procedure until any pair in S occurs just once.

Although it requires a memory space above 10 times the size of the input string, the algorithm presents a linear execution time, being efficient mainly in the decompression, facilitating search operations in the compressed data.

Rytter [16] and Charikar et al. [13] have developed algorithms that approximate the size of the CFG obtained in $O(\log N/g)$ by transforming the representation of the data with LZ77 method [17] to the CFG. Rytter [16] proposed the use of a grammar whose derivation tree is a balanced AVL binary tree in which the height of two daughter sub-trees differ only by one unit, which favours pattern matching operations. Charikar et al. [13] imposed the condition that the binary derivation tree be balanced in width, favouring operations such as union and division.

Sakamoto [18] adopted a strategy similar to RePair [15] and obtained an algorithm requiring a smaller memory space, performing iterative substitution of pairs of distinct symbols and repetitions of the same symbol, using double-linked lists for storing the input string and a priority queue for the frequency of the pairs.

Jez [19] modified the algorithm of [18] obtaining a CFG of size $O(g\log(N/g))$ for input strings with alphabet Σ that can be identified by numbers of $\{1,\ldots,N^c\}$ for a constant c. Unlike the previous research, it was not based on Lempel-Ziv representation.

Maruyama et al. [5] developed an algorithm based on a context-dependent grammar subclass *Σ-sensitive* proposed to optimize the pattern matching operation on the compacted data.

Tabei et al. [4] has devised a scalable algorithm for least-squares partial regression based on a grammar-packed representation of high-dimensional matrices that allows quick access to rows and columns without the need for decompression. Compared to probabilistic techniques, this approach showed superiority in terms of precision, computational efficiency and interpretability of the relationship between data and tag variables and responses.

The text compression is also a focus of research such as the fully-online compression algorithm (FOLCA) proposed by Maruyama et al. [20] which infers partial parsing trees [16] whose inner nodes are traversed post-order and stored in a concise representation. For class $C = \{x_1, x_2, \ldots, x_n\}$ of n objects, log_n is the minimum of bits to represent any $x_i \in C$. If the representation method requires $n + (n)$ bits for any $x_i \in C$, the representation is called succinct [3]. They present experimental results proving the scalability of the algorithm in memory space and execution time in processing human genomes with high number of repetitive texts with presence of noise.

Fukunaga et al. [11] proposed an online algorithm approach to approximate frequent patterns of grammatically compressed data with less memory consumption compared to offline methods. *Edit-sensitive parsing* [21], which measures the similarity of two symbol strings by the edit distance, is used for comparison of grammars subtree.

2.3. Grammar inference using adaptive technology

An adaptive rule driven device has the self-modifying capability [9], that is, it changes the rules of its operation according to the input data at run time without the need for external intervention. An example is the adaptive automaton [22] which consists of a traditional automaton with the addition of an adaptive mechanism. This mechanism allows modifications in the configuration of the underlying traditional automaton by invoking adaptive functions which can change its set of rules, enabling the adaptive automaton to have a computational power equivalent to the Turing machine [10].

Grammar-based compression is a specific case of grammatical inference whose purpose is to learn grammar from information available in a language [8]. In this case the available information corresponds to the text to be compressed which is the only sentence of a given language. José Neto and Iwai [23] proposed the use of adaptive automaton to build a recognizer with ability to learn a regular language from the processing of positive and negative samples of strings belonging to that language. The recognizer is obtained from the agglutination of two adaptive automata. One constructs the prefix tree from the input strings and the other produces a suffix tree from the inverse sequence of the same string. Matsuno [24] implemented this algorithm based on adaptive automata, and presented an application of the Charikar algorithm [13] to obtain a CFG from samples of the language defined by this grammar.

In this paper, we applied the adaptive automaton to infer a grammar to generate a compressed version of a string.

3. GRAMMAR BASED COMPRESSION USING ADAPTIVE AUTOMATA

Our approach using adaptive automaton is inspired in the algorithm of RePair [15]. The automaton is used in the process of finding pairs of symbols to be substituted by a grammar production rule.

