Estudo comparativo de algoritmos de compressão de textos baseada em gramática

Newton K. Miura e J. José Neto

***Resumo*—A compressão baseada em gramática busca inferir a menor gramática possível para descrever sem perdas uma cadeia de símbolos. Este documento apresenta uma breve descrição de algoritmos de compressão gramatical encontrados na literatura nos quais o uso da tecnologia adaptativa será investigado para verificar potenciais vantagens trazidas por esta abordagem na simplificação da expressão de soluções, no aprendizado incremental da sintaxe de textos.**

***Palavras-chave:*—compressão baseada em gramática, inferência de gramáticas, gramáticas livres de contexto, autômatos adaptativos.**

I. Introdução

A compressão de texto baseada em gramática é um método para representar uma cadeia de caracteres por meio de uma gramática livre de contexto (GLC) definida conforme hierarquia de Chomsky [1], que gera uma única cadeia idêntica à original. Pesquisada desde a década de setenta [2], esta abordagem de compactação de dados baseia-se na ideia de que uma GLC pode representar de forma compacta as estruturas repetitivas existentes dentro de um texto. Intuitivamente, uma maior compressão seria obtida para cadeias de entrada que contenham uma maior quantidade de subcadeias repetidas e que consequentemente seriam representadas por uma mesma regra de produção gramatical. Exemplos de dados com estas características são as sequências de genes de uma mesma espécie, textos com controle de versão.

A definição naturalmente hierárquica de uma GLC permite que algoritmos de manipulação de cadeias possam realizar operações diretamente nas suas representações compactadas, sem a necessidade de uma descompressão prévia [3], [4], [5], [6]. Isso traz a vantagem de diminuir a o espaço de armazenamento temporário requerido para a manipulação de dados, além de abrir possibilidade dos algoritmos apresentarem tempos menores de execução ao se diminuir o dado a ser processado [3]. Essas características são atraentes por melhorar a eficiência no processamento de grandes quantidades de dados cuja demanda tem crescido principalmente em aplicações de *Big Data* e manipulação de genomas.

Operações em textos compactados gramaticalmente [7] incluem busca de cadeias, cálculos de distância de edição, frequência de repetição de cadeias ou caractere, cálculo do acesso a uma posição específica da cadeia original, obtenção da primeira ocorrência de um caractere e indexação para acesso randômico. Exemplos de aplicações de compressão baseada em gramática como mineração de estruturas repetitivas, reconhecimento de padrões, mineração de dados utilizando são citados em [8], [4] e [3].

Os autores podem ser contatados através dos seguintes endereços de correio eletrônico: nkmiura@usp.br e jjneto@usp.br.

O processo de obtenção deste tipo de gramática com características específicas, conforme descritas no item II-A, é um processo de inferência de gramática, que consiste num campo de estudos dentro das ciências da computação [9] desde a década de sessenta [10].

Este trabalho foca na compressão de textos baseada em gramática adotando um dispositivo adaptativo guiado por regras [11] para a identificação de dados repetitivos.

Um dispositivo adaptativo [12] tem a capacidade auto modificação, ou seja, ele altera as regras de seu funcionamento em tempo de execução sem a necessidade de intervenção externa. Um exemplo é o autômato adaptativo, que é uma máquina de estado com poder computacional equivalente à máquina de Turing [13], capaz de alterar a sua configuração de acordo com a cadeia de entrada.

Na seção II-A é apresentado o conceito básico desse tipo de compressão e alguns algoritmos encontrados na literatura. Na seção III é apresentada uma implementação baseada em autômato adaptativo.

