Compressão gramatical com extração eficiente

Danyelle da Silva Oliveira Angelo

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2024

Danyelle da Silva Oliveira Angelo

Compressão gramatical com extração eficiente

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Felipe Alves da Louza
Coorientador: Guilherme Pimentel Telles

Uberlândia
2024

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

A584 Angelo, Danyelle da Silva Oliveira, 1998-
2024 Compressão gramatical com extração eficiente [recurso eletrônico] / Danyelle da Silva Oliveira Angelo. - 2024.

Orientador: Felipe Alves da Louza.
Coorientador: Guilherme Pimentel Telles.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2024.341
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Louza, Felipe Alves da,1988-, (Orient.). II. Telles, Guilherme Pimentel,1972-, (Coorient.). III. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação de Mestrado, 13/2024, PPGCO				
Data:	06 de maio de 2024	Hora de início:	14:47	Hora de encerramento:	16:40
Matrícula do Discente:	12222CCP009				
Nome do Discente:	Danyelle da Silva Oliveira Angelo				
Título do Trabalho:	Compressão gramatical com extração eficiente				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Ciência de Dados				
Projeto de Pesquisa de vinculação:	FAPEMIG (APQ-01217-22)				

Reuniu-se por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Guilherme Pimentel Telles- IC/UNICAMP (Coorientador), Marcelo Keese Albertini- FACOM/UFU, Gonzalo Navarro Badino - Dept. of Computer Science, Universidad de Chile e Felipe Alves da Louza - FEELT/UFU, orientador da candidata.

Os examinadores participaram desde as seguintes localidades: Gonzalo Navarro Badino - Santiago/Chile. Os outros membros da banca e a aluna participaram da cidade de Uberlândia.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Felipe Alves da Louza, apresentou a Comissão Examinadora e a candidata, agradeceu a presença do público, e concedeu à Discente a palavra para a exposição do seu trabalho. A duração da apresentação da Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir a candidata. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando a candidata:

Aprovada

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

Ressalta-se que o examinador externo Gonzalo Navarro Badino, por ser estrangeiro, residente em outro país e não possuir CPF registrado no Brasil não assinará a ata de defesa.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Guilherme Pimentel Telles**, **Usuário Externo**, em 07/05/2024, às 14:11, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Felipe Alves da Louza**, **Professor(a) do Magistério Superior**, em 07/05/2024, às 14:12, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marcelo Keese Albertini**, **Professor(a) do Magistério Superior**, em 07/05/2024, às 18:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5372523** e o código CRC **6FC2BC89**.

Referência: Processo nº 23117.029463/2024-27 SEI nº 5372523

Agradecimentos

Mais uma etapa desta jornada incrível que é a vida está sendo finalizada e não poderia encerrá-la sem os devidos agradecimentos.

Agradeço imensamente à minha família, meu maior incentivador. À minha mãe, Adriana Angelo, e ao meu pai, Wilton Angelo, que desde a infância me ensinaram que os livros e os estudos abrem portas para um mundo de infinitas possibilidades, seu apoio incansável foi fundamental para que eu alcançasse este momento. Às minhas irmãs, Dayane e Marlayne, agradeço por sonharem comigo e por me apoiarem de todas as formas possíveis. Agradeço também aos meus amigos, cujo apoio me ajudou a seguir firme neste caminho.

Não poderia deixar de expressar minha gratidão aos meus orientadores, Prof. Dr. Felipe Louza e Prof. Dr. Guilherme Telles. Este trabalho é fruto da confiança e apoio de vocês, obrigada por me guiarem. Agradeço pela paciência, compreensão, risos, dedicação e por todo conhecimento compartilhado. Ao Prof. Dr. Daniel Saad do Instituto Federal de Brasília, minha sincera gratidão, seus ensinamentos foram peças fundamentais na minha jornada, aprendi muito com você ao longo dos anos e continuo aprendendo através dos seus trabalhos.

Agradeço às agências de fomento CAPES, CNPq (406418/2021-7 e 408314/2023-0) e FAPEMIG (APQ-01217-22) pelo apoio ao desenvolvimento deste trabalho.

A Deus, meu alicerce, que esteve ao meu lado nos dias bons e mals, mostrando-me que sempre há tempo para sonhar.

Resumo

Apresentamos um compressor, denominado GCX (Grammar Compression modulo X), baseado na técnica de compressão gramatical por ordenação de sufixos induzida, introduzida no GCIS (NUNES et al., 2022). Nosso método incorpora a fatoração de textos utilizada pelo algoritmo de ordenação de sufixos DC3 (KäRKKäINEN; SANDERS; BURKHARDT, 2006), para criar uma gramática livre de contexto capaz de produzir o texto de entrada. Nós avaliamos o desempenho do nosso algoritmo utilizando diferentes valores de cobertura X, e introduzimos uma heurística baseada na média do prefixo comum mais longo entre as regras da gramática para definir o valor dessa cobertura. GCX suporta operações de extração rápidas sobre o texto codificado sem a necessidade de descompressão completa. Nossos experimentos foram realizados com conjuntos de dados reais e artificiais e os resultados mostraram que o GCX, em comparação com o GCIS, na maioria dos casos é mais rápido para comprimir, mais rápido para descomprimir, tem uma taxa de compressão pior na maioria das vezes; por outro lado, possui velocidade de extração, aproximadamente 100 vezes mais rápida. Observa-se um comportamento semelhante ao comparar o desempenho do GCX com o do método RePair.

Palavras-chave: Compressão, extração, gramática, estrutura de dados compactas, algoritmos.

Abstract

We present a grammar compressor, called GCX (Grammar Compression modulo X), based on the induced suffix sorting grammar compression technique introduced in GCIS (NUNES et al., 2022). Our method incorporates the text factorization used by algorithm DC3 (KäRKKäINEN; SANDERS; BURKHARDT, 2006) to create a context-free grammar that produces the input string. We evaluated the performance of our algorithm using different values of covering X, and we introduce a heuristic based on the average longest common prefix between the rules of the grammar to define this coverage. GCX supports very fast extraction on the encoded grammar without the need to complete decompression. Experiments with real and artificial datasets showed that GCX, compared with GCIS, in most cases, is faster to compress, faster to decompress, have worse compression ratio most often; however, it has an extraction speed approximately 100 times larger. Similar behavior is observed when comparing the performance of GCX with that of RePair.

Keywords: Compression, extraction, grammar, compact data structures, algorithms.

Sumário

Capítulo 1

Introdução

O crescente avanço da tecnologia aliado à sua disseminação tem levado a um aumento progressivo na produção de dados. Esse avanço transformou a forma como vivemos e trabalhamos. Em um estudo realizado em 2020 pela *International Data Corporation* (IDC) foi destacado que o crescimento de novas aplicações e dispositivos, com uma variedade de formatos e recursos, lançados anualmente, combinado ao aumento da adoção de serviços de transporte, streaming e armazenamento em nuvem, tem contribuído significativamente para o aumento na produção de dados (REINSEL; GANTZ; RYDNING, 2020). De acordo com as previsões desse estudo, em 2025 75% da população mundial estará interagindo diariamente com essas aplicações, resultando em um aumento substancial na produção de dados. O estudo *"Data never Sleeps 11"* da Domo, revelou que em 2023 64,6% da população mundial interagia com esse tipo de aplicação (DOMO, 2023). Segundo a IDC, esse crescimento exponencial impulsionará o volume de dados produzidos de 45 Zettabytes (ZBs) em 2019 para 175 ZBs em 2025.

Além dos dados comportamentais baseados na sociedade (mercados, compras, tráfego, cliques, navegação na WEB, curtidas, redes sociais e outros), a sociedade moderna tem produzido em larga escala dados gerados por humanos (voz, livros, notícias e e-mails) e dados do mundo físico (astronômicos, climatológicos, geográficos e biológicos), como observado por Navarro (2016). O autor ressalta que lidar com essas informações não é uma tarefa simples, pois embora nossa capacidade de armazenar dados tenha crescido rapidamente, os dados em si também têm aumentado exponencialmente. Dessa forma, restringir o processamento desses dados à memória principal, onde a latência é menor, está se tornando cada vez mais desafiador.

Nesse contexto, a compressão de dados desempenha um papel crucial. Os algoritmos de compressão são empregados para reduzir o número de bits necessários para representação de dados, que incluem imagens, vídeos, textos e dados gerados por outros processos. As técnicas de compressão podem ser classificadas em compressão com perdas e compressão sem perdas. Como o nome indica, dados codificados a partir de técnicas de compressão com perdas não podem ser reconstruídos (decodificados) com integridade to-

tal. Essa perda não é considerada um problema para algumas aplicações, por exemplo em aplicações que envolvem a transmissão de fala, se a qualidade da fala a ser reconstruída for similar à ouvida no telefone, uma perda significativa de informação pode ser tolerada. Esse tipo de compressão geralmente oferece taxas de compressão muito mais altas do que a compressão sem perdas. Técnicas de compressão sem perdas são essenciais quando os dados originais precisam ser reconstruídos com integridade total, como é o caso de compressão de textos, ou de imagens radiológicas (SAYOOD, 2012).

Dedicamos este trabalho à análise de compressores sem perda para dados textuais, especificamente no estudo do GCIS (NUNES et al., 2022), um compressor gramatical baseado na técnica de ordenação de sufixos induzida. A técnica de compressão gramatical consiste em produzir uma gramática livre de contexto, G, capaz de produzir o texto de entrada. Essa abordagem identifica regularidades em um texto de entrada e as utiliza para gerar um conjunto de regras de produção tal que a sua representação em bits seja menor do que a representação do texto de entrada. Os compressores baseados em gramática geralmente fornecem suporte a extração de subcadeias do texto compactado sem a necessidade de descompactá-lo por completo. O GCIS apresenta um bom tempo de compressão e taxa de compressão competitiva. Além disso, através de estruturas de dados adicionais, o GCIS fornece suporte à extração de subcadeias diretamente do texto compactado. Essa operação é realizada através de descompactação de intervalos do texto determinados por sucessivas buscas binárias em cada nível da gramática.

Neste trabalho apresentamos um compressor gramatical denominado GCX, que utiliza a técnica de fatoração de textos, semelhante à empregada pelo algoritmo de ordenação de sufixos baseado em indução DC3 (KäRKKäINEN; SANDERS; BURKHARDT, 2006), para gerar o conjunto de regras de uma gramática G. Essas regras, diferentemente das geradas pelo compressor GCIS, têm tamanho fixo, o que permite o acesso direto da localização das subcadeias que fazem parte do processo de extração de um intervalo.

Experimentos com dados reais e artificiais mostraram que o GCX é significativamente mais rápido na compressão e descompressão do texto e na extração de subcadeias quando comparado ao GCIS e ao RePair. Entretanto, a taxa de compressão do GCX é inferior.

O Capítulo 2 apresenta conceitos básicos. No Capítulo 3 realizamos um estudo dos algoritmos que ajudaram a fundamentar nossa proposta, são eles: SAIS, GCIS e DC3. Nossa proposta é descrita de forma detalhada no Capítulo 4. Por fim, o Capítulo 5 apresenta os resultados dos experimentos realizados com o GCIS e GCX, enquanto o Capítulo 6 expõe as conclusões em relação aos objetivos e resultados alcançados, bem como as nossas perspectivas em relação a trabalhos futuros.

Capítulo 2

Definições

2.1 Cadeias de caracteres

Definição 1 *Alfabeto (Σ):*

Conjunto ordenado e finito de símbolos $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{\sigma-1}\}$ tal que $\sigma_0 < \sigma_1 < \ldots < \sigma_{\sigma-1}$. O tamanho de um alfabeto é denotado como $|\Sigma| = \sigma$.

Definição 2 *Cadeia de caracteres S:*

Sequência finita de símbolos formada a partir do alfabeto Σ, podendo ser representada por:

$$S = S[0] \ldots S[n-1] \ \mid \ S[i] \in \Sigma$$

O tamanho de uma cadeia é denotado por $|S| = n$.

Definição 3 *Subcadeia:*

Sequência de elementos consecutivos compreendidos no intervalo $[i,j], 0 \leq i < j \leq n-1$ de uma cadeia S. Ou seja, $S[i,j] = S[i] \ldots S[j]$.

Definição 4 *Super-cadeia:*

Uma cadeia T é uma super-cadeia de uma cadeia S, se S é uma subcadeia de T.

Definição 5 *Prefixo ($S[0,j]$):*

O prefixo de uma cadeia S é uma subcadeia no intervalo $[i,j]$, de modo que $i = 0$ e $j \leq n-1$. Por exemplo, na cadeia $S[0,13]$ = banaananaanana, banaa é o prefixo correspondente ao intervalo $[0,4]$ da cadeia S.

Definição 6 *Sufixo ($S[i,n-1]$):*

Um sufixo de S é uma subcadeia no intervalo i,j tal que $i \geq 0$ e $j = n-1$. Por exemplo, naanana é o sufixo correspondente a $S[7,13]$ em banaananaanana.

Definição 7 *Concatenação (·):*

Dadas duas cadeias $X[0,n-1]$ e $Y[0,n'-1]$, a operação de concatenação (·) forma uma nova cadeia ao anexar a segunda cadeia ao final da primeira. Em outras palavras, a concatenação de X e Y é definida como $X \cdot Y = X[0] \dots X[n-1]Y[0] \dots Y[n'-1]$.

Definição 8 *Ordem lexicográfica (<):*

Através da ordem lexicográfica podemos determinar a precedência de cadeias. Considere os símbolos σ_1 e σ_2. Sejam $\sigma_1 X$ e $\sigma_2 Y$ duas cadeias, a relação de ordem lexicográfica entre $\sigma_1 X$ e $\sigma_2 Y$ pode ser expressa como:

$$\sigma_1 X \ < \ \sigma_2 Y \ \rightarrow \ (\sigma_1 < \sigma_2) \ \vee \ (\sigma_1 = \sigma_2 \ \wedge \ X < Y)$$

Definição 9 *$LCP(S_1, S_2)$:*

Do inglês longest common prefix, $LCP(S_1, S_2)$ define o comprimento do maior prefixo comum entre duas cadeias S_1 e S_2.

Definição 10 *LCP_{mean}:*

Considere a cadeia S de tamanho n. O vetor LCP armazena o comprimento do maior prefixo comum entre sufixos consecutivos de S em ordem lexicográfica, isto é, $LCP[0] = 0$ e $LCP[i] = LCP(S[SA[i], n-1], S[SA[i-1],n-1])$ para $1 \leq i \leq n-1$.

O LCP_{mean} é definido como a média dos valores armazenados no vetor LCP, ou seja:

$$LCP_{mean} = \frac{1}{n-1} * \sum_{i=0}^{n-1} LCP[i]$$

2.2 Vetor de Sufixos

Com aplicações em áreas relacionadas a bioinformática, compressão e recuperação de dados, um vetor de sufixos (SA, *Suffix Array*) é definido como um vetor de índices inteiros associados ao início de cada sufixo de uma cadeia S, sendo que os sufixos devem aparecer em ordem lexicográfica em SA (MANBER; MYERS, 1993).

Devido ao amplo uso dos vetores de sufixos e ao grande volume de dados produzidos/armazenados pelas aplicações que fazem o uso dessa estrutura, diversos algoritmos foram propostos para a construção dos vetores de sufixos de maneira eficiente em termos de espaço e tempo (PUGLISI; SMYTH; TURPIN, 2007; LOUZA; GOG; TELLES, 2020), entre eles, destaca-se o algoritmo *Suffix Array by Induced-Sorting* (SAIS), proposto por (NONG; ZHANG; CHAN, 2009). Abordaremos com mais detalhes este e outro algoritmo para a construção de vetores de sufixos no Capítulo 3.

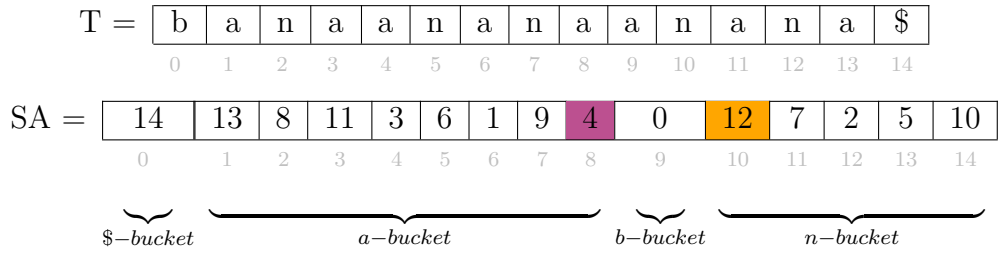
Vamos assumir que toda cadeia $S \in \Sigma^+$ é terminada com um símbolo \$ que é o menor símbolo em Σ e que não ocorre em nenhuma outra posição de S.

Definição 11 *c-bucket:*

Seja S uma cadeia sobre Σ e c um símbolo em Σ. Seja SA o vetor de sufixos para S. c-bucket é o intervalo contíguo em SA onde todos os sufixos de S começam com o símbolo c.

Definição 12 *Cabeça e cauda: Posição inicial e final em um c-bucket, respectivamente.*

O vetor de sufixos SA para a cadeia *banaananaanana* é mostrado abaixo. Cada *c-bucket* é indicado na parte inferior de SA. A cabeça do *n-bucket* é destacada em laranja, enquanto a cauda do *a-bucket* é destacada em rosa.



Encontrar todas as ocorrências de uma subcadeia P, chamada de padrão, em S utilizando SA leva tempo $O(m \log n)$, onde m é o tamanho da subcadeia P. Como os sufixos prefixados pelo padrão P estão armazenados de forma contígua em SA, podemos realizar duas buscas binárias para encontrar a posição inicial (sp) e final (ep) dos sufixos prefixados por P (NAVARRO; MäKINEN, 2007). Obter o índice sp significa localizar todos os sufixos que são lexicograficamente maiores ou iguais a P em SA. Da mesma forma, obter o índice ep, significa obter o maior sufixo de S que é prefixado por P. Assim, o intervalo $SA[sp, ep]$ conterá todas as posições iniciais das ocorrências do padrão P em S. A busca por sp e ep requer comparar P e $S[SA[i], SA[i] + m - 1]$, com um tempo de execução no pior caso de $O(m)$. Considerando que cada busca binária tem complexidade de tempo de $O(\log n)$, então o tempo total para encontrar todas as ocorrências de P em S é de $O(m \log n)$. No entanto, ao utilizar espaço de armazenamento extra para guardar o valor do LCP entre sufixos consecutivos é possível reduzir o tempo de busca para $O(m + \log n)$ (MANBER; MYERS, 1993).

2.3 Compressão

Definição 13 *Compressão:*

Processo de transformação de uma cadeia S em uma cadeia S' de modo que a representação em bits de S' seja menor do que a representação em bits de S para cadeias que podem ser comprimidas.

Existem duas categorias principais de compressores sem perdas:

- Dicionário:

  Nesta abordagem é construído um dicionário para armazenar o mapeamento das subcadeias de uma cadeia a padrões (códigos) menores. Durante o processo de compressão, as subcadeias da cadeia são substituídas pelos respectivos padrões. O processo de descompressão consiste na substituição dos padrões pelas subcadeias originais. Um exemplo desse tipo de compressor é o *Lempel-Ziv and Welch* (SAYOOD, 2012).

- Símbolo (ou estatística): Gera códigos para os símbolos de entrada com base na frequência do símbolo. Um exemplo de compressão orientada a símbolos é a codificação de *Huffman* (BELL; WITTEN, 1994).

2.3.1 Compressão Gramatical

As gramáticas são amplamente usadas por serem capazes de descrever linguagens, como documentos XML, fórmulas aritméticas, linguagens de programação e outros (NAVARRO, 2016). No âmbito de compressão, uma gramática é uma forma de compactar uma cadeia.