The adaptive automaton modifies the configuration of the traditional finite automaton, which is represented by a tuple (Q, Σ, P, q_0, F). The meaning of the elements of the tuple, along with other elements used in this work are described in the Table 1 for quick reference. We adopted a notation used by Cereda et al. [25].

Modification in the underlying automaton occurs through adaptive functions to be executed either before or after the transitions, according to the consumed input string symbols. The adaptive function executed before the transition is represented by the character '·' written after the function name (e.g. $\mathcal{A}\cdot$) and the function executed after the transition is represented by the same character written before the name (e.g. $\cdot\mathcal{B}$). They can modify the automaton configuration by performing elementary adaptive actions for searching, exclusion or insertion of rules. The adaptive functions use variables and generators to perform editing actions in the automaton. Variables are filled only once in the execution of the adaptive function. Generators are special types of variables, used to associate unambiguous names for each new state created in the automaton and are identified by the '*' symbol, for example, g_1^*, g_2^*.

Table 1. List of elements

Element	Meaning
Q	set of states
$F \subset Q$	subset of accepting states
$q_0 \in Q$	initial state
Σ	input alphabet
D	set of adaptive functions
P	P: $D \cup \{\varepsilon\} \times Q \times \Sigma \mapsto Q \times \Sigma \cup \{\varepsilon\} \times D \cup \{\varepsilon\}$, mapping relation
$\sigma \in \Sigma$	any symbol of the alphabet
$\mathcal{A}(q, x)$	adaptive function $\mathcal{A} \in D$ with arguments q, x triggered before a symbol consumption
$\mathcal{B}(y, z)$	adaptive function $\mathcal{B} \in D$ with arguments y, z triggered after the symbol consumption
g_i^*	generator used in adaptive functions that associates names with newly created states
$-(q_i, \sigma) \rightarrow (q_j)$	elementary adaptative action that removes the transition from q_i to q_j and consumes σ
$+(q_i, \sigma) \rightarrow (g_i^*, \varepsilon), \mathcal{A}$	rule-inserting elementary adaptive action that adds a transition from state q_i, consuming the symbol σ, and leading to a newly created state g_1^* with the adaptive function $\mathcal{A}$ to be executed before the consumption of σ
Out_i	semantic action to be performed in state q_i, as in the Moore machine [1]

The adaptive automaton presented in this paper analyses trigrams contained in the original input string to search the most appropriate pair of symbols to be to be replaced by a grammar

production rule as in an iteration of the RePair [15] algorithm. From the initial state to any final state it will have a maximum of 3 transitions, as it analyses only 3 symbols. Thus, for each trigram the automaton restarts its operation from the initial state q_0. This requires obtaining the set of all the trigrams present in the input string. For example, considering the sequence $abcab$, the input will be the set of trigrams $\{abc, bca, cab\}$.

The automaton counts the occurrence of every trigram and its containing prefix and suffix bigrams. This is accomplished by counters that are incremented by the semantic action functions Out_i executed in the states in which a bigram or trigram is recognized. Considering the trigram abc, it counts the occurrence of the trigram itself and the occurrence of ab and bc as a prefix and suffix.

Each bigram will have 2 counters, one for prefix and one for suffix, which can be incremented in any state of the automaton that has the associated semantic action function. Based on these counters, after processing all the trigrams it will be possible to choose the pair to be replaced. The pair that occurs most frequently inside the most frequent trigram, or the prefix or suffix bigram of this trigram are examples of criteria for the pair selection.

Starting from the terminal state of the automaton in which the most frequent trigram is recognized, it is possible to identify the constituent bigrams and get the values of their counters by traversing the automaton in the opposite direction of the transactions, that is, towards the starting state. In the bigram counter associated with a terminal state, it is obtained the total count of occurrences of the suffix bigram, and in the previous state the count of the prefix bigram.

The use of adaptive technique guides the design of this automaton by considering how it should be incrementally build as the input symbols are consumed, performing the actions just presented above.

In the following lines, we describe the processing of the trigram $\sigma_0\sigma_1\sigma_2$ of a hypothetical input string to better illustrate the configuration evolution of the automaton.