II. Revisão da literatura

A. Compressão baseada em gramática

A compressão de texto é realizada por meio de um tipo de GLC $G = (\Sigma, V, D, X_s)$ [14] em que: Σ é o conjunto finito de símbolos terminais de tamanho $m = |\Sigma|$; V é o conjunto de símbolos não-terminais (ou variáveis) com $\Sigma \cap V = \emptyset$; $D \subset V \times (V \cup \Sigma)^*$ é o conjunto finito de regras de produção com tamanho $n = |V|$; e $X_s \in V$ é o não-terminal que representa o símbolo inicial da gramática. A compressão gramatical de uma cadeia S é uma GLC que produz somente S deterministicamente, ou seja, para qualquer $X \in V$ existe somente uma regra de produção em D e não há ciclos. A árvore sintática de G é uma árvore binária ordenada em que os nós internos são etiquetados com os símbolos não-terminais de V e as folhas com os terminais de Σ, ou seja, a sequência de etiquetas nas folhas corresponde à cadeia de entrada S. Cada nó interno Z corresponde a uma regra de produção $Z \rightarrow XY$ com os nós filhos X à direita e Y à esquerda. Como G pode ser expressa na forma normal de Chomsky [1] qualquer gramática de compressão é um *Straight Line Program* (SLP) [15], [14], [3] que é definida como uma compressão gramatical sobre $\Sigma \cup V$ e regras de produção na forma $X_k \rightarrow X_i X_j$ onde X_k, X_i, $X_j \in \Sigma \cup V$, e $1 \leq i, j < k \leq n + m$.

A figura 1 apresenta um exemplo de compactação da cadeia $S = (a, b, a, a, a, b, c, a, b, a, a)$, que resulta na sequência compactada $S^4 = \{X_4, c, X_3\}$ e o dicionário de derivações

$D = \{X_1 \rightarrow ab, X_2 \rightarrow aa, X_3 \rightarrow X_1X_2, X_4 \rightarrow X_3X_1\}$, que corresponde à floresta de árvores sintáticas. As linhas tracejadas de uma mesma cor identificam trechos da árvore que possuem nós pais com uma mesma etiqueta.

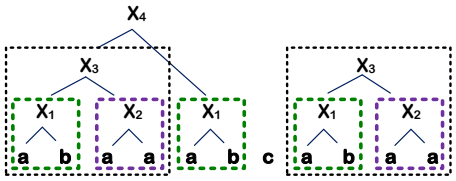

Figura 1. Exemplo de compressão baseada em gramática.

A taxa de compressão será maior quanto menor for a GLC inferida, o que constitui no principal desafio da compressão baseada em gramática, pois este problema foi demonstrado ser intratável. Storer e Szymanski [16] demonstraram que dada uma cadeia S e uma constante k a obtenção de uma GLC de tamanho k que gere S é um problema NP-completo, e Charikar et al. [17] demonstrou que a GLC mínima é aproximável a uma taxa logarítmica calculando que $8569/8568$ é o limite desta taxa de aproximação dos algoritmos para a obtenção do menor valor de k, assumindo que $P \neq NP$. Além dos algoritmos de inferência das gramáticas, outro objeto de pesquisa é a obtenção de uma estrutura de dados para armazenar a GLC inferida que tenham características adequadas para suportar operações no texto compactado [5] e sejam também compactas. Conforme analisado por [4] vários algoritmos iniciais adotaram a codificação de Huffman para compactar o dicionário embora ele não permita acesso aleatório das cadeias, e mais recentemente tem-se usado o método de representação *succint data structure*.

São apresentadas a seguir breves descrições de algumas pesquisas desenvolvidas.

B. Algoritmos de compressão

Nas descrições abaixo N é o tamanho da cadeia de entrada, g é o tamanho da gramática GLC mínima e $\log$ refere-se a $\log_2$.

O algoritmo Sequitur proposto por Nevill-Manning [18] opera de forma incremental em relação à cadeia de entrada com a restrição de que cada bigrama esteja presente somente uma única vez numa regra de derivação da gramática inferida e que cada regra seja usada mais de uma vez. Sequitur opera num espaço e tempo de execução linear em relação ao tamanho da cadeia de entrada.