Definição 14 *Gramática livre de contexto:*

$$G = \{\Sigma, \Gamma, P, X_s\}$$

sendo:

- Σ *o conjunto de símbolos terminais de* G*;*
- Γ *o conjunto de símbolos não-terminais de* G*;*
- P *o conjunto de regras de produção, onde cada regra tem o formato:*

  $$X_i \rightarrow \alpha$$

  com $X_i \in \Gamma$ *e* $\alpha \in (\Sigma \cup \Gamma)^*$*;*
- X_s *o símbolo inicial de* G*.*

Definição 15 *Derivação:*
$\beta\ X\ \gamma$ *deriva* $\beta\ \alpha\ \gamma$ *se* $\beta, \gamma, \alpha \in (\Sigma \cup \Gamma)^*$ *e* $X \rightarrow \alpha \in P$*.*

Definição 16 *Compressão gramatical:*
Técnica aplicada sobre uma cadeia de entrada S*, de tamanho* n*, que consiste em encontrar uma gramática livre de contexto capaz de produzir exclusivamente* S*.*

A gramática G apresentada abaixo gera a cadeia de entrada $S = banaananaanana$. O algoritmo GCIS, detalhado no Capítulo 3, foi empregado para sua criação. Ao aplicar sucessivamente as regras do conjunto P sobre X_s, obtém-se unicamente S, conforme demostrado a seguir.

$$G = \{\{a, b, n, \$\}, \{r_0^2, r_1^2, r_2^2, r_3^2, r_0^1, r_1^1, r_2^1, r_3^1, r_4^1\},\ P, r_0^2 \cdot r_3^2 \cdot r_2^2 \cdot r_1^2\}$$

Regras de produção[1]

$$\begin{aligned} P\ =\{&r_0^1 \to b,\ r_4^1 \to an\textcolor{gray}{a},\ r_2^1 \to aan\textcolor{gray}{a},\ r_3^1 \to ana\textcolor{gray}{\$},\ r_1^1 \to \$,\ r_0^2 \to r_0^1 \cdot r_4^1, \\ &r_3^2 \to r_2^1 \cdot r_4^1 \cdot \textcolor{gray}{r_2^1},\ r_2^2 \to r_2^1 \cdot r_3^1 \cdot \textcolor{gray}{r_1^1},\ r_1^2 \to r_1^1\} \end{aligned}$$

Ao aplicar as regras de produção dessa gramática sobre X_s, geramos:

$$S = r_0^1 \cdot r_4^1 \cdot r_2^1 \cdot r_4^1 \cdot r_2^1 \cdot r_3^1 \cdot r_1^1$$

Neste ponto ainda é possível aplicar derivações sobre os símbolos de S. Portanto, ao aplicar novamente as regras de P sobre S, teremos:

$$S = b \cdot an \cdot aan \cdot an \cdot aan \cdot ana \cdot \$$$

Encontrar a menor gramática que seja capaz de gerar S é um problema NP-difícil (CHARIKAR et al., 2005), no entanto, existem abordagens práticas que são eficazes. Uma das grandes vantagens ao empregar técnicas de compressão gramatical é a possibilidade de extrair qualquer subcadeia da cadeia original sem a necessidade de descompactá-la integralmente.

[1] Os símbolos destacados em cinza representam a sobreposição de trechos de regras geradas a partir de subcadeias consecutivas. Embora o GCIS não armazene estes símbolos na prática, optamos por incluí-los em nossos exemplos para tornar a compreensão mais simples.

Capítulo 3

Trabalhos Relacionados

3.1 DC3

O algoritmo *Difference Cover module 3* (DC3), proposto por Kärkkäinen, Sanders e Burkhardt (2006), é baseado na técnica de divisão e conquista e tem por objetivo construir um vetor de sufixos (SA) em tempo linear, de forma eficiente em termos de espaço.

O DC3 emprega fatoração de cadeias e consiste em 3 etapas. A primeira etapa divide a cadeia de entrada $S[0,n-1]$ em duas outras cadeias: S_{12} e S_0. A cadeia S_{12} é composta por 2/3 dos sufixos da cadeia original, enquanto que S_0 é formada pelos demais sufixos. O resultado da primeira etapa é a produção do vetor de sufixos SA_{12}, produzido a partir da ordenação recursiva dos sufixos de S_{12}.

A segunda etapa tem como resultado a produção do vetor de sufixos SA_0. Esse processo é realizado por meio da indução da ordem dos sufixos de S_0 usando como referência a disposição dos sufixos em SA_{12}.

Finalmente, na terceira e última etapa os vetores de sufixos gerados nos passos anteriores são combinados, resultando no vetor de sufixos final (SA).

Aprofundamos os detalhes desse algoritmo a seguir, mas antes, para melhor compreensão, considere j como o nível atual da recursão, o ranking de um sufixo $S[i,n-1]$ como a sua posição em SA, e considere também a seguinte definição para uma cadeia de caracteres S_k:

Definição 17 *S_k é um conjunto de cadeias definido como:*

$$S_k = \{S[i,n-1] \ \mid i \in [0,n-1] \ e \ i \ mod \ 3 = k, \ para \ k = 0,1,2.\}$$

Algoritmo DC3

1. **Calcula** SA_{12} (Figura 1)

   a) *Redução do problema*: defina os sufixos a serem ordenados, através do conjunto abaixo:

   $$S_{12} = S_1 \cup S_2$$

   b) *Ordenação*: execute DC3 recursivamente, até que todos os sufixos de S_{12} estejam ordenados. O núcleo desta fase é descrito pelos passos a seguir:

      i. Usando o algoritmo *radix-sort*, ordene os sufixos em S_{12} usando como referência os 3 primeiros caracteres de cada sufixo;

      ii. Nomeie cada *tripla* de sufixo usando o seu ranking lexicográfico. Se todas as tuplas forem únicas, então é possível obter SA_{12} imediatamente, e o processo recursivo deve ser interrompido;

      iii. *Caso contrário*, construa uma cadeia reduzida u^{j+1}, de tamanho $m_{12} = |SA_{12}| + 1$. Essa cadeia tem a forma:

      $$u^{j+1} = \text{ ranking dos sufixos em } S_1 \ \# \ \text{ ranking dos sufixos em } S_2$$

      Sendo que:

      - ❏ # é um caractere sentinela, que não aparece em nenhum outro lugar da cadeia. Durante este texto ele é usado como recurso explicativo e tem por objetivo demarcar o fim e início de S_1 e S_2 respectivamente.
      - ❏ O *ranking* de cada sufixo deve ser adicionado em u^{j+1} na ordem em que estes aparecem em S_{12}. Proceder desta forma produzirá uma representação implícita de todos os sufixos em S_{12} (com um alfabeto diferente). Em outras palavras, isso implica que ao ordenarmos u^{j+1} no próximo nível estaremos computando também SA_{12}.
      - ❏ Cada elemento $u^{j+1}[i]$ pode ser mapeado para SA^j_{12} como mostrado abaixo:

      $$i < \frac{m_{12}}{2} \rightarrow u^{j+1}[i] = SA^j_{12}[2 * i]$$
      $$i > \frac{m_{12}}{2} \rightarrow u^{j+1}[i] = SA^j_{12}\left[2 * \left(i - \frac{m_{12}}{2}\right) - 1\right]$$

      Após gerar a nova cadeia (u^{j+1}), chame DC3 de forma recursiva, passando u^{j+1} como parâmetro.

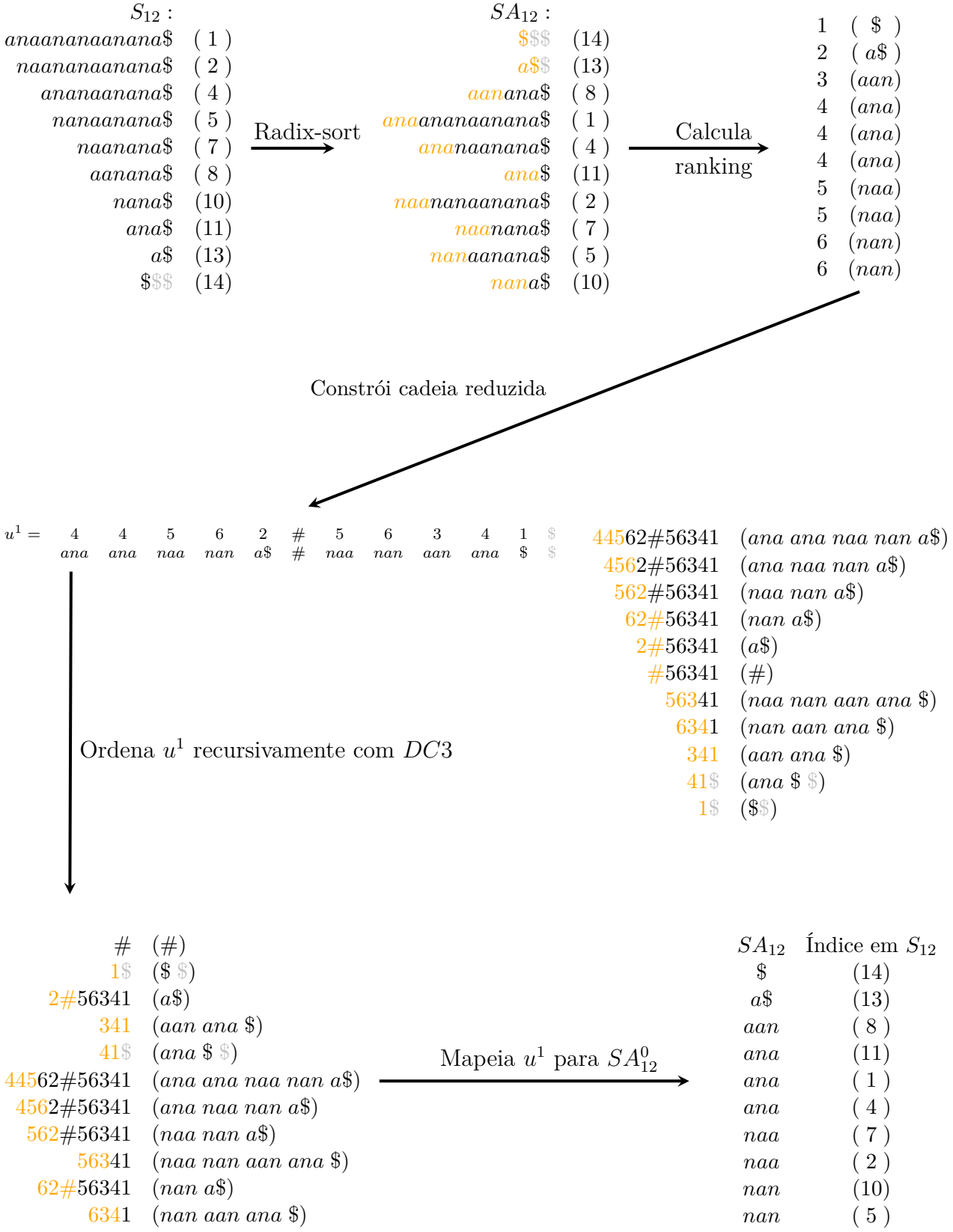


Figura 1 – **Computando SA_{12} com DC3 para $S^0 = banaananaanana\$$.** Destacamos com a cor laranja as tuplas utilizadas durante o processo de ordenação com radix-sort. O símbolo \$ foi adicionado a S_{12} e a u^1 com a finalidade de ajustar o tamanho da cadeia.

2. **Calcula** SA_0 (Figura 2)

   a) *Ordenação com radix-sort*: ordene S_0 usando o *primeiro* caractere de cada sufixo como referência.

   b) Utilize a operação de ranking para nomear as cadeias ordenadas no passo anterior. Se não ocorrer empate entre as cadeias ordenadas, então SA_0 pode ser produzido e a recursão interrompida;

   c) *Ordenação com indução*: caso não tenha sido possível obter SA_0 a partir do item 2b, considere o seguinte:

   ❏ Os índices $i \bmod 3 \neq 0$ estão ordenados em SA_{12}, logo, se inserirmos $i \bmod 3 = 0$ na ordem em que $i+1$ aparece em SA_{12}, teremos SA_0 ordenado em relação ao sufixo $i+1$.
   Assim, considere os sufixos $S[i,n-1]$ e $S[j,n-1] \in S_0$ para determinar qual destes possui menor ordem lexicográfica (ranking) basta verificar se $S[i] = S[j]$ e qual a ordem dos sufixos $S[i+1,n-1]$ e $S[j+1,n-1]$ em SA_{12}. Portanto, a ordem destes sufixos determinará a ordem de $S[i,n-1]$ e $S[j,n-1]$ em SA_0.

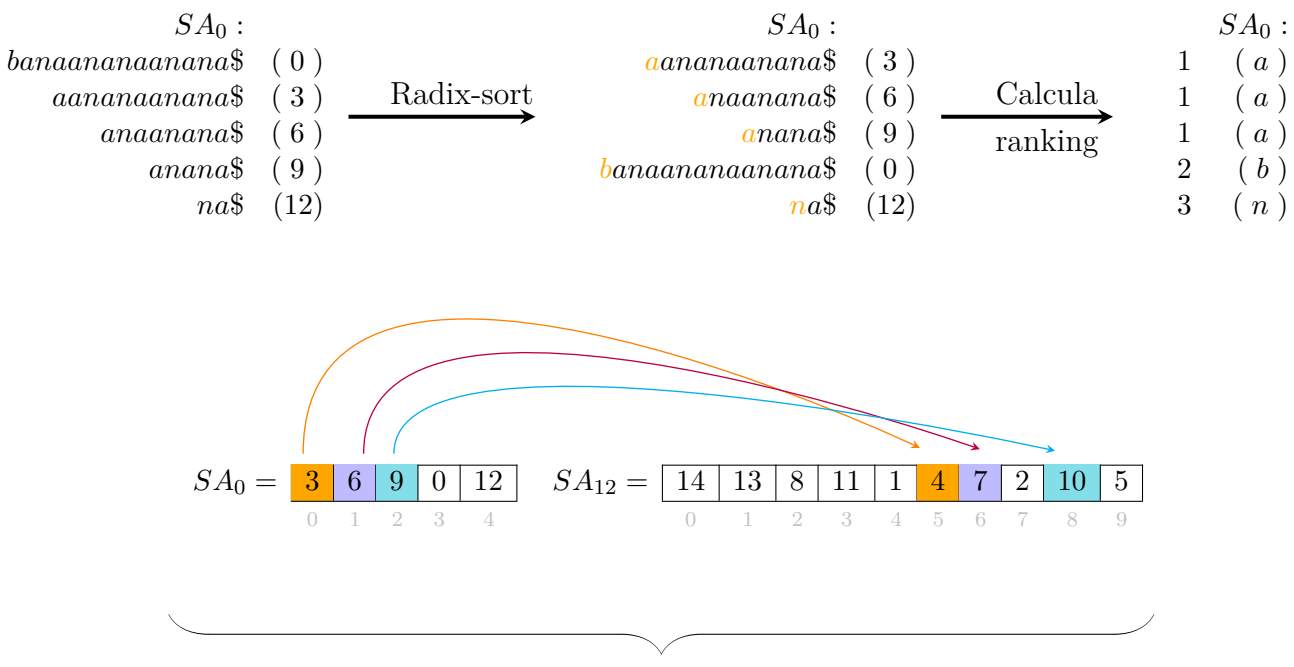


Figura 2 – **Computando** SA_0 **com DC3 para** $S^0 =$ *banaananaanana*$.

3. **União** (Figura 3)

   Para esta etapa, considere posições i, j em SA_{12} e SA_0 respectivamente.

   a) Verifique qual dos sufixos $S[i,n-1]$ e $S[j,n-1]$ tem o menor ranking, para isso, faça:

      i. Compare o primeiro caractere do sufixo $S[i,n-1] \in SA_{12}$ com o primeiro caractere do sufixo $S[j,n-1] \in SA_0$;

      ii. Se os caracteres forem diferentes, a ordem dos sufixos pode ser obtida diretamente. No entanto, se os caracteres forem iguais, a ordem lexicográfica pode ser determinada considerando os dois cenários a seguir:

         (a) $SA_{12}[i] \mod 3 = 1 \rightarrow SA_{12}[i]+1,\ SA_0[j]+1 \in SA_{12}$, portanto é possível obter a ordem de precedência de $SA_{12}[i]$ e $SA_0[j]$ diretamente de SA_{12}.

         (b) $SA_{12}[i] \mod 3 = 2 \rightarrow SA_{12}[i]+1 \in SA_0,\ SA_0[j]+1 \in SA_{12}$. Nesse caso, é necessário comparar os símbolos $S[SA_{12}[i]+1]$ e $S[SA_0[j]+1]$, pois não é possível induzir a sua ordem lexicográfica a partir de SA_{12}. Se esses símbolos forem iguais, adicionamos 1 aos seus índices, isso implica que $SA_{12}[i]+2 \in SA_{12}$ e $SA_0[j]+2 \in SA_{12}$ tornando possível obter a sua ordem lexicográfica a partir de SA_{12}.

   Esse processo de tomada de decisão do menor sufixo em ordem lexicográfica realiza no máximo 2 comparações.

   b) Adicione o sufixo com menor ordem lexicográfica em SA; incremente o índice i caso o menor sufixo esteja em SA_{12}, e o índice j caso contrário.

   c) Repita esse processo até que o vetor SA seja preenchido completamente.

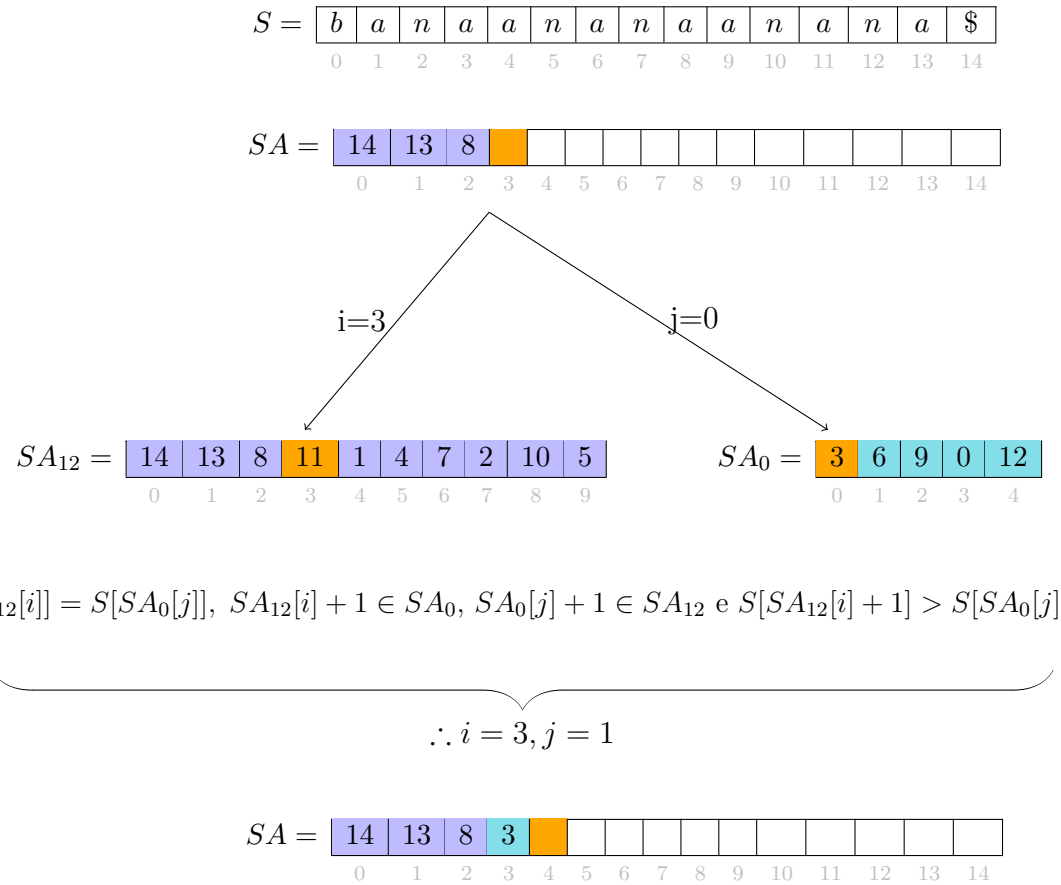
Custo computacional.