Figure 2 shows a simplified version of the starting configuration of the automaton composed by a single state q_0 and transitions for each symbol $\sigma \in \Sigma$ by executing the adaptive function $\mathcal{A}_0(\sigma, q_0)$ before its consumption. For ease of visualization, the representation of the set of these transitions has been replaced by a single arc in which the symbol to be consumed is indicated as $\forall\sigma$. This same simplification of representation was adopted in the description of the algorithm 1 (Figure 3) of the adaptive function $\mathcal{A}_0$.

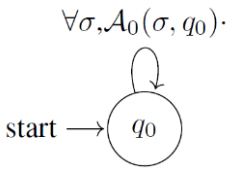

Figure 2. Initial topology of the adaptive automaton

Function $\mathcal{A}_0$ modifies the automaton by removing the transition that consumes the first symbol σ_0 of the trigram, and creating three new elements: the state q_1, the transition from q_0 to q_1 to allow the consumption of σ_0 and the loop transition from q_1 associating it with the consumption of $\forall\sigma \in \Sigma$ and another adaptive function $\mathcal{A}_1$.

Algorithm 1: Adaptive function $\mathcal{A}_0$

adaptive function $\mathcal{A}_0(s,\ q_x)$
Generators: g_1^*
$-(q_x, s) \rightarrow q_x$
$+(q_x, s) \rightarrow (g_1^*, \epsilon)$
$+(g_1^*, \forall\sigma) \rightarrow (g_1^*, \epsilon), \mathcal{A}_1$
end

Figure 3. Algorithm 1: Adaptive function $\mathcal{A}_0$

Figure 4 shows the new adaptive automaton topology after consumption of the first symbol σ_0.

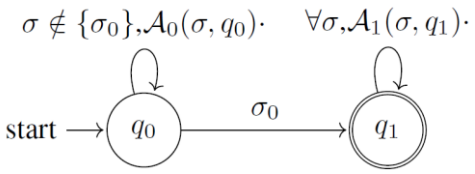

Figure 4. Topology after consuming the first symbol σ_0

The algorithm 2 (Figure 5) presents the adaptive function $\mathcal{A}_1$. Its operation is similar to $\mathcal{A}_0$ creating a new state q_2. It also prepares the consumption of a third symbol by inserting a transition with the adaptive function $\mathcal{A}_2$, which is described in the algorithm 3 (Figure 6).

Algorithm 2: Adaptive function $\mathcal{A}_1$

adaptive function $\mathcal{A}_1(s,\ q_x)$
Generators: g_1^*
$-(q_x, s) \rightarrow q_x$
$+(q_x, s) \rightarrow (g_1^*, \epsilon),$
$+(g_1^*, \sigma) \rightarrow (g_1^*, \epsilon), \mathcal{A}_2$
end

Figure 5. Algorithm 2: Adaptive function $\mathcal{A}_1$

Function $\mathcal{A}_2$ (Figure 6) modifies the automaton configuration by removing the transition from q_2 to itself by consuming the symbol σ_3 (the third symbol of the trigram) creating a new state q_3 and the transition to it consuming σ_3. The newly created state q_3 will be associated with the semantic function Out_1.

Algorithm 3: Adaptive function $\mathcal{A}_2$

adaptive function $\mathcal{A}_2(s,\ q_x)$
Generators: g_1^*
$-(q_x, s) \rightarrow q_x$
$+(q_x, s) \rightarrow (g_1^*, \epsilon)$
end

Figure 6. Algorithm 3: Adaptive function $\mathcal{A}_2$

Figure 7 shows the automaton topology after consumption of the second and third symbol of the input string.

States q_2 and q_3 are associated with output functions Out_0 and Out_1 respectively. They are related to semantic actions in these states.

Out_0 is the function responsible for incrementing the occurrence counter of the prefix bigram $\sigma_0\sigma_1$. Out_1 is responsible for incrementing the occurrence counter of the suffix bigram $\sigma_1\sigma_2$, and the counter of the trigram $\sigma_0\sigma_1\sigma_2$.