O algoritmo RePair desenvolvido por Larsson e Moffat [19] constrói uma gramática substituindo iterativamente pares de símbolos, terminal ou não, por um símbolo não-terminal fazendo um processamento *off-line* do texto completo, ou de trechos longos, e adotando uma representação compacta do dicionário. Embora exija um espaço de memória acima de 10 vezes o tamanho da cadeia de entrada, o algoritmo apresenta um tempo de execução linear, sendo eficiente principalmente na descompressão, favorecendo operações de busca no dado compactado.

Rytter [20] e Charikar et al. [17] desenvolveram algoritmos que aproximam o tamanho da GLC obtida em $O(\log N/g)$ a partir da transformação para GLC da representação dos dados campactados com o método LZ77 de Lempel e Ziv [21]. Rytter propõe o uso de uma gramática cuja árvore de derivação é uma árvore binária AVL balanceada [22] em que a altura de dois sub-árvores filhas diferem somente de uma unidade, o que favorece operações de casamento de padrões. Charikar et al. impôs a condição de que a árvore de derivação binária seja balanceada em largura, favorecendo operações de união e divisão.

Sakamoto [23] adotou estratégia similar ao RePair [19] e obteve um algoritmo com exigência de um espaço em memória menor, realizando a substituição iterativa de pares de símbolos distintos e repetições de um mesmo símbolo, usando listas duplamente ligadas para armazenar a cadeia de entrada e uma fila de prioridade para a frequência dos pares.

O algoritmo de [23] foi modificado por Jez [24] que obteve uma gramática de tamanho $O(g \log(N/g))$ para cadeias de entrada com alfabeto Σ que possam ser identificados por números de $\{1, ..., N^c\}$ para uma constante c. Diferentemente das pesquisas anteriores a cadeia de entrada nas demonstrações e nas análises não foi baseada na representação Lempel-Ziv.

Maruyama et al. [6] desenvolveu um algoritmo baseado numa subclasse de gramática dependente de contexto Σ-$sensitive$ por ele proposto com o objetivo de otimizar a operação de casamento de padrão no dado compactado.

Tabei et al. [5] elaborou um algoritmo escalável para regressão parcial de mínimos quadrados baseado numa representação compactada por gramática de matrizes com alta dimensionalidade e que permite acesso rápido a linhas e colunas sem a necessidade de descompactação. Quando comparada a técnicas probabilísticas, esta abordagem mostrou superioridade em termos de precisão, eficiência computacional e facilitou a interpretação da relação entre os dados e as variáveis de etiquetas e respostas.

A compressão *online* de texto é o foco no algoritmo *Fully-online compression algorithm* (FOLCA) proposto por Maruyama et al. [25] que infere árvores de *parsing* parciais [20] cujos nós internos são percorridos *post-order* e armazenados numa representação sucinta. Para a classe $C = \{x_1, x_2, ..., x_n\}$ de n objetos, $\log n$ é o mínimo de bits para representar qualquer $x_i \in C$. Se o método de representação requer $n + O(n)$ bits para qualquer $x_i \in C$, a representação é chamada de sucinta [4]. São apresentados resultados experimentais comprovando a escalabilidade do algoritmo em espaço de memória e tempo de execução no processamento de genomas humanos com alta quantidade de textos repetitivos com presença de ruídos.

Fukunaga et al. [15] propôs um algoritmo *online* para identificação aproximada de padrões frequentes em dados compactados gramaticalmente com menor consumo de memória comparado com métodos *offline*. O *edit-sensitive parsing* (ESP) [26], que mede a similaridade de duas cadeias de símbolos pela distância de edição, é usado para a comparação de sub-árvores de gramáticas.

C. Adaptatividade em inferência gramatical

A compressão baseada em gramática é um caso específico de inferência gramatical cuja finalidade é a identificação de padrões sintáticos que permitem estabelecer as regras de formação de uma linguagem a partir de amostras [9]. Neste caso a amostra corresponde ao texto a ser compactado que é a única sentença que a gramática permite obter.