Na etapa 1 do DC3 a cadeia de entrada é reduzida a uma cadeia de tamanho $\lceil 2n/3 \rceil$, a cadeia é ordenada de forma recursiva, todas as outras etapas podem ser feitas em tempo linear. Assim, a complexidade de tempo do DC3, é dada pela recorrência:

$$T(n) = T\left(\left\lceil \frac{2n}{3} \right\rceil\right) + O(n) = O(n)$$

3.2 SAIS

O algoritmo *Suffix Array by Induced-Sorting* (SAIS), proposto por Nong, Zhang e Chan (2009), foi o primeiro algoritmo para a criação de vetores de sufixos em tempo linear, que também é rápido na prática. Ao longo desta seção exploraremos o funcionamento do SAIS.



$$S[SA_{12}[i]] = S[SA_0[j]],\ SA_{12}[i]+1 \in SA_0,\ SA_0[j]+1 \in SA_{12} \text{ e } S[SA_{12}[i]+1] > S[SA_0[j]+1]$$

$$\therefore i = 3, j = 1$$

Após percorrer SA_{12} e SA_0 escolhendo o menor sufixo, o vetor SA abaixo é computado.

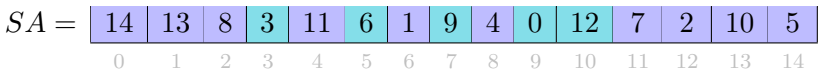


Figura 3 – **Combinando soluções com DC3 para** $S^0 = banaananaanana\$$ - união de SA_{12} e SA_0. Os elementos destacados em laranja indicam a região da memória que cada índice está referenciando. Os elementos destacados em azul e roxo, indicam os sufixos pertencentes à SA_0 e SA_{12} respectivamente.

Tipos de sufixos

Considere os seguintes tipos de sufixos:

Definição 18 *S-sufixo e L-sufixo*[1]

$S[i, n-1]$ *é um S-sufixo se* $S[i, n-1] < S[i+1, n-1]$, *caso contrário* $S[i, n-1]$ *é um L-sufixo. O último sufixo,* $S[n-1] = \$$, *é por padrão S-sufixo.*

[1] Do inglês *Smaller Suffix* e *Larger Suffix*, respectivamente

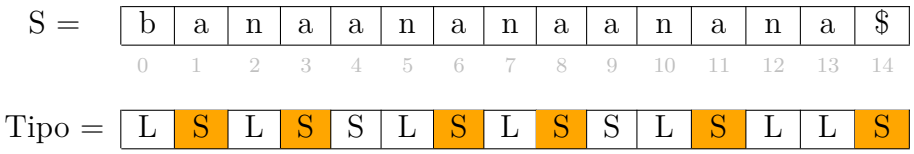


Figura 4 – **SAIS:** tipos de sufixos, os índices i,j das *LMS-subcadeias* estão destacados em laranja.

Definição 19 *LMS-sufixo*[2]

Um sufixo $S[i, n-1]$ *é LMS-sufixo se* $S[i, n-1]$ *é um S-sufixo e* $S[i-1, n-1]$ *é um L-sufixo.*

Definição 20 *LMS-subcadeia*

Subcadeia $S[i,j]$*, com* $S[i, n-1]$ *e* $S[j, n-1]$ *sufixos do tipo LMS, com nenhum outro sufixo LMS entre* $[i+1, j-1]$.

Procedimento detalhado do SAIS

De maneira geral o algoritmo SAIS decompõe o problema da construção do vetor de sufixos através da separação e ordenação de diferentes tipos de sufixos. As quatro etapas do algoritmo SAIS estão descritas a seguir:

0. Identifique os tipos de sufixos da cadeia de entrada, de acordo com sua ordem lexicográfica. Esse processo é feito em tempo linear, e deve iniciar-se da esquerda para a direita (Figura 4);

1. Ordene todo os *LMS-sufixos* e os armazene em um vetor auxiliar SA^R; detalharemos o método para ordenação desses sufixos posteriormente;

2. Percorra SA^R partindo do final em direção à posição 0, e insira cada LMS-sufixo na cauda de seu respectivo *c-bucket* em SA. Em seguida decremente a cauda (Figura 6 - Passo 2);

3. Percorra SA a partir da posição 0, e para cada sufixo $S[SA[i], n-1]$, se $S[SA[i-1], n-1]$ é um sufixo do tipo L, então insira $SA[i-1]$ na cabeça do seu respectivo *c-bucket*. Em seguida incremente a cabeça (Figura 6 - Passo 3);

4. Percorra SA da direita para a esquerda, e para cada $S[SA[i], n-1]$, se $S[SA[i-1], n-1]$ é um sufixo do tipo S, então insira $SA[i-1]$ na cauda do seu *c-bucket*. Em seguida de decremente a cauda (Figura 6 - Passo 4).

[2] Do inglês: Leftmost S-suffix

Ordenando LMS-sufixos

Para ordenar todos os *LMS-sufixos* é necessário primeiro obter a ordem das *LMS-subcadeias*.

Essa ordem pode ser obtida a partir da execução do algoritmo apresentado anteriormente, com uma pequena modificação introduzida no Passo 2: ao invés de percorrer SA^R, examinamos $S[0, n-1]$ e inserimos cada *LMS-subcadeia* na cauda de seu *c-bucket* em SA. As etapas subsequentes são executadas sem modificações. Ao final do Passo 4, são necessárias duas novas etapas:

1. **Cópia**: $SA[0, n-1]$ deve ser percorrido da esquerda para a direita e cada *LMS-subcadeia* precisa ser inserida no início de SA.

2. **Nomeação**: $s_0, \cdots, s_{n-1^R}$ é o conjunto de todas as *LMS-subcadeias* ordenadas. Para cada s_i deve-se atribuir um *lex-name* (nome) r_i de acordo com a sua classificação lexicográfica. O SAIS realiza esse processo de nomeação como demonstrado abaixo.

   - σ^R é o número de *LMS-subcadeias* distintas em $S[0,n-1]$;
   - SA é percorrido da esquerda para a direita e cada par s_i,s_j é comparado, s_i receberá um nome $r_i \in [1, \sigma^R]$ obedecendo aos seguintes critérios:

$$r_i < r_j \iff s_i < s_j$$
$$r_i = r_j \iff s_i = s_j$$

Através dos nomes gerados é possível obter uma cadeia reduzida $S^R = r_1 \cdots r_{n^R}$. O vetor de *LMS-subcadeias* final pode ser calculado a partir de S^R (SA^R), se todos os nomes gerados pela etapa anterior forem distintos ($n^R = \sigma^R$). Caso contrário, o algoritmo SAIS é chamado de forma recursiva para ordenar os sufixos de S^R.

A Figura 5 ilustra o passo a passo para ordenar as *LMS-subcadeias* da cadeia *banaananaanana*.

Custo computacional.

O tamanho da cadeia reduzida (S^R) recebida como entrada no Passo 1 é no máximo $n/2$, com isso podemos afirmar que o problema é reduzido pelo menos pela metade em cada nível da recursão; os Passos 2, 3 e 4 são realizados em tempo linear, portanto, a relação de recorrência que descreve a complexidade de tempo do SAIS é:

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n) = O(n)$$

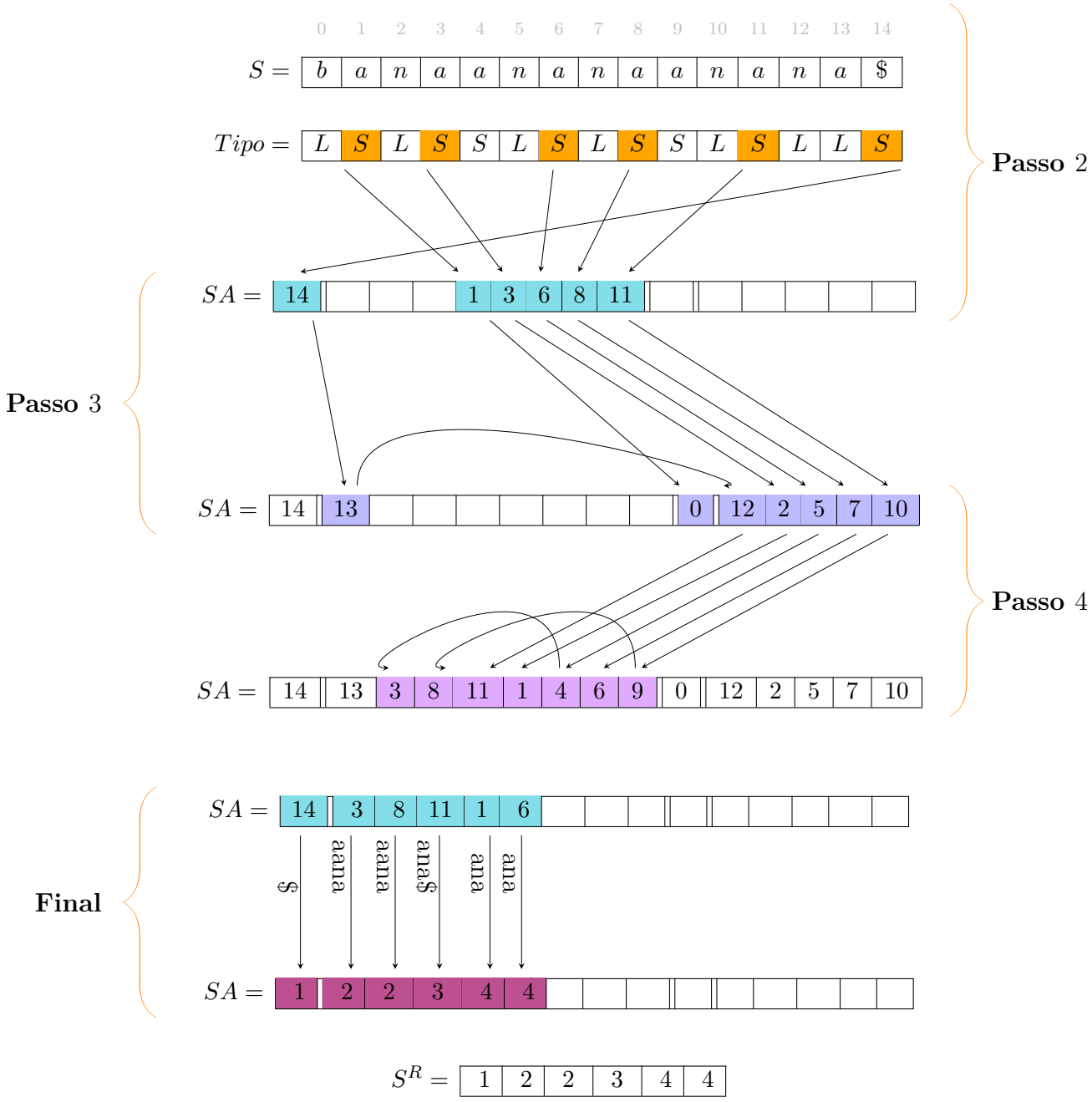


Figura 5 – **SAIS:** ordenando LMS-subcadeias. O resultado final dessa etapa produz $S^R = 122345$. Como o tamanho da cadeia reduzida é maior do que o número de *lex-names* únicos, o SAIS é chamado recursivamente para ordenar S^R. Essa chamada recursiva retornará $SA = 14,8,3,11,6,1$. Destacamos em cada passo os índices dos sufixos que foram adicionados a SA. Os índices destacados em azul correspondem às *LMS-subcadeias*, a cor roxa está associado aos *L-sufixos*, em rosa destacamos os sufixos do tipo S, e por último a cor rosa-escuro representa os *lex-names* atribuídos a cada *LMS-subcadeia*.

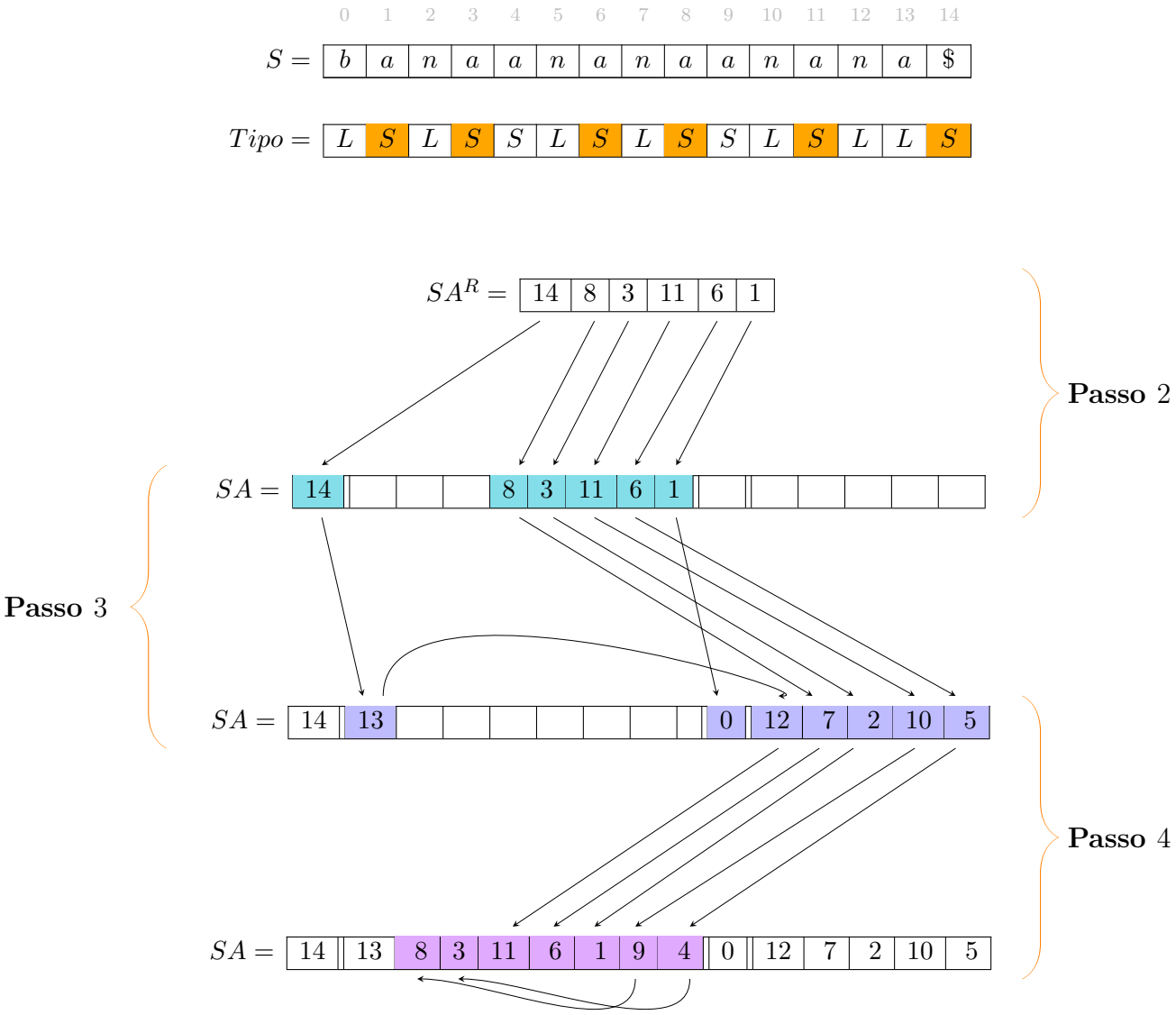


Figura 6 – **SAIS:** ordenando sufixos do tipo L e do tipo S através de indução com as *LMS-subcadeias*. Destacamos em cada passo os índices dos sufixos que foram adicionados a SA. Os índices destacados em azul correspondem às *LMS-subcadeias*, a cor roxa está associado aos *L-sufixos*, e em rosa destacamos os sufixos do tipo S. Ao final da etapa 4 é produzido $SA = 14,13,8,3,11,6,1,9,4,0,12,7,2,10,5$.

3.3 GCIS

O algoritmo *Grammar Compression by Induced Suffix Sorting* (GCIS), proposto por Nunes et al. (2022), é um compressor gramatical que produz regras a partir das subcadeias identificadas pelo algoritmo de ordenação de sufixos SAIS.

O GCIS é o primeiro algoritmo de compressão baseado em ordenação de sufixos por indução e se destaca em relação a outros compressores gramaticais pela taxa de compressão e pelo tempo necessário para compactação (NUNES et al., 2022).

Em particular, GCIS possui maior velocidade de compactação, baixo uso de memória quando comparado ao RePair (LARSSON; MOFFAT, 2000) e a métodos baseados na transformada de Lempel-Ziv (WITTEN; MOFFAT; BELL, 1999), ao mesmo tempo que possui melhor taxa de compressão frente ao RePair, para textos repetitivos. O GCIS é mais lento no processo de descompressão de texto, e mais lento no processo de extração de subcadeias se comparado ao RePair. Por último, o GCIS fornece suporte à geração de vetores de sufixos e ao cálculo de LCP.

3.3.1 Compressão

Para gerar a gramática G, GCIS modifica o SAIS para que a cada nível j da recursão seja possível usar os nomes gerados para as *LMS-subcadeias* para compor o conjunto de símbolos terminais e não-terminais ($\Sigma \cup \Gamma$) da gramática G. Dessa forma o lado esquerdo das regras de produção presentes em P é dado pelo *l*ex-name de cada *LMS-subcadeia*, ao mesmo tempo em que o lado direito da regra é composto pela respectiva *LMS-subcadeia*, ou seja:

$$r_i^j \rightarrow r_1^{j-1} \cdots r_{n^j}^{j-1}$$

Além disso o GCIS cria uma regra adicional a cada nível. Essa regra é definida por: $r_0^j \rightarrow r_0^{j-1} S^j[0, m-1]$ se $j > 0$ ou $r_0^j \rightarrow S[0, m-1]$ se $j = 0$, com m representando o índice do *LMS-sufixo* mais à esquerda de S^j. O objetivo da adição dessa regra é capturar trechos da cadeia (a parte inicial) que não foram capturados pela definição de *LMS-subcadeia*.

Após a criação das regras, GCIS cria uma cadeia reduzida S^{j+1}, com tamanho n^{j+1}, que é a entrada do algoritmo no próximo nível da recursão. GCIS é chamado recursivamente, enquanto $n^{j+1} > \sigma^{j+1}$, ou seja, enquanto houver *LMS-subcadeias* repetidas no texto. Quando não houver mais repetições a execução do algoritmo é interrompida, e o símbolo inicial, X_s, da gramática é criado, de modo que $X_s \rightarrow r_0^\ell \cdots r_{n^\ell}^\ell$ gera $S[0,n-1]$, onde ℓ é o último nível de recursão.

Conforme demonstrado por Nunes et al. (2020), regras consecutivas no conjunto de produções P tendem a compartilhar um prefixo comum devido à ordenação das *LMS-subcadeias* feita pelo $SAIS$. Por esse motivo, cada regra $r_k^j \rightarrow r_i^{j-1}$ é armazenada usando a tupla de valores $(\ell_i, s(r_i^{j-1}))$, onde ℓ_i representa o valor do *lcp* entre duas regras consecutivas (r_{i-1}^{j-1},r_i^{j-1}), e $s(r_i^{j-1})$ denota a parte da cadeia que não é compreendida pelo *lcp*, ou seja, $s(r_i^{j-1}) = r_i^{j-1}[\ell_i + 1,|r_i^{j-1}|]$.

A Figura 7 ilustra o processo de geração da gramática criada pelo GCX.