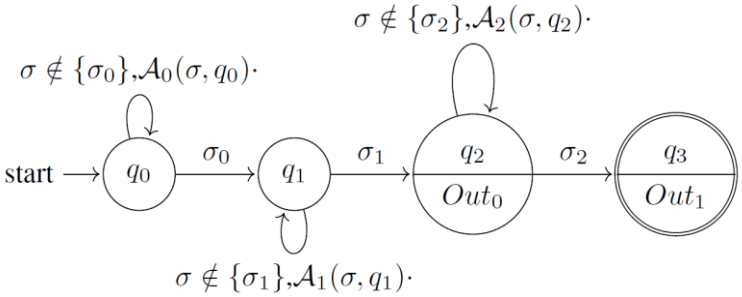

Figure 7. Topology after consuming the initial trigram $\sigma_0\sigma_1\sigma_2$

To better illustrate the operation of the adaptive automaton, Figure 8 shows its configuration after processing the sample string *abcaba*. The index *i* of the states q_i corresponds to the sequence in which they were entered by the algorithm.

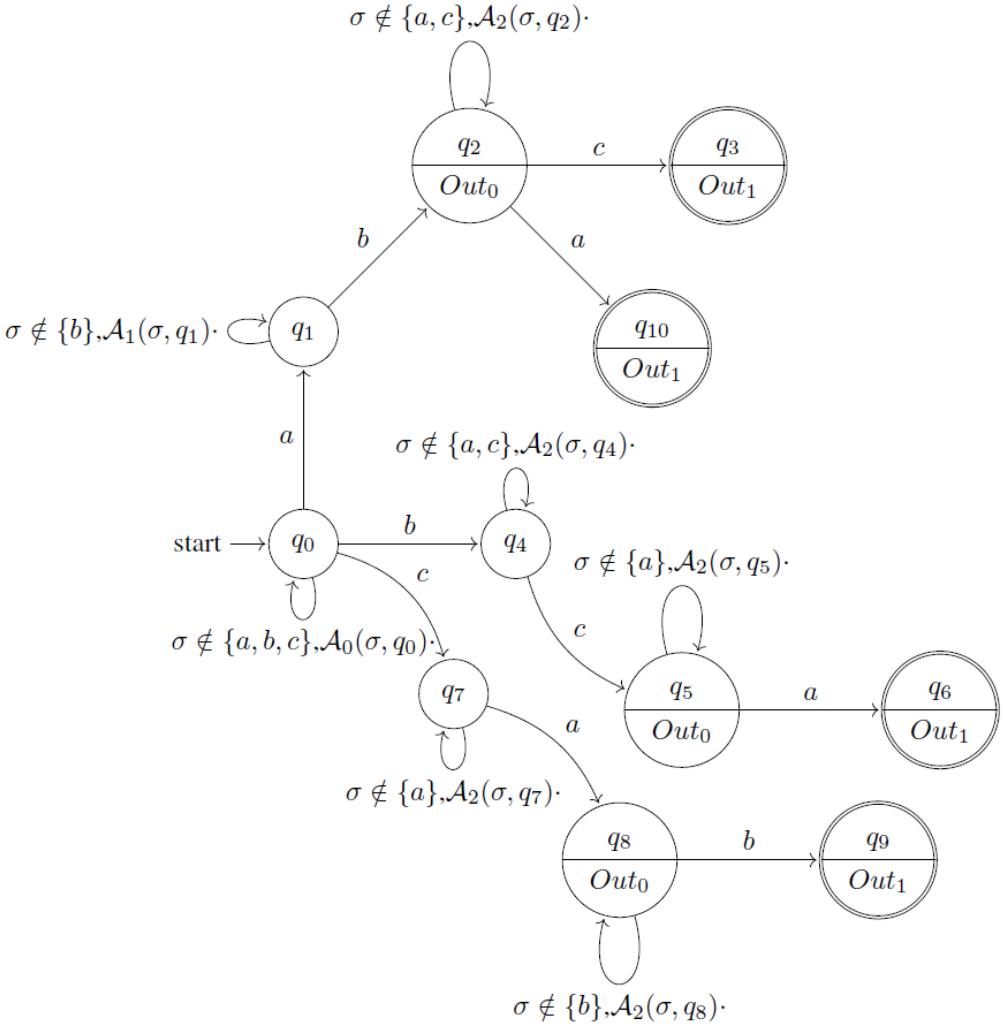

Figure 8. Automaton topology after processing *abcaba*

3. EXPERIMENTS

A test system is being developed using a Java language library for adaptive automaton[1] proposed by [26] as the core engine, with the surrounding subsystems such as the semantic functions and the iterations that substitutes the frequently occurring patterns.