As pesquisas brevemente descritas abaixo empregaram técnicas baseadas em dispositivos adaptativos para a inferência gramatical.

Neto e Iwai [27] propuseram o seu uso para inferir um reconhecedor com capacidade de aprendizado de uma linguagem regular a partir do processamento de amostras positivas e negativas de cadeias pertencentes a essa linguagem. O reconhecedor é obtido a partir da aglutinação de dois autômatos adaptativos. Um deles constrói a árvore de prefixos a partir das cadeias de entrada e o outro produz uma árvore de sufixos a partir da sequência inversa da mesma cadeia. Matsuno [28] implementou este algoritmo baseado em autômatos adaptativos, e apresentou uma aplicação do algoritmo de Charikar [17] para obter uma GLC a partir de amostras da linguagem definida por essa gramática.

Tabela I
LISTA DE ELEMENTOS.

Elemento	Significado
Q	Conjunto dos estados do autômato finito
$F \subset Q$	Subconjunto dos estados de aceitação
$q_0 \in Q$	Estado inicial
Σ	Alfabeto da cadeia de entrada
D	Conjunto das funções adaptativas
P	$P : D \cup \{\epsilon\} \times Q \times \Sigma \mapsto Q \times \Sigma \cup \{\epsilon\} \times D \cup \{\epsilon\}$, Relação de mapeamento
$\sigma \in \Sigma$	Qualquer símbolo do alfabeto
$\mathcal{A}(q, x)$	Função adaptativa $\mathcal{A} \in D$ com argumentos q, x executado antes do consumo de um símbolo
$\mathcal{B}(y, z)$	Função adaptativa $\mathcal{B} \in D$ com argumentos y, z executado após o consumo de um símbolo
g_i^*	Gerador usado em funções adaptativas que associa nomes com novos estados criados
$-(q_i, \sigma) \rightarrow (q_j)$	Ação adaptativa elementar que remove a transição de q_i para q_j e que consome σ
$+(q_i, \sigma) \rightarrow (g_i^*, \epsilon), \mathcal{A}$	Ação adaptativa elementar que adiciona uma transição a partir do estado q_i, consumindo o símbolo σ, levando a um novo estado criado g_1^* pela função adaptativa $\mathcal{A}$ a ser executado antes do consumo do síbolo σ
Out_i	Ação semântica a ser executada em alguns estados, como na máquina de Mealy [1].

III. COMPRESSÃO BASEADA EM GRAMÁTICA COM IMPLEMENTAÇÃO ADAPTATIVA

A abordagem adaptativa adotada neste trabalho é inspirada no algoritmo RePair [19] e utiliza um autômato adaptativo no processo de identificação de padrões de símbolos repetidos na cadeia de entrada a serem substituídos por uma regra de produção de gramática.

O autômato adaptativo modifica a configuração de um autômato finito tradicional, que pode ser representado por uma tupla (Q, Σ, P, q_0, F). O significado de cada elemento desta tupla e os outros elementos utilizados nesta seção estão descritos na tabela I para referência. Foi adotada uma notação similar a apresentada por Cereda et al em [29].

A modificação no autômato subjacente ocorre por meio de funções adaptativas a serem executadas antes ou depois das transições, ou seja, de acordo com os símbolos consumidos da cadeia de entrada. A função adaptativa executada antes da transição é representada pelo caractere '$\cdot$' grafado após o nome da função (ex. $\mathcal{A}\cdot$) e função posterior pelo mesmo caractere grafado antes do nome (ex. $\cdot\mathcal{B}$). Elas podem modificar a configuração do autômato realizando ações adaptativas elementares de consulta, deleção ou inserção de regras. As funções adaptativas podem usar variáveis e geradores para as ações de alteração do autômato. As variáveis são preenchidas somente uma única vez durante a execução. Os geradores são um tipo especial de variáveis usadas para associar um nome único, sem ambiguidade, para cada estado novo criado no autômato e são identificados pelo símbolo '$*$' (ex. g_1^*, g_2^*).