3.3.2 Descompressão

O processo de descompressão inicia-se a partir da aplicação das regras de produção do último nível sobre o símbolo inicial X_s, que gera $S^{\ell-1}$. A cada nível j são aplicadas sobre S^j as regras de produção do nível anterior. O processo é interrompido ao finalizar

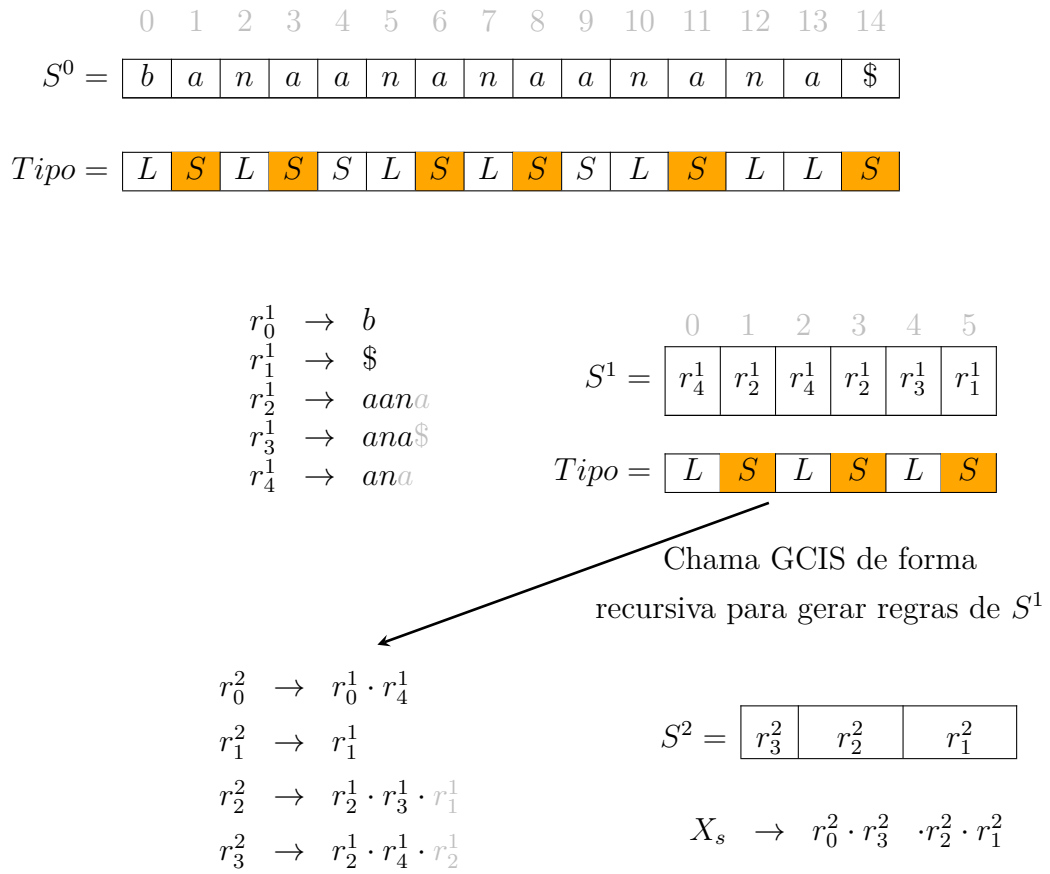


Figura 7 – **GCIS:** recebe a cadeia $S^0 = banaananaanana\$$ como parâmetro de entrada. O primeiro nível da recursão gera 5 regras e produz $S^1 = r_4^1 \cdot r_2^1 \cdot r_4^1 \cdot r_2^1 \cdot r_3^1 \cdot r_1^1$. Como o tamanho da cadeia reduzida é maior que o número de regras geradas, GCIS é chamado recursivamente para ordenar S^1, essa chamada produz 4 regras e $S^2 = r_3^2 \cdot r_2^2 \cdot r_1^2$. Nesse momento o tamanho da cadeia reduzida é menor que o número de regras (σ), logo o processo recursivo é interrompido e a cadeia inicial X_s é gerada. Os índices i,j das *LMS-subcadeias* estão destacadas na cor laranja no vetor "tipo". Os símbolos destacados em cinza indicam sobreposições de regras geradas por sufixos consecutivos.

o primeiro nível da recursão, onde o texto original (S^0) é recuperado. A complexidade de tempo desse processo é limitada por $O(n)$.

3.3.3 Extração

GCIS fornece suporte à extração de qualquer subcadeia compreendida em um intervalo $S[\ell, r]$, através da emulação do processo de descompressão dentro desse intervalo.

Para viabilizar a operação de extração é preciso despender de espaço de armazenamento extra para alocar uma estrutura de dados L que armazenará o comprimento de cada regra. Além de L, é necessário armazenar um vetor de somas parciais (P_s) dos comprimentos dos símbolos da cadeia reduzida gerada no último nível da recursão. Essas

estruturas de dados auxiliarão no processo de busca e localização do intervalo $[\ell', r']$ de S^j que deve ser descomprimido no nível j.

$$P_S(i) = \sum_{j=0}^{i-1} |G(X_{S_i})|,\ X_S \to X_{S_0} \dots X_{S_{k-1}} \text{ e } 0 \leq i \leq k$$

$$L(X) = |G(X)|,\ X \in \Sigma \cup \Gamma$$

L é definido recursivamente como:

$$L(X) = \begin{cases} 1, X \in \Sigma \\ \sum_{i=0}^{|S|-1} L(S[i]),\ X \to S \end{cases}$$

Para descompactar uma subcadeia $S[\ell, r]$, o framework de extração do GCIS segue os passos a seguir (considere k como o tamanho da cadeia gerada por X_s):

1. **Busca binária**: determina o intervalo a,b dos símbolos que serão descompactados na cadeia resultante da derivação de X_S.

$$a = max\{0 \leq k < |S^\ell| \ \mid P_s(k) \leq \ell\}$$

$$b = min\{0 \leq k < |S^\ell| \ \mid P_s(k) > r\} - 1$$

2. $\boldsymbol{E^\ell = S[a,b]}$ **:** seja ℓ o número de níveis da gramática e S a cadeia derivada de X_s. Defina $E^\ell = S[a, b]$ e siga os passos abaixo para $i = \ell$ até $i = 1$;

    1. Derive cada símbolo não-terminal $X \in E^i$ para descompactar a cadeia compreendida no intervalo $[\ell'$,$r']$, ou seja E^{i-1}. E^{i-1} corresponde à cadeia que passou pelo processo de compressão no nível $i - 1$. Além disso, $G(E^{i-1}) = S[\ell', r']$ é uma super-cadeia de $S[\ell, r]$;
    2. Ajuste o limite inferior ($[\ell'$) e superior ($r']$) do intervalo de E^{i-1}, de modo que $S[\ell', r']$ sempre seja considerada super-cadeia de $S[\ell, r]$. Considere:
        - ❑ Se $i = 1$, então E^0 contém apenas símbolos terminais. Desse modo, a extração da subcadeia desejada deve ser feita no intervalo $E^0[\ell - \ell', r - \ell']$;
        - ❑ Se $i > 1$, então E^{i-1} contém apenas símbolos não-terminais, logo é necessário continuar o processo de derivação. Neste caso, atualizaremos o intervalo da cadeia E^{i-1} por meio uma busca linear em L pelos índices a e b, tais que:

$$a = max\left\{0 \leq k < |E^i| \ \middle|\ \ell' + \sum_{j=0}^{k-1} L(E^i[j]) \leq \ell\right\}$$

$$b = max\left\{0 \leq k < |E^i| \ \middle|\ r' + \sum_{j=k}^{|E^i|-1} L(E^i[j]) \geq r\right\}$$

O processo de extração é finalizado, quando $i = 0$.

A Figura 8 é um exemplo do procedimento usado pelo GCIS para extrair a subcadeia compreendida no intervalo 5,13 da cadeia *banaananaanana*, compactado na Figura 7.

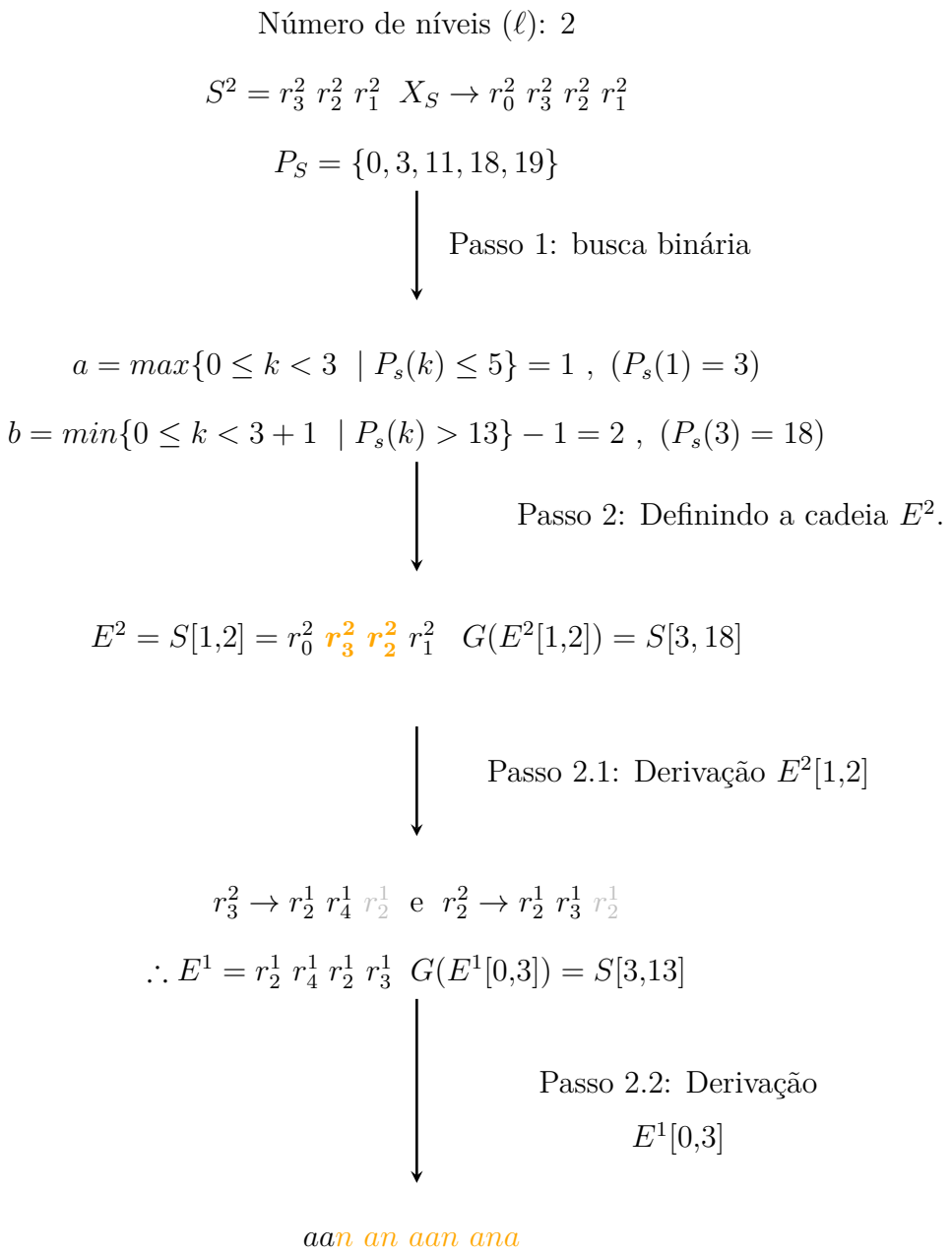


Figura 8 – **GCIS:** extração da subcadeia compreendida no intervalo $S[5,13]$.

Codificação

O GCIS oferece duas opções de codificação para salvar o tamanho de cada cadeia $s(r_i^{j-1})$ e o valor do LCP entre duas regras consecutivas: *Simple8B* e *Elias-Fano*.

A codificação *Simple8B* expande todas as regras da cadeia durante o processo de descompressão, o que a torna altamente eficiente. No entanto, para fornecer suporte a operação de extração é necessário decodificar regras aleatórias de forma individual em cada nível da gramática, devido à isso, só é possível realizar operações de extração utilizando a codificação *Elias-Fano*. Para os símbolos de cada cadeia $s(r_i^{j-1})$ é empregada uma codificação de inteiros de tamanho fixo.

Capítulo 4

GCX

Neste capítulo apresentamos o algoritmo *Grammar Compression modulo X* (GCX), um compressor gramatical inspirado no GCIS. O GCX utiliza o principio de fatoração de cadeias empregado pelo DC3 para gerar as regras que compõe sua gramática. Em comparação com o GCIS, o GCX é mais eficiente para a extração de subcadeias do texto compactado. Essa melhoria decorre da natureza da gramática produzida, que contém regras de tamanho fixo. Devido a essa característica, a navegação pelo texto compactado durante a extração é otimizada, pois, ao conhecer previamente o tamanho das regras de cada nível através de cálculos matemáticos simples, eliminamos a necessidade de descompactar intervalos irrelevantes do texto para encontrar um intervalo [a,b] no texto compactado.

4.1 Compressão

O algoritmo GCX comprime uma cadeia de entrada S através da produção de uma gramática $G = \{\Sigma, \Gamma, P, X_s\}$ que gera unicamente S. Seja X um número inteiro. Para uma cadeia de entrada S são adicionados terminadores \$ de modo que $|S| \mod X = 0$.

Para gerar o conjunto P de regras dessa gramática é realizada a ordenação de todas as subcadeias α de S cuja posição inicial seja múltipla de X. Após a ordenação, para cada subcadeia distinta α_i é atribuído um nome (lex-name), r_k, o que possibilitará a criação de uma regra de produção no formato $r_k \rightarrow \alpha_i$, onde k é a classificação (ranking) da cadeia α_i. Cada ocorrência da subcadeia α_i em S é substituída por r_k, o que produz uma nova cadeia reduzida u^{j+1} que será fornecida para o algoritmo em uma chamada recursiva.

O processo para a geração de G é finalizado quando $|S^j| = |P^j|$, ou, quando o processo de criação de regras deixar de convergir. O GCX modifica o DC3 nos seguintes aspectos.

Considere o $j-$ésimo nível da recursão.

1. No DC3 a cobertura dos sufixos é fixada em 3. Para construir a gramática do GCX definimos os seguintes critérios de cobertura:

- ❏ *Variável por nível*: X define o tamanho das regras (cobertura) de um nível e é atualizado no início de cada chamada recursiva. Dessa forma é possível minimizar o número de regras em cada nível ao definir a cobertura com base nas características de u^j;
- ❏ *Definida pelo LCP médio ($LCP_{mean} = X$) de S^j*: essa heurística visa capturar a taxa de repetitividade da cadeia, e consiste em a cada nível j definir janelas de tamanho y. As subcadeias de S de tamanho y que começam em posições múltiplas de y são ordenadas. Em seguida o LCP_{mean} é calculado para determinar o valor de X. Para essa operação consideramos $y = 32$, se $j = 0$, e $y = X^{j-1}$ se $j > 0$.

2. O algoritmo DC3 ordena primeiro os sufixos cujo o resto da divisão por 3 é diferente de 0. Com o tamanho de regras variável por nível, para garantir que não ocorram perdas da cadeia no processo de compressão essa característica foi modificada. Assim, o GCX constrói as regras de produção do conjunto P com base nos sufixos cujo índice $i \mod X = 0$, essa modificação tornou também o processo de compressão mais simples, uma vez que apenas um tipo de sufixo é considerado;

3. Como o GCX altera o tipo de sufixo usado para ordenação, o formato da cadeia reduzida (u^j) também é alterado. O valor de u^j, utilizado como parâmetro de entrada no $j-$ésimo nível da recursão é construído através da inserção do *lex-name* de cada subcadeia da cadeia u^{j-1} na ordem em que os sufixos aparecem nessa cadeia;

4. Por fim, como mencionado anteriormente, o GCX apresenta um segundo critério de parada: *convergência*. Esse critério foi introduzido com o objetivo de minimizar o tempo de compressão. Quando o cálculo do LCP_{mean} das subcadeias é igual a 1, a continuação da recursão não proporciona benefícios adicionais à taxa de compressão da cadeia, nesse caso é factível interromper o processo de produção de regras e gerar X_s.

Quando o processo recursivo de GCX é interrompido, X_s é produzido, finalizando a última etapa para a geração de G. O Exemplo 1 demonstra a aplicação dos passos acima sobre a cadeia $S = abcabbabcabbaccaccabcabbabcabca$.

Exemplo 1 *Considere o texto:*

- ❏ *Nível 0:*

$S^0 =$

a	b	c	a	b	b	a	b	c	a	b	b	a	c	c	a	c	c	a	b	c	a	b	b	a	b	c	a	b	c	a	\$	\$	\$	\$	\$
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5

  - *Inicialmente, vamos considerar uma janela de tamanho* 6 *(para facilitar a demonstração) para o cálculo do LCP_{mean}. O cálculo do $\lceil LCP_{mean} \rceil$ retorna* $3 \therefore x = 3$*;*

- *Com $x = 3$, os sufixos que precisamos considerar para a produção das regras são aqueles iniciados em:* $0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33$*;*
- *Ao ordenar estes sufixos com o radix-sort, teremos:*

Sufixo	Índice	Rank
a$$$$$	(30)	1
abbabcabbaccaccabcabbabcabca$$$$$	(3)	2
abbaccaccabcabbabcabca$$$$$	(9)	2
abbabcabca$$$$$	(21)	2
abcabbabcabbaccaccabcabbabcabca$$$$$	(0)	3
abcabbaccaccabcabbabcabca$$$$$	(6)	3
abcabbabcabca$$$$$	(18)	3
abcabca$$$$$	(24)	3
abca$$$$$	(27)	3
accaccabcabbabcabca$$$$$	(12)	4
accabcabbabcabca$$$$$	(15)	4

- *A cadeia reduzida tem o formato:*

$$u^1 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline 3 & 2 & 3 & 2 & 4 & 4 & 3 & 2 & 3 & 3 & 1 \\ \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \end{array}$$

- *As regras geradas neste nível são:*

$$1 \rightarrow a \cdot \$ \cdot \$$$
$$2 \rightarrow a \cdot b \cdot b$$
$$3 \rightarrow a \cdot b \cdot c$$
$$4 \rightarrow a \cdot c \cdot c$$

❑ *Nível 1: Neste nível precisamos adicionar à cadeia de entrada o caractere sentinela* \$*, pois a janela do cálculo do LCP_{mean} é definido pelo tamanho das regras do nível anterior, que é* 3 *e* $|S^1| = 11$.

$$S^1 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 3 & 2 & 3 & 2 & 4 & 4 & 3 & 2 & 3 & 3 & 1 & \$ \\ \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \end{array}$$

- *O LCP_{mean} para essa cadeia é* 1*, como regras desse tamanho não agregam valor, iremos atribuir a x o mesmo valor do nível anterior (*3*);*
- *Com $x = 3$, os sufixos que serão considerados na construção das regras são aqueles iniciados em:* $0, 3, 6, 9$*;*
- *Ao ordenar os sufixos $i \bmod 3 = 0$ com o radix-sort temos:*

Sufixo	Índice	Rank
24432331$	(3)	1
31$	(9)	2
32324432331$	(0)	3
32331$	(6)	3

– *A cadeia reduzida tem o formato:*

$$u^2 = \begin{array}{|c|c|c|c|} \hline 3 & 1 & 3 & 2 \\ \hline \end{array}$$
$$\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}$$

– *As regras gerada neste nível são:*

$$1 \rightarrow 2 \cdot 4 \cdot 4$$
$$2 \rightarrow 3 \cdot 1 \cdot \$$$
$$3 \rightarrow 3 \cdot 2 \cdot 3$$

❑ *Apesar do tamanho do alfabeto ser menor do que o tamanho da cadeia, o* LCP_{mean} *desse nível resultou em* 1*, portanto o processo recursivo é interrompido. E* X_s *é definido como:*

$$X_s = 3 \cdot 1 \cdot 3 \cdot 2$$

Custo computacional.