Publicly available corpora[2] will be used as the input data.

Compression rates, execution time and temporary storage requirements for the different criteria of choice of repetitive patterns that the algorithm allows will be measured and compared to other techniques.

4. CONCLUSION

In this work, we presented the adaptive automaton as a device to identify a bigram, prefix or suffix, which are most repeated within the most repeated trigram in a sequence of symbols to obtain a substitution rule in a compression algorithm based on grammar.

In this paper, we presented the adaptive automaton as the device to identify repetitive bigrams in a sequence of symbols considering its presence inside frequently occurring trigrams. This bigram can be specified to be whether prefix or suffix of the trigram allowing the adoption of different configurations to be experimented.

As a future work, the adaptive automaton can be further expanded to analyse n-grams larger than trigrams. In addition, a comparative performance study could be done with other techniques. Another point to be analysed is the adoption of an adaptive grammatical formalism [27] in the description of the inferred grammar with the aim of making some operation in the compressed data.

Adaptive rule-driven device allows the construction of a large and complex system by simplifying the representation of the problem. This type of automaton is designed by specifying how the device must be incrementally modified in response to the input data, from a simple initial configuration, aiming at its desired intermediate configurations, and the output to be obtained by associated semantic actions.

ACKNOWLEDGEMENT

The first author is supported by Olos Tecnologia e Sistemas.

REFERENCES

[1] Sipser, M. (2006) “Introduction to the theory of computation”, Thomson Course Technology.

[2] Lohrey, M. (2012) “Algorithmics on SLP-compressed strings: A survey.” Groups Complexity Cryptology, vol. 4, no. 2, pp. 241–299.

[3] Sakamoto, H. (2014) “Grammar compression: Grammatical inference by compression and its application to real data”, in Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, September 17-19, 2014., pp. 3–20.

[4] Tabei,Y., Saigo, H., Yamanishi, Y. & Puglisi, S. J. (2016) “Scalable partial least squares regression on grammar-compressed data matrices”, in 22nd KDD, pp. 1875—1884.

[1] https://github.com/cereda/aa

[2] http://pizzachili.dcc.uchile.cl/repcorpus.html

[5] Maruyama, S., Tanaka, Y., Sakamoto, H., & Takeda, M. (2010) "Context-sensitive grammar transform: Compression and pattern matching", IEICE Transactions on Information and Systems, vol. E93.D, no. 2, pp. 219–226.

[6] Tabei, Y. (2016) "Recent development of grammar compression", Information Processing Society of Japan Magazine, vol. 57, no. 2, pp. 172–178.

[7] Jez, A. (2016) "A really simple approximation of smallest grammar", Theoretical Computer Science, vol. 616, pp. 141 – 150.

[8] De la Higuera, C. (2010) Grammatical inference: learning automata and grammars. Cambridge University Press.

[9] José Neto, J. (2001) "Adaptive rule-driven devices - general formulation and case study", in CIAA 2001 6th International Conference on Implementation and Application of Automata, ser. Lecture Notes in Computer Science, B. W. Watson and D. Wood, Eds., vol. 2494. Pretoria, South Africa: Springer-Verlag, pp. 234–250.

[10] Rocha, R. L. A. & José Neto, J. (2000) "Adaptive automaton, limits and complexity compared to the Turing machine", in Proceedings of the I LAPTEC, pp. 33–48.

[11] Fukunaga, S., Takabatake, Y., I, T. & Sakamoto, H. (2016) "Online grammar compression for frequent pattern discovery", CoRR, vol. abs/1607.04446.