O autômato adaptativo apresentado neste trabalho analisa trigramas presentes na cadeia de entrada para definir o par de símbolos mais adequado a ser substituído por uma regra de produção de gramática numa iteração do algoritmo RePair. Desta forma, a cada trigrama o autômato reinicia a sua operação a partir do estado inicial q_0. Isso requer a obtenção do conjunto de todos os trigramas presentes na cadeia de entrada. Por exemplo, considerando a sequência $abcab$, o conjuto das entradas para o autômato será $\{abc, bca, cab\}$.

O autômato totaliza a ocorrência de cada trigrama e de seus bigramas constituintes, prefixo e sufixo, da cadeia de entrada. Isso é realizado por meio de contadores que são incrementados por funções de ação semântica Out_i executados nos estados em que um bigrama ou trigrama é reconhecido. Por exemplo, para um trigrama abc, são contabilizados a quantidade total de repetição do trigrama, do prefixo ab e do sufixo bc.

Cada bigrama terá 2 contadores, um para prefixo e outro para sufixo, que podem ser incrementados em qualquer estado do autômato que tenha a função de ação semântica associada. Com base nestes contadores, após o processamento de todos os trigramas será possível escolher o par a ser substituído. Um exemplos de critério de escolha é o par com a maior quantidade de repetições dentro do trigrama mais frequente. Outros critérios pode ser a escolha do prefixo ou sufixo deste trigrama.

Partindo do estado terminal no qual o trigrama mais frequente é reconhecido, ao percorrer o autômato no sentido inverso das transações, ou seja, em direção ao estado inicial, é possível identificar os bigramas constituintes e obter os valores de seus contadores. No contador de bigrama associado ao estado terminal do trigrama é obtido o valor da contagem para o bigrama sufixo, e no estado anterior o contador do bigrama prefixo.

O uso da adaptatividade orienta o projeto deste autômato de forma a considerar como ele deve ser construído de forma incremental para que à medida que se consome os símbolos da cadeia de entrada o objetivo acima seja atingido. Do estado inicial até qualquer estado final ele terá no máximo 3 transições, pois ele analisa apenas 3 símbolos.

A seguir é apresentada a evolução das configurações deste

autômato adaptativo para o primeiro trigrama $\sigma_0\sigma_1\sigma_2$ de uma cadeia de entrada.

A figura 2 apresenta a configuração inicial simplificada do autômato composto por um único estado q_0 e transições para cada símbolo $\sigma \in \Sigma$ executando a função adaptativa $\mathcal{A}_0(\sigma, q_0)$ antes do consumo. Para facilitar a visualização, a representação do conjunto destas transições foi substituída por um único arco em que o símbolo a ser consumido está indicado como $\forall\sigma$. Esta mesma simplificação de representação foi adotada na descrição do algoritmo 1 da função adaptativa $\mathcal{A}_0$. Ela remove a transição que consome o primeiro símbolo σ_0 do trigrama, cria um novo estado q_1 e a transição para ele consumindo σ_0, e também transições de q_1 para ele mesmo, associados ao consumo de $\forall\sigma \in \Sigma$, e outra função adaptativa $\mathcal{A}_1$.

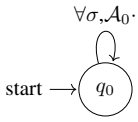

Figura 2. Configuração inicial do autômato adaptativo.