A etapa de geração de regras do GCX reduz o problema a uma cadeia de tamanho $\lceil n/X \rceil$ com $1 < X < n$, todas as outras etapas podem ser feitas em tempo linear. Assim, a complexidade de tempo do GCX, é dada pela recorrência:

$$T(n) = T\left(\left\lceil \frac{n}{X} \right\rceil\right) + O(n)$$

como $x \geq 2$, então $T(n) = O(n)$.

4.1.1 Tamanho da gramática

Considere 1 como o nível inicial da gramática G e ℓ como o número total de níveis. Seja σ^j a quantidade de regras geradas no $j-$ésimo nível da recursão e $|r^j|$ o tamanho das regras nesse mesmo nível. Além disso, considere r_s^ℓ como sendo o lado direito do símbolo inicial X_s da gramática. Dessa forma, o tamanho de G pode ser expresso por meio da seguinte relação:

$$|G| = |r_s^\ell| + \sum_{j=1}^{\ell} |r^j| * \sigma^j$$

4.1.2 Detalhes de implementação

O resultado da compressão de uma cadeia S é um arquivo que contém os seguintes elementos:

1. *Cabeçalho*: um vetor na forma

$$\begin{array}{cccccccccc} \mathrm{C} = & \ell & |X_s| & \sigma^{\ell} & \sigma^{\ell-1} & \dots & \sigma^{1} & |r^{\ell}| & \dots & |r^{1}| \\ & 0 & 1 & 2 & 3 & \dots & \ell+1 & \ell+2 & \dots & (2*\ell)+2 \end{array}$$

   onde ℓ é o número de níveis de G, σ^j representa a quantidade de regras e $|r^j|$ é o tamanho das regras no $j-$ésimo nível de G;

2. *Símbolo inicial (X_s)*: cadeia reduzida gerada no último nível da recursão;

3. O *conjunto de regras* P formado por ℓ subconjuntos de regras gerados durante a recursão segue as diretrizes abaixo:

   - ❏ Os conjuntos de regras são armazenados partindo do nível ℓ até o nível 1;
   - ❏ Para cada nível j, cada produção de r_i é inserida em P^j respeitando a ordem lexicográfica dos símbolos *não-terminais* de j. Além disso, apenas o lado direito da regra é armazenado. Por exemplo, dadas as regras:

$$\begin{aligned} r_{16} &\rightarrow r_{20} \cdot r_2 \cdot r_5 \cdot r_{18} \\ r_7 &\rightarrow r_{20} \cdot r_{32} \cdot \$ \cdot \$ \\ r_4 &\rightarrow r_{45} \cdot r_2 \cdot r_{16} \cdot r_9 \end{aligned}$$

     O conjunto de regras P^j é:

$$\{\underbrace{r_{45} \cdot r_2 \cdot r_{16} \cdot r_9}_{r_4} \ , \ \underbrace{r_{20} \cdot r_{32} \cdot \$ \cdot \$}_{r_7} \ , \ \underbrace{r_{20} \cdot r_2 \cdot r_5 \cdot r_{18}}_{r_{16}}\}$$

     Armazenar as regras dessa maneira acelera o processo de decodificação, uma vez que o acesso ao lado direito de r_i pode ser feita através de uma simples operação de multiplicação.

4. O alfabeto usado para representar as regras de cada nível (com exceção do último) é dado por $\mathbb{Z}^*$.

4.2 Descompressão

A descompressão começa com a derivação das regras que compõem X_s. Ao decodificar os símbolos de X_s obtemos S^{ℓ}, que é a cadeia de entrada do nível ℓ. Esse processo é repetido para cada nível j em G, resultando na cadeia de entrada desse nível, assim, ao alcançarmos o primeiro nível da recursão, obtemos a cadeia de entrada S (Figura 9).

$$X_s \longrightarrow \boxed{\text{descomp.}^{\ell}} \longrightarrow S^{\ell} \longrightarrow \boxed{\text{descomp.}^{\ell-1}} \longrightarrow S^{\ell-1} \longrightarrow \cdots \longrightarrow \boxed{\text{descomp.}^{0}} \longrightarrow S$$

Figura 9 – Entradas e saídas produzidas pelo GCX ao longo de diferentes níveis.

O processo de descompressão segue uma sequência de operações matemáticas entre um índice k, que demarca o início das regras de produção do nível j na cadeia comprimida, e o cabeçalho C da gramática.

Como exemplo, considere o $j-$ésimo nível da recursão e a cadeia $S^j = r_2^j \cdot r_1^j \cdot r_3^j$. A decodificação de S^j ocorre mediante derivações sequenciais, realizadas da esquerda para a direita, nas regras (símbolos) de S^j. Ao decodificar r_3^j, consultamos em C o tamanho das regras de j ($|r^j|$), multiplicamos esse valor por i (i é o *lex-name/rank* atribuído a regra, nesse exemplo $i = 3$), e somamos o resultado a k. A derivação de r_3^j é dada pelos próximos $|r^j|$ caracteres na cadeia comprimida[1]. Isso é possível porque armazenamos as regras de acordo com a ordem lexicográfica dos símbolos *não-terminais*. A última etapa de decodificação do nível j consiste em atualizar o valor do índice k para $k + (\sigma^j * |r^j|)$, marcando o início da codificação das regras do nível $j - 1$.

O custo computacional para a descompressão é $\Theta(n)$.

4.3 Extração

A estratégia empregada pelo GCX para extrair uma subcadeia $S[a,b]$ de S assemelha-se ao processo de descompressão, diferindo apenas no momento de selecionar os símbolos de uma derivação que irão compor S^j. Durante a construção de S^j, ao invés de replicar integralmente todo o conteúdo derivado de uma regra, o GCX seleciona trechos específicos dessa derivação. Esse refinamento nos intervalos é aplicado exclusivamente na primeira e na última regras da cadeia no nível j, que correspondem ao intervalo inferior e superior da subcadeia, definindo a fronteira da extração. Os resultados das derivações das regras intermediárias são incorporadas integralmente à composição de S^j.

A atualização desses intervalos é computada de maneira eficiente, dado que conhecemos o tamanho das regras em cada nível, permitindo-nos determinar a quantidade de símbolos terminais (folhas) cobertos por cada regra r_i no intervalo da cadeia decodificada. Os passos executados para extração são detalhados a seguir, para isso considere $j = \ell$ e $S^j = r_{20} \cdot r_2 \cdot r_5 \cdot r_{18}$:

- Calculamos a quantidade (q^j) de folhas que os símbolos de S^j cobrem. Em outras palavras, q^j é obtido a partir do produto do tamanho das regras no nível j até o primeiro nível da recursão[2].

  Essa etapa pode ser computada uma única vez, os resultados podem ser armazenados em um vetor auxiliar e serem consultados a cada nível da recursão.

[1] Os símbolos que decodificam r_3^j estão localizados a partir da posição $k + (3 * |r^j|)$.
[2] Neste nível o tamanho das regras é sempre 1.

- Calculamos o intervalo de corte que será aplicado em r_{20} e r_{18}:

$$\delta_1 = \left(\left\lfloor \frac{a}{q} \right\rfloor\right) \ , \ \delta_2 = \left(\left\lfloor \frac{b}{q} \right\rfloor\right)$$

- Construímos a cadeia S^{j-1}, anexando os resultados das derivações de r_i na ordem em que as regras aparecem em S^j, da esquerda para a direita, nesse caso:

  1. Para r_{20}, copiamos os símbolos resultantes da derivação (r'_{20}) partindo do índice δ_1 até $|r^{j-1}|$;
  2. Para as regras r_2 e r_5 anexamos a S^{j-1} o resultado integral das derivações;
  3. Para a regra r_{18}, anexamos os símbolos resultantes da derivação (r'_{18}) partindo da posição 0 até a posição δ_2.

  Ou seja:

$$S^{j-1} = r'_{20}[\delta_1, |r^{j-1}|] \cdot r'_2[0, |r^{j-1}|] \cdot r'_5[0, |r^{j-1}|] \cdot r'_{18}[0, \delta_2]$$

- Ajustamos os valores de a e b para corresponder ao novo tamanho da cadeia coberto por S^{j-1}:

$$a = a \bmod q$$
$$b = b \bmod q$$

Esse processo é repetido até alcançarmos o nível das folhas, onde obteremos a subcadeia desejada.

O Exemplo 2 e a Figura 11 ilustram esse processo.

Exemplo 2 *A Figura 10 mostra uma árvore de regras de derivação, os nós que descendem diretamente da raiz formam o símbolo inicial (X_s) da gramática que comprime $S = abcabbabcabbaccaccabcabbabcabca\$\$$.*

Suponha que precisemos extrair a subcadeia compreendida no intervalo S[14,28]. O GCX executará os seguintes passos para isso:

Considere $a = 14$ e $b = 28$. Ao longo deste exemplo, destacamos em laranja as regras que precisarão ter o intervalo ajustado durante o processo de expansão:

- *Nível 0:*
  - $a = 14$, $b = 28$ *e* $S^3 = X_s$ *produz:*

$$X_s \rightarrow 3 \cdot 1 \cdot 3 \cdot 2$$

  - *Número máximo de folhas (q) cobertas pelos nós do nível* 1 *é dado por:*

$$|r^1| * |r^2| * |r^3| = 3 * 3 * 1 = 9;$$

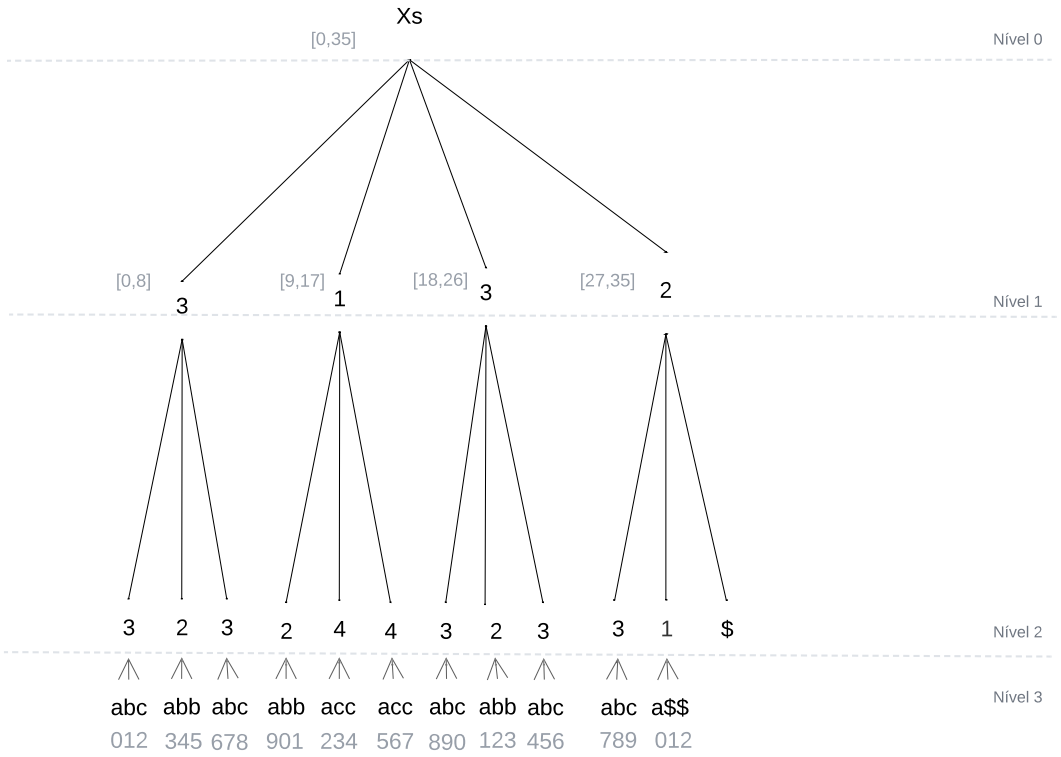


Figura 10 – Árvore de regras de produção da cadeia *abcabbabcabbaccaccabcabbabcabca*$$

– *Intervalo de corte:*

$$\delta_1 = \left\lfloor \frac{14}{9} \right\rfloor = 1 \ , \ \delta_2 = \left\lfloor \frac{28}{9} \right\rfloor = 3$$

$\therefore$ *os símbolos de* X_s *que representam* S^3 *são o símbolo da posição* 1 *de* X_s*, o último símbolo a ser anexado em* S^3 *é o símbolo da posição* 3 *de* X_s*;*

– $S^3 = 1 \cdot 3 \cdot 2$

– *Ajuste de intervalo:*

$$a = 14 \mod 9 = 5 \ , \ b = 28 \mod 9 = 1$$

❑ *Nível 1:*

– $a = 5$, $b = 1$ *e* $S^3 = 1 \cdot 3 \cdot 2$*;*

$$1 \rightarrow 2 \cdot 4 \cdot 4$$
$$2 \rightarrow 3 \cdot 1 \cdot \$$$
$$3 \rightarrow 3 \cdot 2 \cdot 3$$

- $q = |r^2| * |r^3| = 3 * 1 = 3$;
- *Intervalo de corte:*

$$\delta_1 = \left\lfloor \frac{5}{3} \right\rfloor = 1 \ , \ \delta_2 = \left\lfloor \frac{1}{3} \right\rfloor = 0$$

$\therefore$ *ao derivar* S^3 *anexaremos a* S^2 *a derivação do símbolo* 1 *a partir da posição* 1*; a derivação de* 3 *será anexada a* S^2 *na íntegra, e somente o símbolo da posição* 0 *resultante da derivação do símbolo* 2 *será anexado em* S^2.

- $S^2 = 4 \cdot 4 \cdot 3 \cdot 2 \cdot 3 \cdot 3$
- *Ajuste de intervalo:*

$$a = 5 \mod 3 = 2 \ , \ b = 1 \mod 3 = 1$$

❑ *Nível 2:*

- $a = 2$, $b = 1$ *e* $S^2 = 4 \cdot 4 \cdot 3 \cdot 2 \cdot 3 \cdot 3$;

$$2 \rightarrow a \cdot b \cdot b$$
$$3 \rightarrow a \cdot b \cdot c$$
$$4 \rightarrow a \cdot c \cdot c$$

- $q = 1$;
- *Intervalo de corte:*

$$\delta_1 = \left\lfloor \frac{2}{1} \right\rfloor = 2 \ , \ \delta_2 = \left\lfloor \frac{1}{1} \right\rfloor = 1$$

- $S^1 = c \cdot a \cdot c \cdot c \cdot a \cdot b \cdot c \cdot a \cdot b \cdot b \cdot a \cdot b \cdot c \cdot a \cdot b$
- *Ajuste de intervalo:*

$$a = 2 \mod 1 = 0 \ , \ b = 1 \mod 1 = 0$$

❑ *Nível 3:*

- $S^1 = c \cdot a \cdot c \cdot c \cdot a \cdot b \cdot c \cdot a \cdot b \cdot b \cdot a \cdot b \cdot c \cdot a \cdot b$

  Neste ponto, não é mais possível realizar derivações adicionais, o processo recursivo é encerrado, resultando em

$$S^0 = caccabcabbabcab$$

  como resposta final.

O percurso realizado pelo GCX neste exemplo para descompactar $S[14,28]$ *é ilustrado pela Figura 11.*

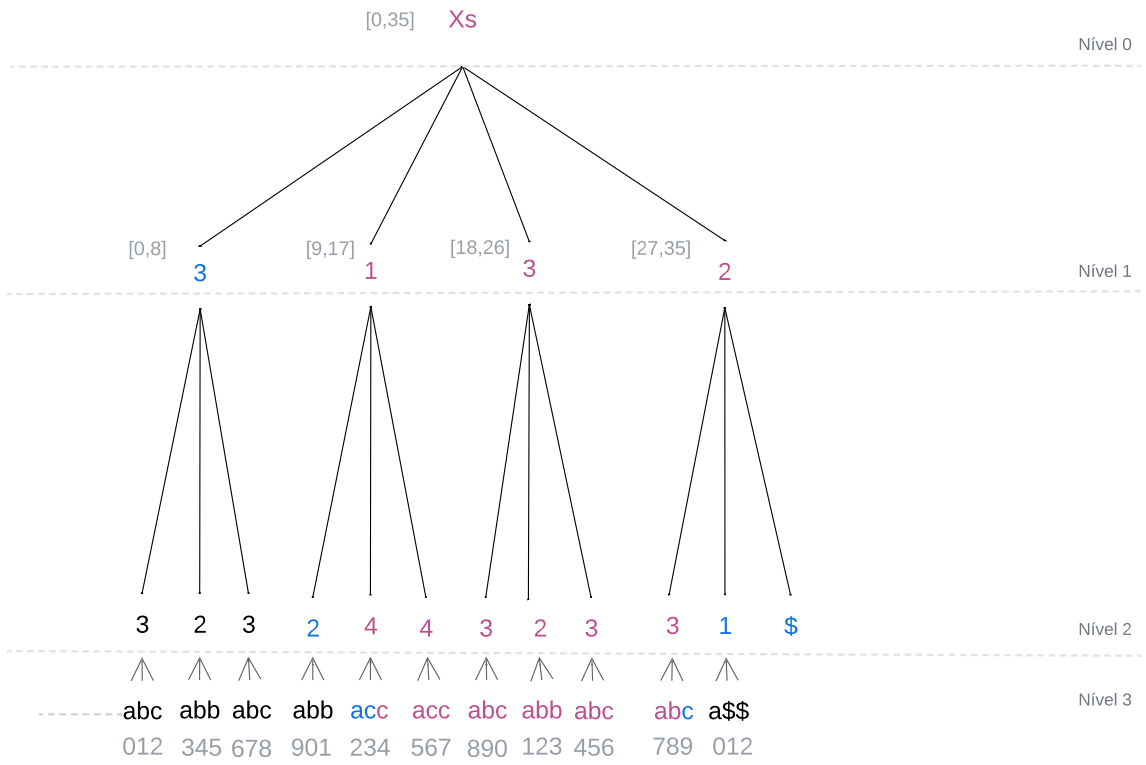


Figura 11 – Extração de $S[14{,}28]$. Em rosa evidenciamos os símbolos selecionados em cada derivação. Os símbolos em azul representam os trechos descartados em uma regra r_i durante a extração.

Codificação

As regras da gramática gerada pelo GCX são salvas em ordem crescente de ranking e são armazenadas usando $\lceil \log_2(\sigma^j) \rceil + 1$ bits, onde σ^j é o tamanho do alfabeto da cadeia de entrada do nível j.

Capítulo 5

Experimentos e Análise dos Resultados

5.1 Método de validação

Neste capítulo apresentamos uma análise de desempenho com base em um conjunto de medidas de eficiência que incluem taxa de compressão, tempo de compressão, descompressão e extração, além de consumo de memória. Conduzimos diversos experimentos práticos com o GCX, comparando os resultados com execuções equivalentes do GCIS (consideramos as codificações *Elias-Fano* e *Simple8b* para o GCIS[1]) e do RePair.

O GCX foi compilado com a versão 14 do `C++`, utilizando a flag `-O3`. A compilação do GCIS seguiu os passos disponíveis em <https://github.com/danielsaad/GCIS/>. Ambos os algoritmos foram compilados com a biblioteca `malloc_count` para medir o consumo de memória.

Em relação ao conjunto de dados, utilizamos coleções de textos repetitivos e não repetitivos explorados nos testes de desempenho do artigo de Nunes et al. (2022). As Tabelas 1 e 2 mostram os arquivos selecionados para os testes (em ordem crescente de tamanho).

O código-fonte do GCX está disponível em <https://github.com/DanyelleAngelo/GCX/>.

5.1.1 Configurações de ambiente

Os testes de desempenho foram executados em uma máquina com as seguintes configurações:

- ❏ Memória RAM: 128GB
- ❏ CPU: Intel Core i7-10700KF CPU @ 3.80GHz
- ❏ Sistema operacional:

[1] A partir de agora iremos nos referir as execuções do GCIS utilizando *Elias-Fano* e *Simple8b* como GCIS-ef e GCIS-s8b respectivamente.