[12] Takabatake, Y., Tabei, Y., & Sakamoto, H. (2015) Online Self-Indexed Grammar Compression. Springer International Publishing, pp. 258–269.

[13] Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A. & Shelat, A. (2005) "The smallest grammar problem." IEEE Trans. Inf. Theory, vol. 51, no. 7, pp. 2554–2576.

[14] Nevill-Manning, C. G. & Witten, I. H. (1997) "Identifying hierarchical structure in sequences: A linear-time algorithm", J. Artif. Intell. Res.(JAIR) vol. 7, pp. 67–82.

[15] Larsson, N. J. & Moffat, A. (1999) "Offline dictionary-based compression", in Data Compression Conference, 1999. Proceedings. DCC '99, pp. 296–305.

[16] Rytter, W. (2002) Application of Factorization to the Approximation of Grammar-Based Compression. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 20–31.

[17] Ziv, J. & Lempel, A. (2006) "A universal algorithm for sequential data compression", IEEE Trans. Inf. Theor., vol. 23, no. 3, pp. 337–343.

[18] Sakamoto, H., (2005) "A fully linear-time approximation algorithm for grammar-based compression", Journal of Discrete Algorithms, vol. 3, no. 2–4, pp. 416–430.

[19] Jez, A. (2015) "Approximation of grammar-based compression via recompression", Theoretical Computer Science, vol. 592, pp. 115–134.

[20] Maruyama, S. & Tabei, Y. (2014) "Fully online grammar compression in constant space." in DCC, Bilgin, A., Marcellin, M. W., Serra-Sagristà, J. & Storer, J. A. Eds. IEEE, pp. 173–182.

[21] Cormode, G. & Muthukrishnan, S. (2007) "The string edit distance matching problem with moves", ACM Transactions on Algorithms (TALG) vol. 3, no. 1, pp. 2:1–2:19.

[22] José Neto, J. (1994) "Adaptive automata for context-sensitive languages", SIGPLAN Notices, vol. 29, no. 9, pp. 115–124.

[23] José Neto, J. & Iwai, M. K. (1998) "Adaptive automata for syntax learning" in Anais da XXIV Conferência Latinoamericana de Informática - CLEI 98, pp. 135–149.

[24] Matsuno, I. P. (2006) "Um estudo dos processos de inferência de gramáticas regulares e livres de contexto baseados em modelos adaptativos", Master's Thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil.

[25] Cereda, P. R. M. & José Neto, J. (2015) "A recommendation engine based on adaptive automata", in Proceedings of the 17th International Conference on Enterprise Information Systems - Volume 2: ICEIS, pp. 594–601.

[26] Cereda, P. R. M. & José Neto, J. (2016) "AA4J: uma biblioteca para implementação de autômatos adaptativos" in Memórias do X Workshop de Tecnologia Adaptativa – WTA 2016, 2016, pp. 16–26.

[27] Iwai, M. K. (2000) "Um formalismo gramatical adaptativo para linguagens dependentes de contexto", PhD Thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil.

AUTHORS

Newton Kiyotaka Miura is a researcher at Olos Tecnologia e Sistemas and a PhD candidate in Computer Engineering at Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da Universidade de São Paulo. He received his Electrical Engineering degree from Escola Politécnica da Universidade de São Paulo (1989) and holds a master's degree in Systems Engineering from the University of Kobe, Japan (1993). His research interests include adaptive technology, adaptive automata, adaptive devices and natural language processing.

João José Neto is an associate professor at Escola Politécnica da Universidade de São Paulo and coordinates the Language and Adaptive Technology Laboratory of Departamento de Engenharia de Computação e Sistemas Digitais. He received his Electrical Engineering degree (1971), master's degree (1975) and PhD in Electrical Engineering (1980) from Escola Politécnica da Universidade de São Paulo. His research interests include adaptive devices, adaptive technology, adaptive automata and applications in adaptive decision making systems, natural language processing, compilers, robotics, computer assisted teaching, intelligent systems modelling, automatic learning processes and adaptive technology inferences.