Algoritmo 1: Função adaptativa $\mathcal{A}_0$

função adaptativa $\mathcal{A}_0(s,\ q_x)$
Geradores: g_1^*
$-(q_x, s) \rightarrow q_x$
$+(q_x, s) \rightarrow (g_1^*, \epsilon)$
$+(g_1^*, \forall\sigma) \rightarrow (g_1^*, \epsilon), \mathcal{A}_1$
fim

A figura 3 apresenta a nova configuração do autômato adaptativo após o consumo do primeiro símbolo σ_0. O algoritmo 2 apresenta a função adaptativa $\mathcal{A}_1$. A sua operação é similar à de $\mathcal{A}_0$, criando um novo estado q_2, porém ela prepara o consumo de um terceiro símbolo inserindo uma transação com a função adaptativa $\mathcal{A}_2$, que está descrita no algoritmo 3.

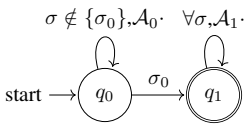

Figura 3. Configuração do autômato após o consumo do primeiro símbolo σ_0.

Algoritmo 2: Função adaptativa $\mathcal{A}_1$

função adaptativa $\mathcal{A}_1(s,\ q_x)$
Geradores: g_1^*
$-(q_x, s) \rightarrow q_x$
$+(q_x, s) \rightarrow (g_1^*, \epsilon),$
$+(g_1^*, \sigma) \rightarrow (g_1^*, \epsilon), \mathcal{A}_2$
fim

O algoritmo 3 modifica a configuração do autômato removendo a trasição de q_2 para ele mesmo consumindo o símbolo σ_3, cria um novo estado q_3 e a transição para ele consumindo σ_3, o terceiro símbolo do trigrama. O novo estado criado será associado à função semântica Out_1.

Algoritmo 3: Função adaptativa $\mathcal{A}_2$

função adaptativa $\mathcal{A}_2(s,\ q_x)$
Geradores: g_1^*
$-(q_x, s) \rightarrow q_x$
$+(q_x, s) \rightarrow (g_1^*, \epsilon)$
fim

Após o consumo do segundo e do terceiro símbolo da cadeia de entrada é obtido o autômato da figura 4.

Os estados q_2 e q_3 são associados a funções de saída Out_0 and Out_1 respectivamente correspondentes a ações semânticas nestes estados.

Out_0 é a função responsável por incrementar o contador de ocorrências do bigrama prefixo $\sigma_0\sigma_1$. Out_1 é responsável por incrementar o contador de ocorrências do bigrama sufixo $\sigma_1\sigma_2$, e o contador do trigrama $\sigma_0\sigma_1\sigma_2$.

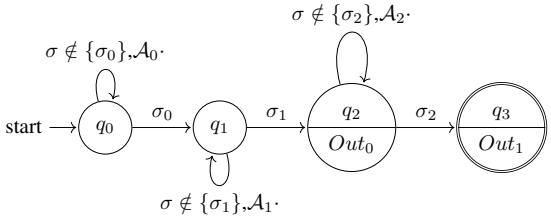

Figura 4. Configuração do autômato após o consumo do trigrama inicial $\sigma_0\sigma_1\sigma_2$.

Para melhor ilustrar o funcionamento do autômato adaptativo, a figura 5 apresenta a sua configuração após o processamento da cadeia exemplo $abcaba$. O índice i dos estados q_i corresponde à sequência em que foram inseridos pelo algoritmo.

IV. Experimentos

Está em elaboração um sistema de testes que implementa o autômato adaptativo utilizando a biblioteca AA4J [1] [30], as funções de saída correspondentes às ações semânticas e os processos iterativos de substituição de pares por regras de produção gramatical, de forma similar ao algoritmo RePair.

Para os ensaios serão utilizados corpora publicamente disponíveis [2].

Nos ensaios serão mensurados e comparados as taxas de compressão, tempo de execução e requisitos de armazenamento temporário para os diferentes critérios de escolha de padrões repetitivos que o algoritmo permite.