Textos repetitivos	Tam. (MB)	Tipo	Fonte
world_leaders	46.97	real - líderes mundiais da CIA	CIA (CHILI, 2010c)
einstein.de.txt	92.76	real -texto em Alemão	Wikipedia (CHILI, 2010c)
dblp.xml.00001.1	104.86	pseudo-real - XML de bibliografia	DBLP (CHILI, 2010b)
dblp.xml.00001.2	104.86	pseudo-real - XML de bibliografia	DBLP (CHILI, 2010b)
dblp.xml.0001.1	104.86	pseudo-real - XML de bibliografia	DBLP (CHILI, 2010b)
dblp.xml.0001.2	104.86	pseudo-real - XML de bibliografia	DBLP (CHILI, 2010b)
dna.001.1	104.86	pseudo-real - DNA	Gutenberg Project (CHILI, 2010b)
english.001.2	104.86	pseudo-real - texto em inglês	Gutenberg Project(CHILI, 2010b)
proteins.001.1	104.86	pseudo-real - proteínas	Swissprot database (CHILI, 2010b)
sources.001.2	104.86	pseudo-real - código fonte	Linux and GCC (CHILI, 2010b)
Escherichia_Coli	112.69	real - DNA de bactéria	NCBI (CHILI, 2010c)
influenza	154.81	real - DNA de bactéria	NCBI (CHILI, 2010c)
coreutils	205.28	real - código fonte	GNU package (CHILI, 2010c)
kernel	257.96	real - código fonte	Linux Kernel (CHILI, 2010c)
fib41	267.91	artificial	Pizza & Chili (CHILI, 2010a)
rs.13	267.75	artificial	Pizza & Chili (CHILI, 2010a)
tm29	268.44	artificial	Pizza & Chili (CHILI, 2010a)
para	429.27	real - DNA	SGRP(CHILI, 2010c)
cere	461.29	real - DNA	SGRP(CHILI, 2010c)
einstein.en.txt	467.63	real - texto em inglês	Wikipedia (CHILI, 2010c)

Tabela 1 – Textos repetitivos.

Textos não-repetitivos	Tam. (MB)	Tipo	Fonte
dickens	10.19	trabalho literário	Project Gutenberg (TECHNOLOGY,)
archive	27.07	fórum de discussão do linux	Samba project (PROJECT, 1998)
nci	33.55	estrutura molecular	NCI (TECHNOLOGY,)
howto	39.42	documentação de hardware	UPO(MANZINI, 2003)
webster	41.46	1913 Webster Dictionary	Project Gutenberg (TECHNOLOGY,)
etext99	105.28	texto em inglês	UPO(MANZINI, 2003)
sprot34.dat	109.62	detalhes da proteína 104K_THEPA	UPO(MANZINI, 2003)
rctail96	114.71	notícias no formato XML	UPO(MANZINI, 2003)
sources	210.87	código fonte	Pizza & Chili (CHILI, 2005b)
dblp.xml	296.14	XML com referências bibliográficas	Pizza & Chili (CHILI, 2005c)
dna	403.93	sequência de DNA	Pizza & Chili (CHILI, 2005a)

Tabela 2 – Textos não-repetitivos.

5.2 Experimentos

Construímos 2 versões para o GCX. A primeira usa um valor fixo para X em todos os níveis da recursão, significando que todas as regras da gramática têm o mesmo comprimento. Além disso o processo recursivo só é interrompido quando todas as subcadeias do texto forem distintas[2]. Essa versão será referida como GC3, GC4, GC5, etc., e coletivamente como GC$*$. Durante os testes de desempenho do GC$*$, experimentamos 2 tipos de codificação para a gramática gerada: uma codificação fixa de 32 bits e outra codificação usando $\lceil \log_2(\sigma^j) \rceil + 1$ bits. Como esperado essa última codificação implicou em uma redução significativa no tamanho da codificação da gramática gerada.

Embora o GC$*$ tenha alcançado os resultados esperados no que diz respeito ao tempo

[2] O número de regras do nível j é igual ao tamanho do texto reduzido.

de extração em comparação ao GCIS, assim como notáveis desempenhos nas demais operações, observamos que as taxas de compressão ainda se mantiveram consideravelmente altas. Além disso não foi possível estabelecer uma relação de causalidade entre o tamanho dessas regras e as taxas de compressão. Por exemplo, para alguns conjuntos de dados, atribuir o valor 8 ao parâmetro X têm como resultado taxas de compressão similares às do GCIS, enquanto para outros conjuntos esse valor não se mostra a melhor escolha.

Com isso, desenvolvemos a versão final do GCX, conforme apresentada no Capítulo 4. Esta versão incorpora uma heurística que atualiza o valor de X a cada chamada recursiva. Esta variação é feita através do cálculo do LCP_{mean} entre todas as subcadeias distintas do texto, dessa forma é possível estimar a taxa de repetitividade do texto. Além disso, o processo recursivo foi ajustado para ser interrompido quando o LCP_{mean} calculado for igual a 1, o que reduz a quantidade de regras armazenadas na versão compactada do texto, resultando em melhorias significativas na taxa de compressão.

5.3 Resultados

5.3.1 Compressão e descompressão

Textos repetitivos

Para textos repetitivos, o GCX apresentou melhor desempenho em tempo de compressão para a maioria dos conjuntos de dados, com exceção dos conjuntos do tipo artificial (formados por `rs.13`, `tm29`, `fib41`). Os tempos médios despendidos pela operação de compressão foram de 8,72 segundos para o GCX, 9,79 segundos para o GCIS-ef, 9,48 segundos para GCIS-s8b e 59,49 segundos para o RePair. As Figuras 12-14 apresentam o tempo de compressão para textos repetitivos, considerando execuções do GCX (barra em azul claro), do GC* (barra em azul escuro), do GCIS (linha tracejada e pontilhada) e do RePair (linha tracejada em verde). Em média, o GCX é 1,08 vezes mais rápido que o GCIS-s8b para comprimir, e 6,8 vezes mais rápido que o RePair.

Em relação à operação de descompressão, o GCX obteve melhor desempenho para todos os conjuntos de dados. Em alguns casos, observamos tempos de descompressão próximos a 0. O tempo médio de descompressão para os algoritmos analisados foi de 0,41 segundos, 5,03 segundos, 1,91 segundos e 1,023 segundos para o GCX, CGIS-ef, GCIS-s8b e RePair, respectivamente, conforme mostrado nas Figuras 15-17.

Textos não-repetitivos

O tempo médio necessário para comprimir textos não-repetitivos foi de 5,28 segundos para o GCX, 9,56 segundos para o GCIS-ef, 7,54 segundos para o GCIS-s8b e 139,66 segundos para o RePair. Se comparado ao GCIS, o pior desempenho do GCX foi ao

comprimir o conjunto de dados `nci`. Em contrapartida, para os demais conjuntos de dados, o GCX apresentou tempo de compressão até 1,42 vezes menor do que a do GCIS.

O resultado da análise de descompressão mostrou que, assim como nos conjuntos de dados repetitivos, o GCX requer menos tempo para descomprimir qualquer conjunto de dados. A média de tempo necessária para descompressão foi de 0,36 segundos, 5,80 segundos, 1,80 segundos e 0,62 segundos para o GCX, GCIS-ef, GCIS-s8b e RePair, respectivamente.

As Figuras 18-19 apresentam a velocidade de compressão para textos não-repetitivos, considerando execuções do GCX, GC*, e do GCIS. Da mesma forma, as Figuras 20-21 mostram o tempo de descompressão para as mesmas entradas, utilizando esses mesmos algoritmos.

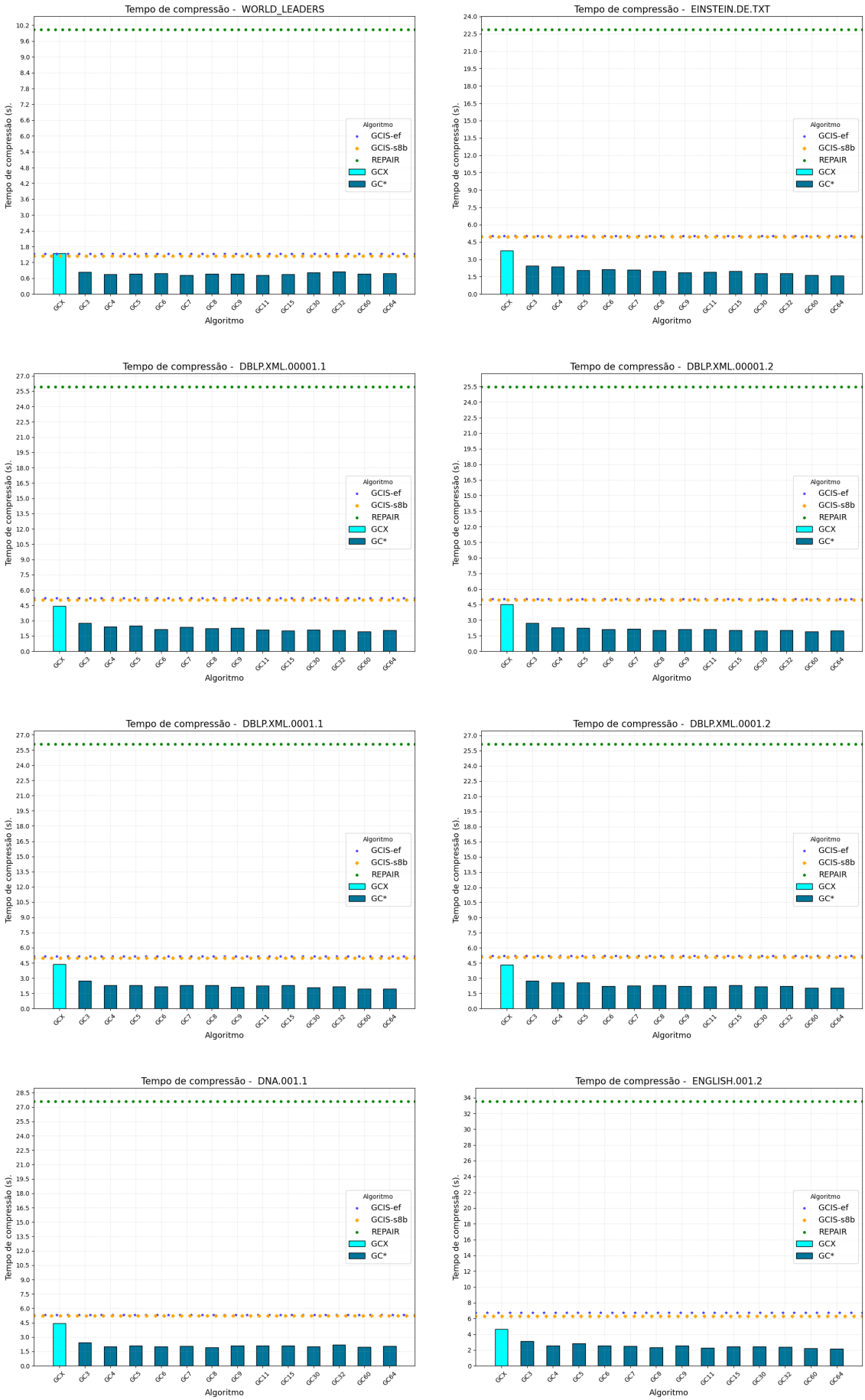


Figura 12 – Tempo de compressão para textos repetitivos.

Figura 13 – Tempo de compressão para textos repetitivos.

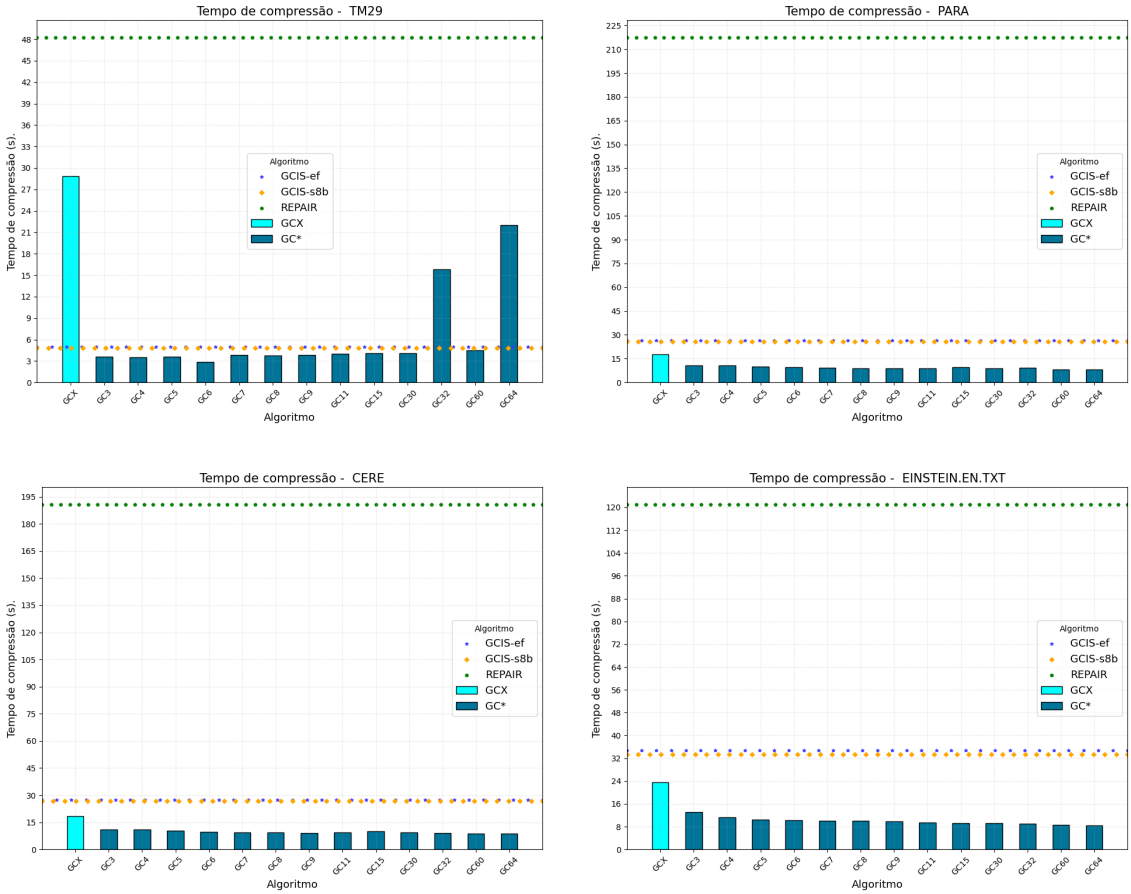


Figura 14 – Tempo de compressão para textos repetitivos.

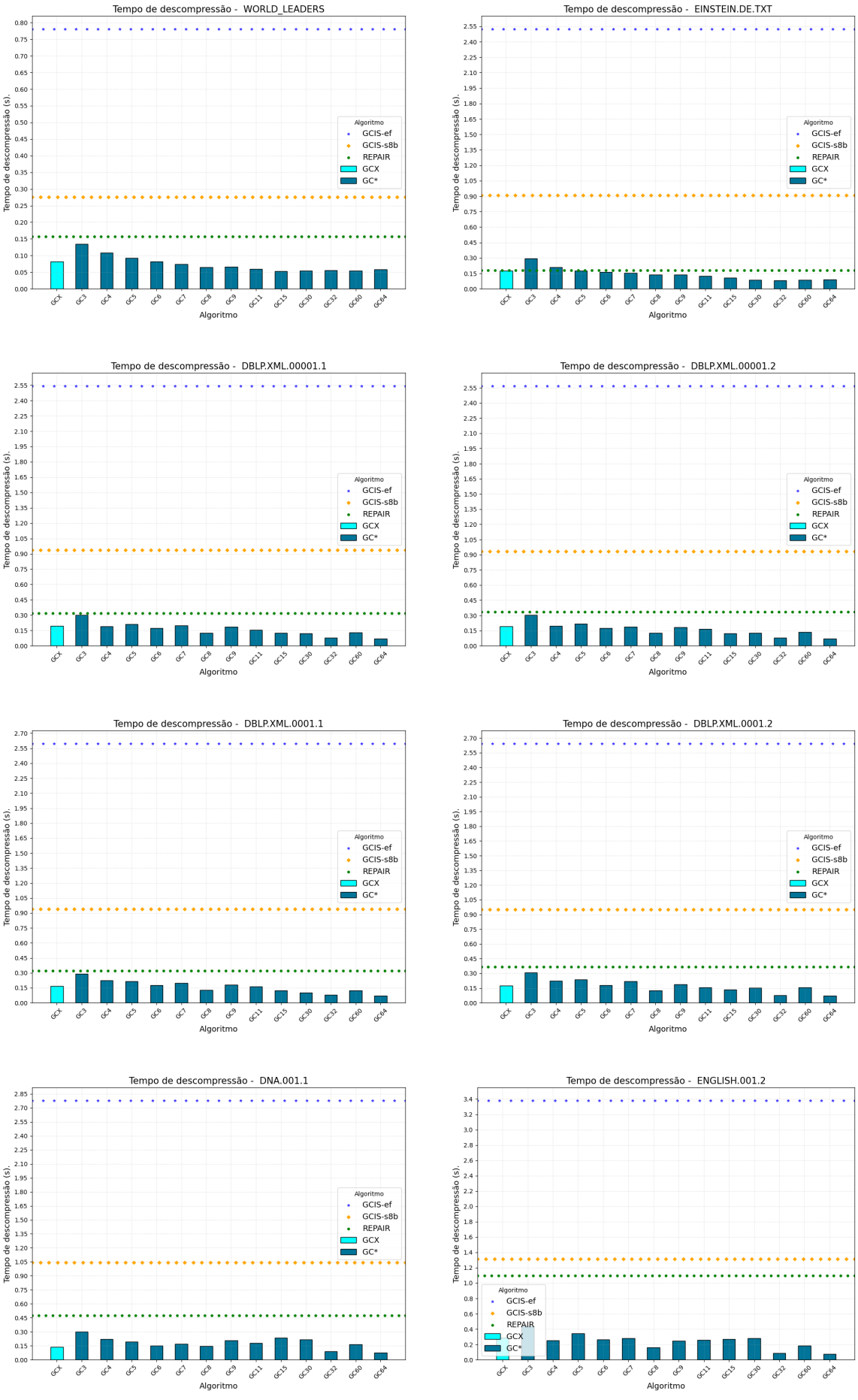


Figura 15 – Tempo de descompressão para textos repetitivos.

Figura 16 – Tempo de descompressão para textos repetitivos.

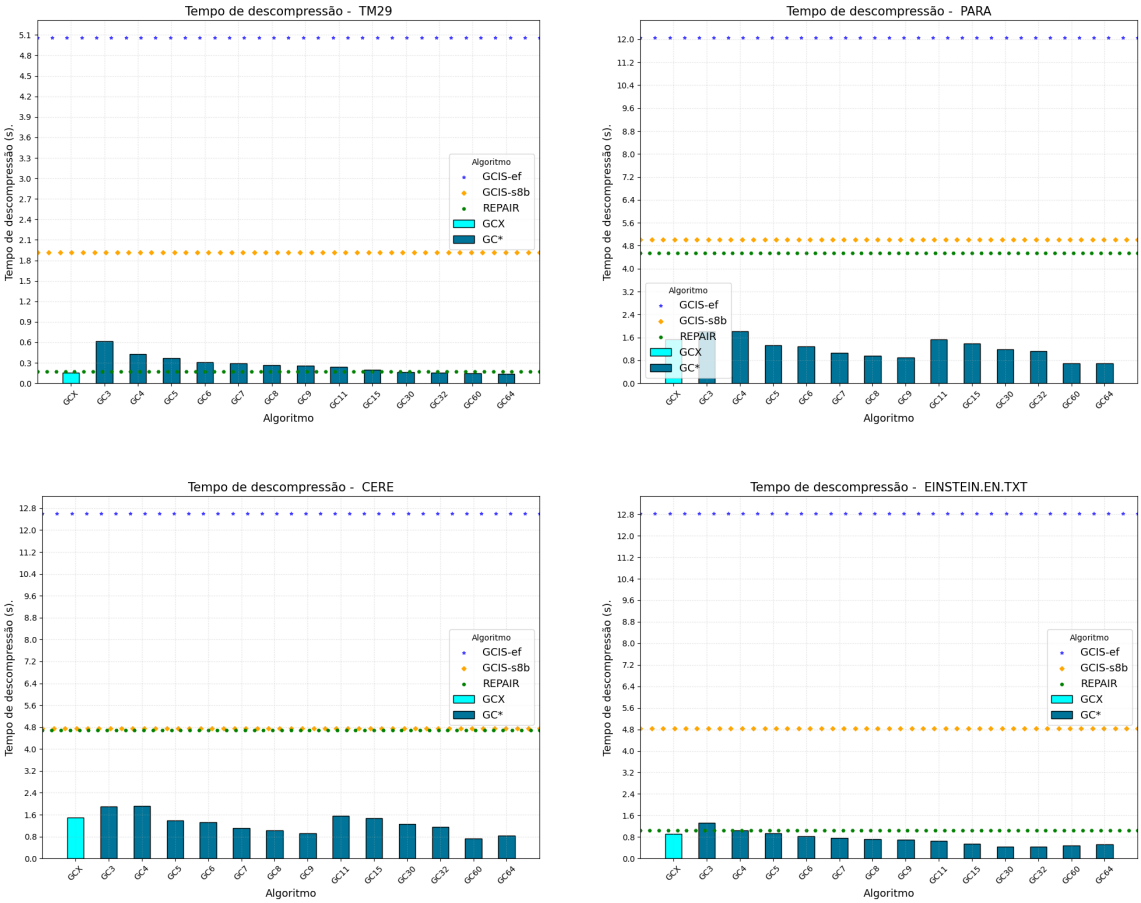


Figura 17 – Tempo de descompressão para textos repetitivos.

Figura 18 – Tempo de compressão para textos não-repetitivos.

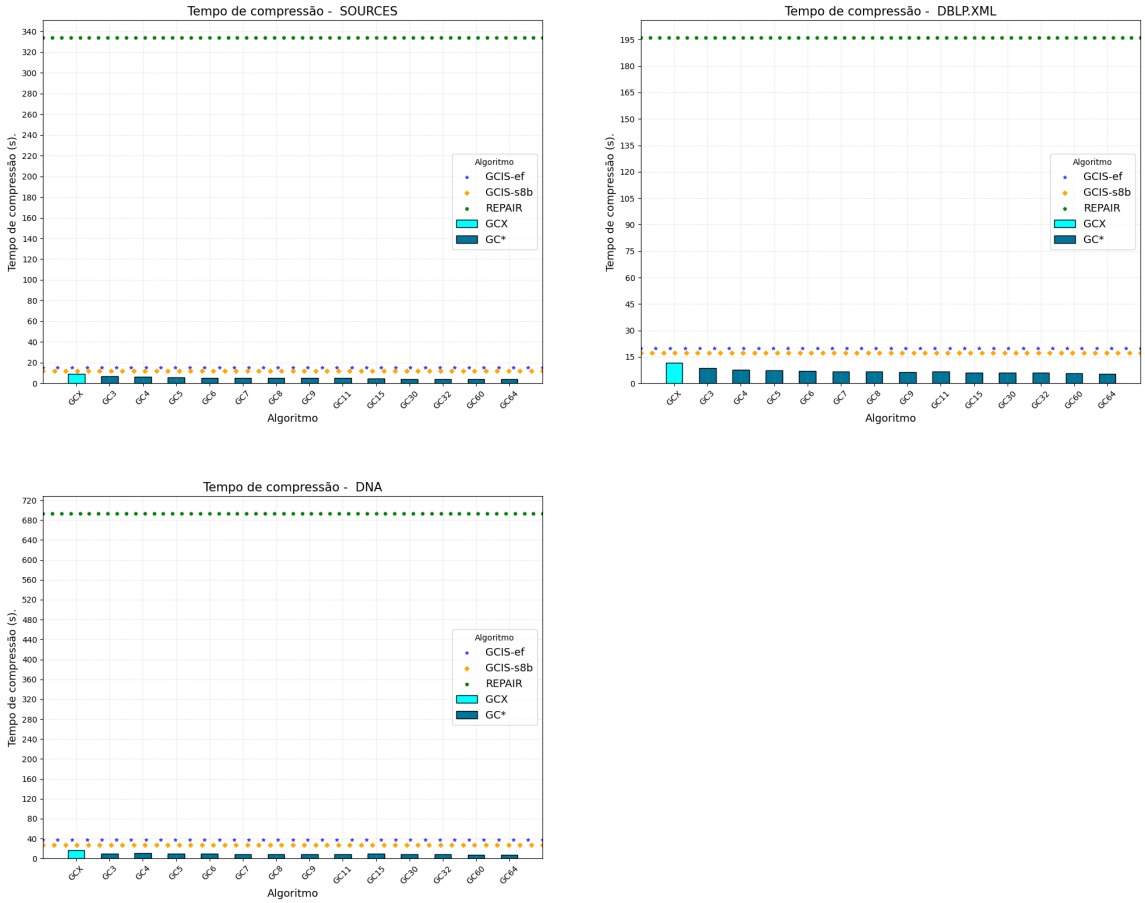


Figura 19 – Tempo de compressão para textos não-repetitivos.

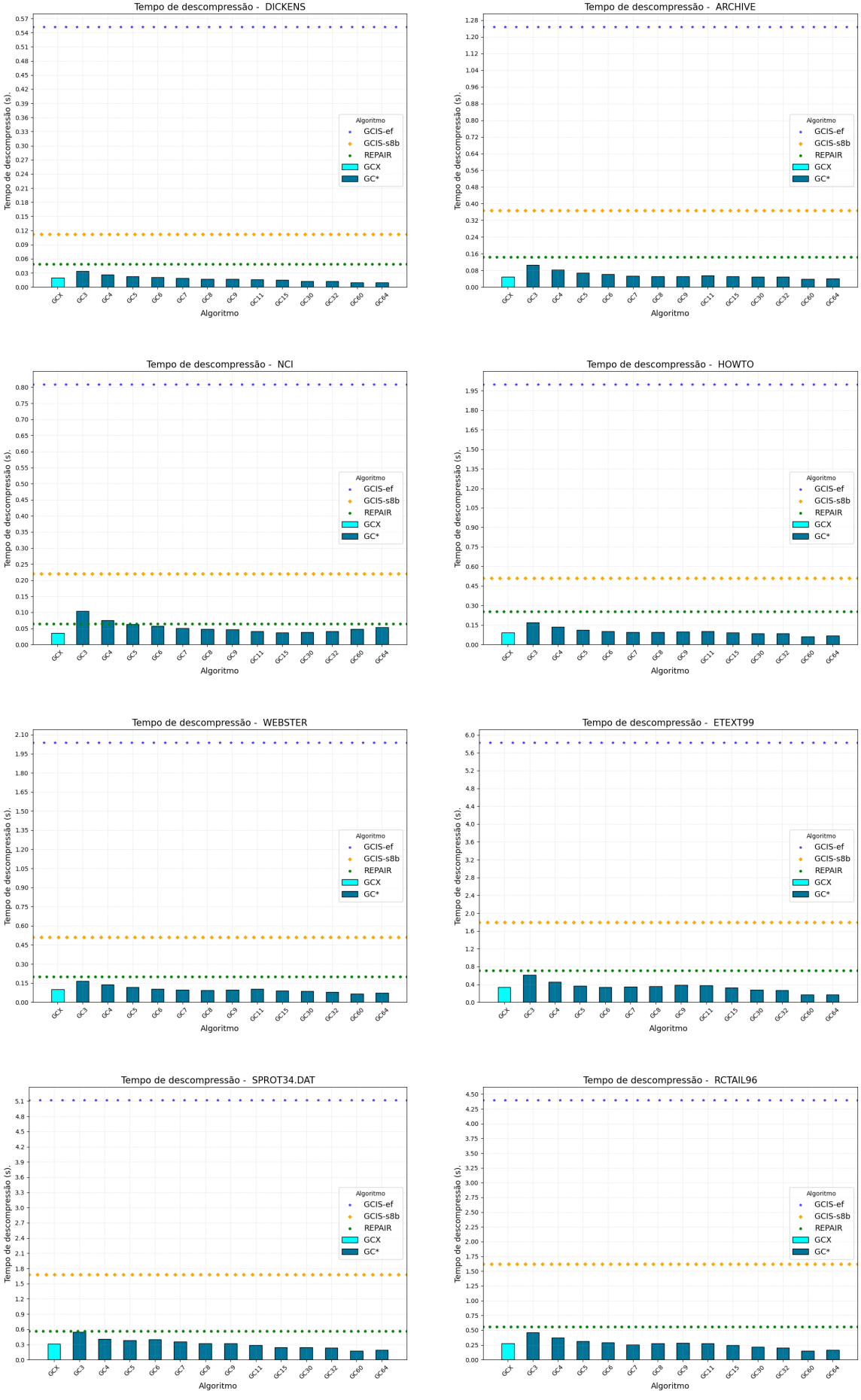


Figura 20 – Tempo de descompressão para textos não-repetitivos.

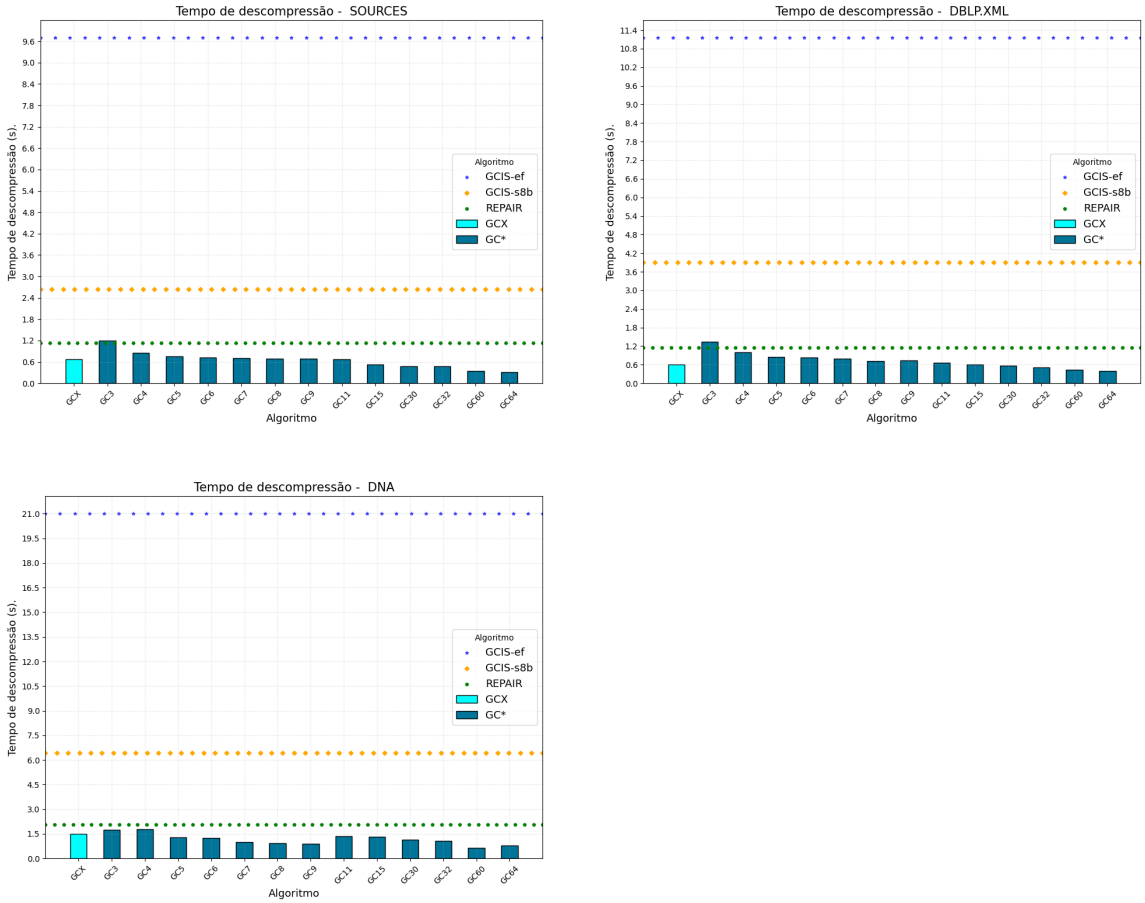


Figura 21 – Tempo de descompressão para textos não-repetitivos.

5.3.2 Taxa de compressão

A taxa de compressão de um texto é definida como a razão entre o tamanho do texto comprimido e o texto original.

Textos repetitivos

Na maioria dos casos, a taxa de compressão alcançada pelo GCX e pelo GC* foi pior do que a taxa do GCIS, com o GCX comprimindo cerca de 7 vezes menos do que o GCIS-s8b e 12 vezes menos do que o RePair. A taxa média de compressão resultante das nossas análises foi de 21,49% para o GCX, 3,34%, 2,88% e 1,70% para o GCIS-ef, GCIS-s8b e RePair respectivamente. Entre todos os conjuntos de dados analisados, apenas os conjuntos `fib41` e `rs.13` tiveram resultados de taxa de compressão semelhantes ao do GCIS. As Figuras 22-24 mostram as taxas de compressão alcançadas por cada algoritmo ao comprimir textos repetitivos.

Textos não-repetitivos

Para textos não-repetitivos, as taxas do GCX e dos GC* estão mais próximas do GCIS. A taxa média de compressão utilizando o GCX foi de 61,72%, enquanto para o GCIS-ef essa taxa foi de 46,73%, para o GCIS-s8b a taxa média de compressão foi de 35,82% e para o RePair 29,69%. Entre os conjuntos de dados analisados, apenas os conjuntos `dikens` e `dna` demonstraram um desempenho satisfatório para o GCX, com taxas de compressão limitadas superiormente pelo GCIS. Esses resultados são mostrados nas Figuras 25-26.

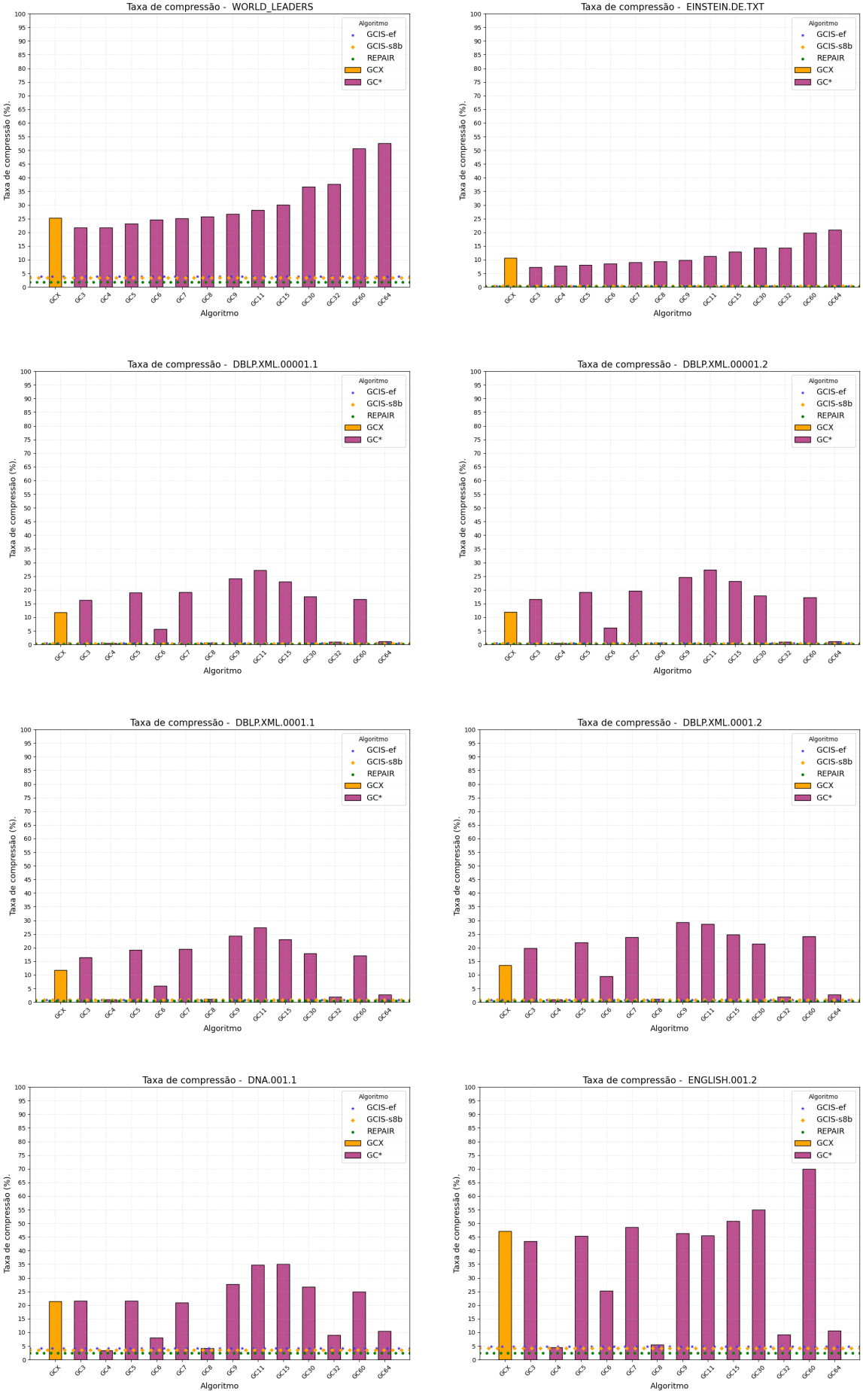


Figura 22 – Taxa de compressão para textos repetitivos.

Figura 23 – Taxa de compressão para textos repetitivos.

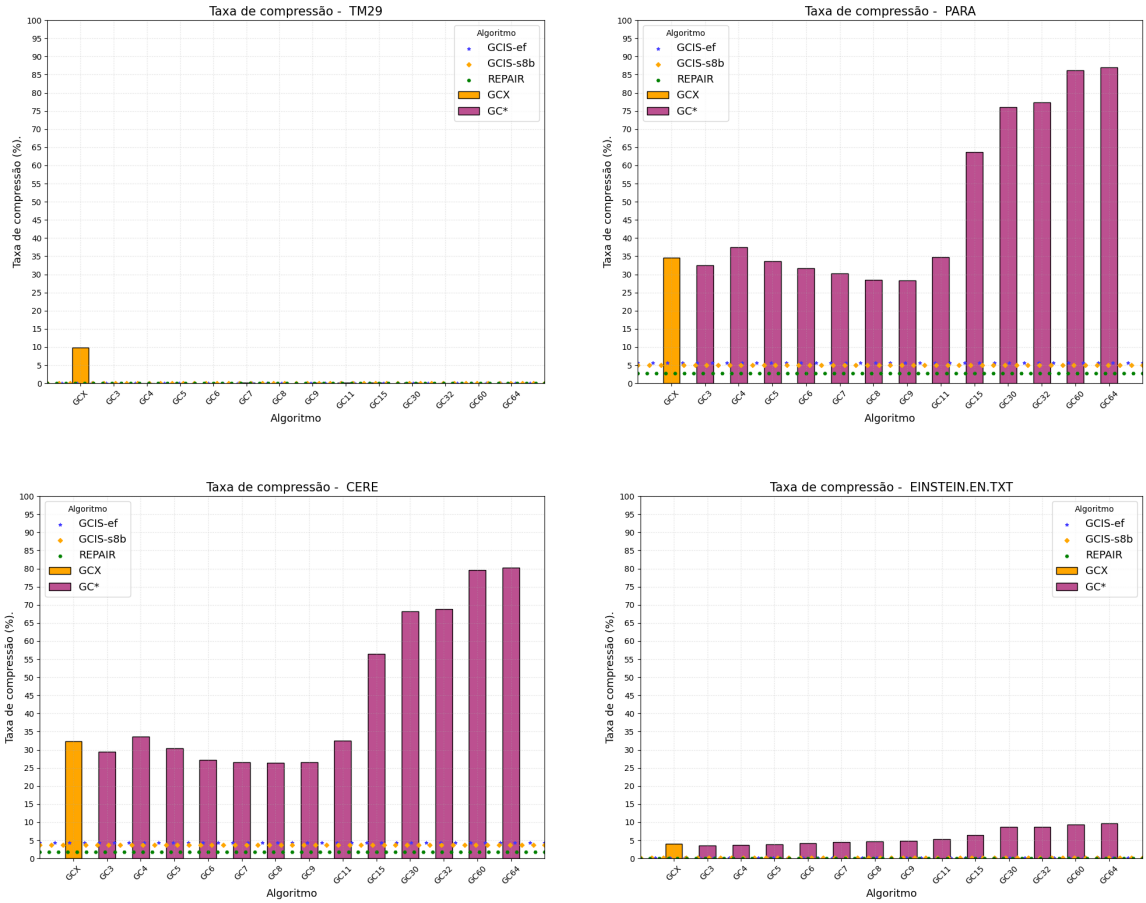


Figura 24 – Taxa de compressão para textos repetitivos.