V. Conclusão

Para obter uma regra de substituição num algoritmo de compressão baseada em gramática, neste trabalho apresentamos o autômato adaptativo como dispositivo para identificar

[1] https://github.com/cereda/aa
[2] http://pizzachili.dcc.uchile.cl/repcorpus.html

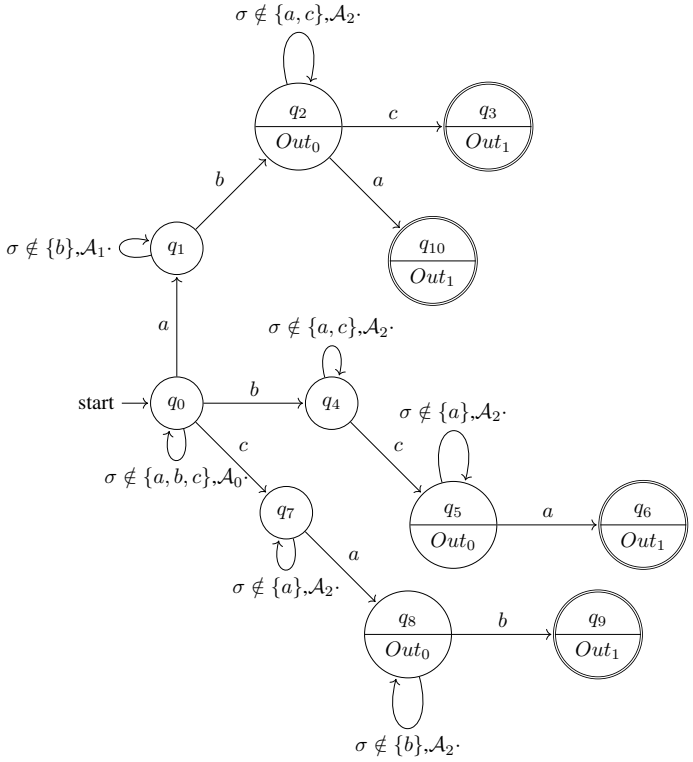

Figura 5. Configuração do autômato após o processamento da cadeia $abcaba$.

bigramas, prefixo ou sufixo, que mais se repetem numa cadeia de símbolos e que também ocorrem dentro de um trigrama mais repetido.

Como trabalho futuro, o autômato adaptativo poderá ser expandido para analisar n-gramas maiores que trigramas. Além disso, um estudo comparativo de desempenho poderá ser feito com outras técnicas. Outro ponto a ser analisado é a adoção de um formalismo gramatical adaptativo [31] na descrição da gramática inferida com o objetivo de viabilizar alguma operação diretamente no dado compactado.

A tecnologia adaptativa possibilita a construção de autômato de grandes dimensões e complexos por meio da simplificação da representação do problema. Este tipo de autômato é projetado por meio da especificação da forma como o dispositivo deve ser incrementalmente modificado de acordo com os dados de entrada, a partir de uma configuração inicial simples, visando a suas configurações intermediárias desejadas, e também a saída a ser obtida pelas ações semânticas associadas.

REFERÊNCIAS

[1] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997.

[2] F. Rubin, "Experiments in text file compression," *Commun. ACM*, vol. 19, no. 11, pp. 617–623, Nov. 1976. [Online]. Available: http://doi.acm.org/10.1145/360363.360368

[3] M. Lohrey, "Algorithmics on slp-compressed strings: A survey." *Groups Complexity Cryptology*, vol. 4, no. 2, pp. 241–299, 2012. [Online]. Available: http://dx.doi.org/10.1515/gcc-2012-0016

[4] H. Sakamoto, "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.*, 2014, pp. 3–20. [Online]. Available: http://jmlr.org/proceedings/papers/v34/sakamoto14a.html

[5] Y. Tabei, H. Saigo, Y. Yamanishi, and S. J. Puglisi, "Scalable partial least squares regression on grammar-compressed data matrices," in *22nd KDD*, 2016, pp. 1875—-1884.

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

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

[8] A. Jeż, *A really Simple Approximation of Smallest Grammar*. Springer International Publishing, 2014, pp. 182–191.

[9] C. De la Higuera, *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[10] M. E. Gold, "Language identification in the limit," *Information and Control*, vol. 10, no. 5, pp. 447–474, 1967.

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

[12] ——, "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, África do Sul: Springer-Verlag, Julho 2001, pp. 234–250.

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

[14] Y. Takabatake, Y. Tabei, and H. Sakamoto, *Online Self-Indexed Grammar Compression*. Springer International Publishing, 2015, pp. 258–269.

[15] S. Fukunaga, Y. Takabatake, T. I, and H. Sakamoto, "Online grammar compression for frequent pattern discovery," *CoRR*, vol. abs/1607.04446, 2016.

[16] J. A. Storer and T. G. Szymanski, "The macro model for data compression," in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, ser. STOC '78. New York, NY, USA: ACM, 1978, pp. 30–39. [Online]. Available: http://doi.acm.org/10.1145/800133.804329

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

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

[19] N. J. Larsson and A. Moffat, "Offline dictionary-based compression," in *Data Compression Conference, 1999. Proceedings. DCC '99*, Mar 1999, pp. 296–305.

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

[21] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theor.*, vol. 23, no. 3, pp. 337–343, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1977.1055714

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[23] H. Sakamoto, "A fully linear-time approximation algorithm for grammar-based compression," *Journal of Discrete Algorithms*, vol. 3, no. 2–4, pp. 416–430, 2005, combinatorial Pattern Matching (CPM) Special IssueThe 14th annual Symposium on combinatorial Pattern Matching. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570866704000632

[24] A. Jez, "Approximation of grammar-based compression via recompression," *Theoretical Computer Science*, vol. 592, pp. 115–134, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397515004685

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

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

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

[28] I. P. Matsuno, "Um estudo dos processos de inferência de gramáticas regulares e livres de contexto baseados em modelos adaptativos," Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil, 2006.

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

[30] ——, "AA4J: uma biblioteca para implementação de autômatos adaptativos," in *Memórias do X Workshop de Tecnologia Adaptativa – WTA 2016*, 2016, pp. 16–26.

[31] M. K. Iwai, "Um formalismo gramatical adaptativo para linguagens dependentes de contexto," Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil, 2000.

Newton Kiyotaka Miura é graduado em Engenharia de Eletricidade pela Escola Politécnica da Universidade de São Paulo (1989) e mestre em Engenharia de Sistemas pela Universidade de Kobe, Japão (1993). Atualmente é pesquisador da Olos Tecnologia e Sistemas e doutorando do Programa de Pós-Graduação em Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo, na área de concentração Engenharia de Computação, atuando como aluno pesquisador no Laboratório de Linguagens e Técnicas Adaptativas do Departamento de Eng. de Computação e Sistemas Digitais. Tem experiência na área de ciência da computação, com ênfase em sistemas de computação, atuando principalmente nos seguintes temas: tecnologia adaptativa, autômatos adaptativos, dispositivos adaptativos, análise e processamento de linguagem natural.

João José Neto é graduado em Engenharia de Eletricidade (1971), mestre em Engenharia Elétrica (1975), doutor em Engenharia Elétrica (1980)e livre-docente (1993) pela Escola Politécnica da Universidade de São Paulo. Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo e coordena o LTA – Laboratório de Linguagens e Técnicas Adaptativas do PCS – Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguintes temas: dispositivos adaptativos, tecnologia adaptativa, autômatos adaptativos, e em suas aplicações à Engenharia de Computação, particularmente em sistemas de tomada de decisão adaptativa, análise e processamento de linguagens naturais, construção de compiladores, robótica, ensino assistido por computador, modelagem de sistemas inteligentes, processos de aprendizagem automática e inferências baseadas em tecnologia adaptativa.