Figura 25 – Taxa de compressão para textos não-repetitivos.

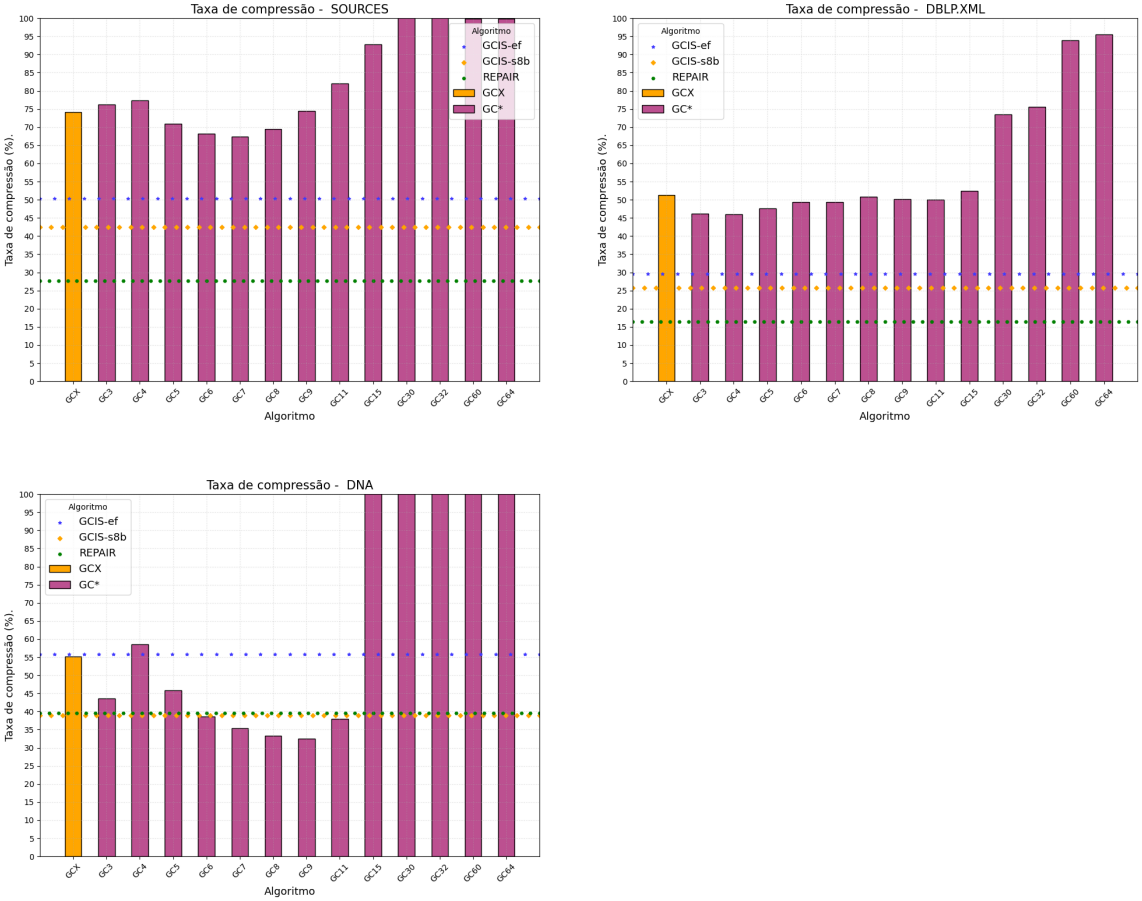


Figura 26 – Taxa de compressão para textos não-repetitivos.

5.3.3 Memória

Textos repetitivos

Quanto ao consumo de memória, observou-se que o GCIS utiliza, em média, aproximadamente 1,6 vezes menos memória do que o GCX e 5,59 vezes menos memória do que o RePair durante a operação de compressão. Os experimentos revelaram que o GC* também utiliza mais memória que o GCIS. A média do pico de consumo de memória para a operação de compressão foi de 1530 MB, 958 MB, 952 MB e 5334 MB, para o GCX, GCIS-ef, GCIS-s8b e RePair, respectivamente. Já para a operação de descompressão, a média foi de 1,45 KB, 0,91 KB, 0,90 KB e 5 KB. Os resultados dessa análise são apresentados nas Figuras 27-29.

Textos não-repetitivos

Os resultados do consumo de memória para textos não-repetitivos foram semelhantes aos de textos repetitivos. O GCX consumiu 1,43 vezes mais memória que o GCIS e 3,41 vezes menos memória que o RePair. O pico médio de consumo foi de 1047 MB, 773 MB, 729 MB e 3573 MB para o GCX, GCIS-ef, GCIS-s8b e RePair, respectivamente. Para a descompressão, os valores médios foram de 0,99 KB, 0,73 KB, 0,69 KB e 3,4 KB para o GCX, GCIS-ef, GCIS-s8b e RePair, respectivamente. As Figuras 33-36 mostram o consumo de memória durante o processo de compressão e descompressão.

Figura 27 – Consumo de memória durante a compressão de textos repetitivos.

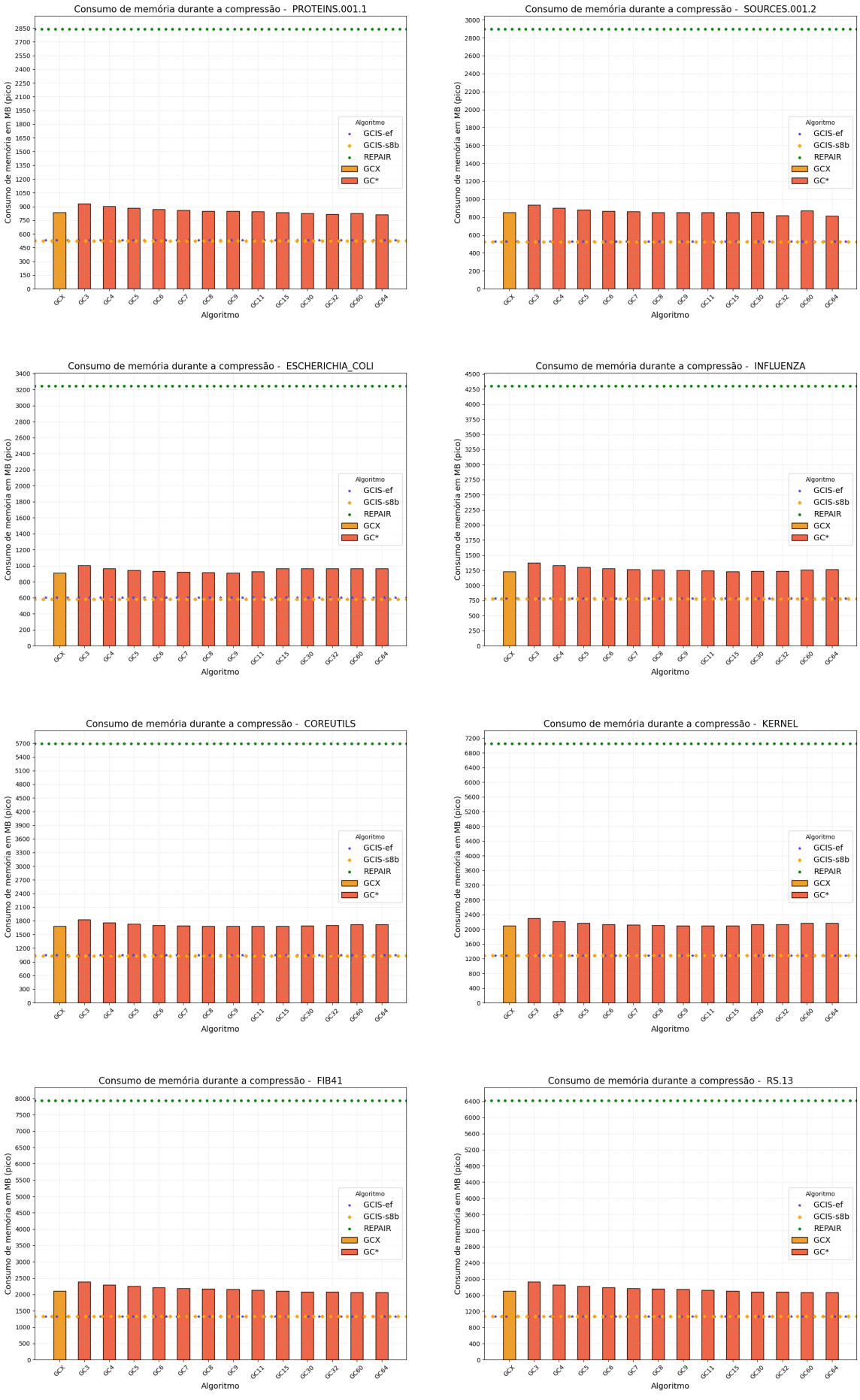


Figura 28 – Consumo de memória durante a compressão de textos repetitivos.

Figura 29 – Consumo de memória durante a compressão de textos repetitivos.

Figura 30 – Consumo de memória durante a descompressão de textos repetitivos.

Figura 31 – Consumo de memória durante a descompressão de textos repetitivos.

Figura 32 – Consumo de memória durante a descompressão de textos repetitivos.

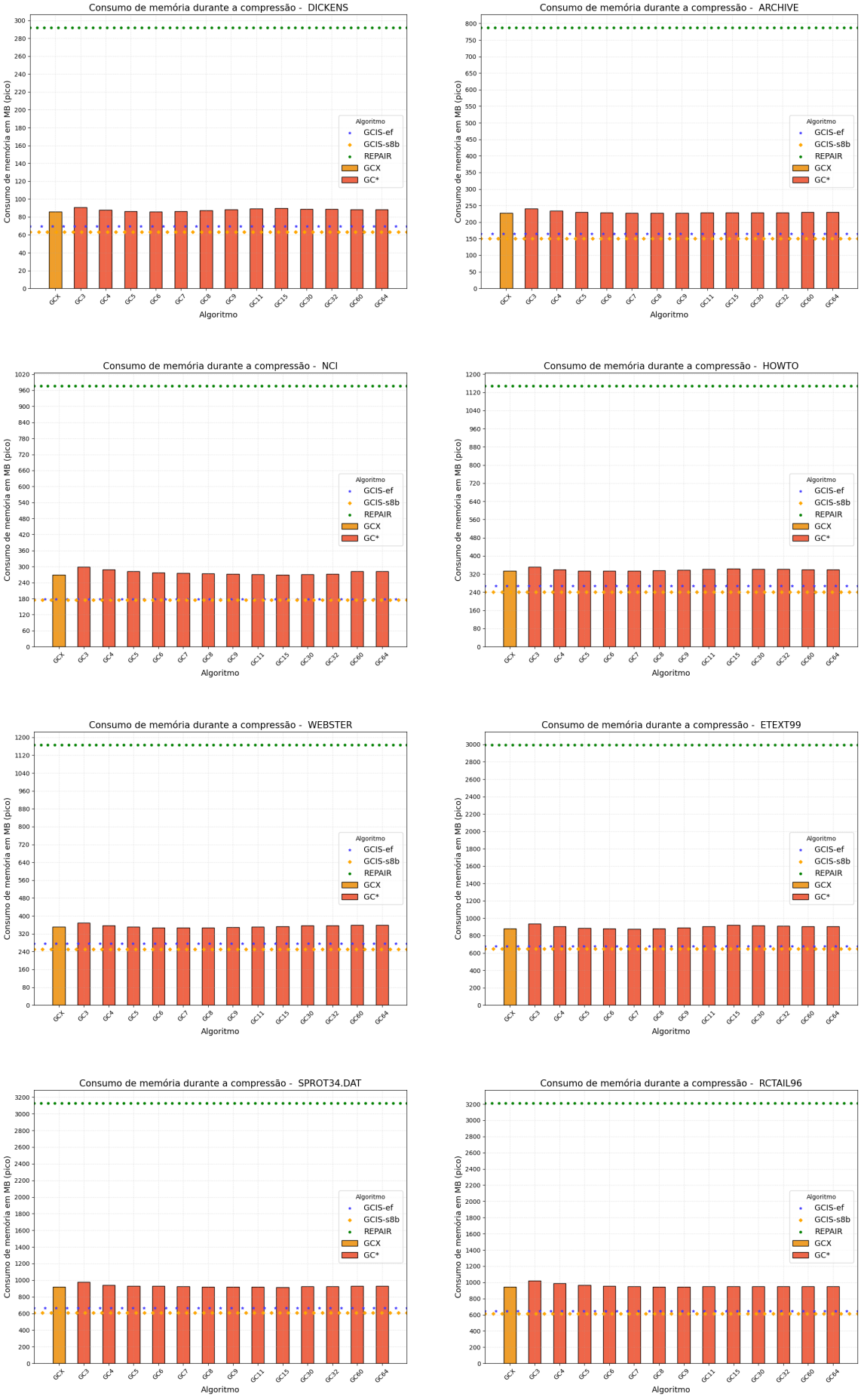


Figura 33 – Consumo de memória durante a compressão de textos não-repetitivos.

Figura 34 – Consumo de memória durante a compressão de textos não-repetitivos.

Figura 35 – Consumo de memória durante a descompressão de textos não-repetitivos.

Figura 36 – Consumo de memória durante a descompressão de textos não-repetitivos.

5.3.4 Extração

Para realizar extração de subcadeias a partir da gramática gerada pelo RePair utilizamos o algoritmo de extração SLP(GAGIE et al., 2020). Nossos testes consideraram as seguintes codificações:

- ❏ PlainSlp_32Fblc: utiliza 32 bits pra representar um vetor.
- ❏ PlainSlp_FblcFblc: utiliza o menor número de bits necessários para representar o maior valor em um vetor da gramática.

Realizamos a operação de extração de subcadeias aleatórias de comprimento 1, 10, 100, 1.000 e 10.000. Os resultados de nossos experimentos demonstram que o GCX é capaz de extrair subcadeias de qualquer tamanho a partir do texto comprimido, apresentando tempo de execução menor do que o do GCIS-ef[3] e do RePair. O tempo médio de extração para os conjuntos de dados foi de 27,09 microssegundos para o GCX, 2635,79 microssegundos para o GCIS-ef e 94,56 microssegundos para o RePair utilizando a codificação PlainSlp_32Fblc[4], respectivamente. Ou seja o GCX foi cerca de 97,29 vezes mais rápido que o GCIS e 3,67 vezes mais rápido que o RePair (utilizando a codificação PlainSlp_32Fblc).

As Figuras 37-41 evidenciam o desempenho bem-sucedido do GCX.

[3] GCIS-s8b não fornece suporte a extração de subcadeias diretamente do texto comprimido.

[4] Não foi possível realizar a extração de subcadeias do texto não-repetitivo *dna* utilizando a codificação PlainSlp_FblcFblc

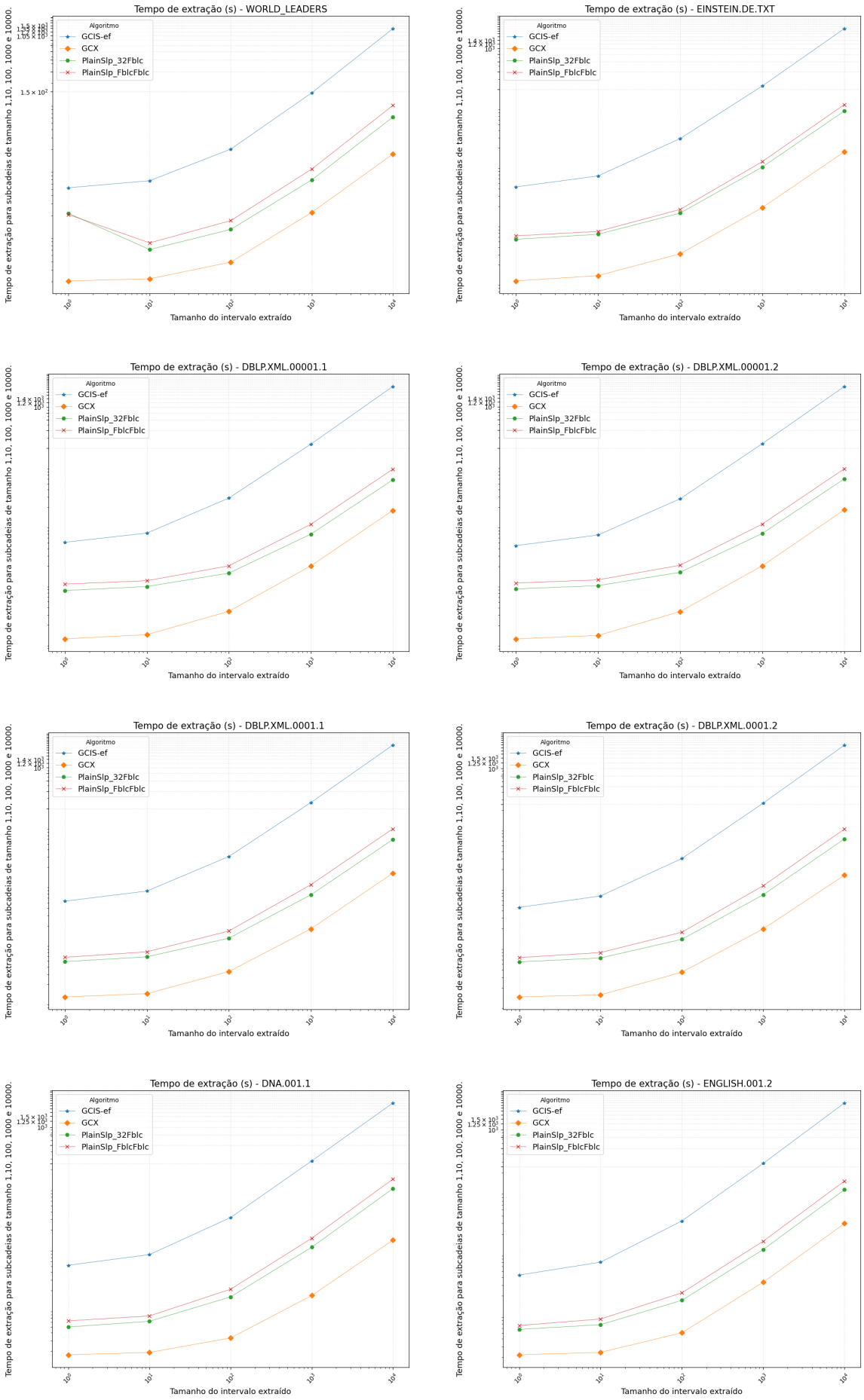


Figura 37 – Tempo extração para textos repetitivos.

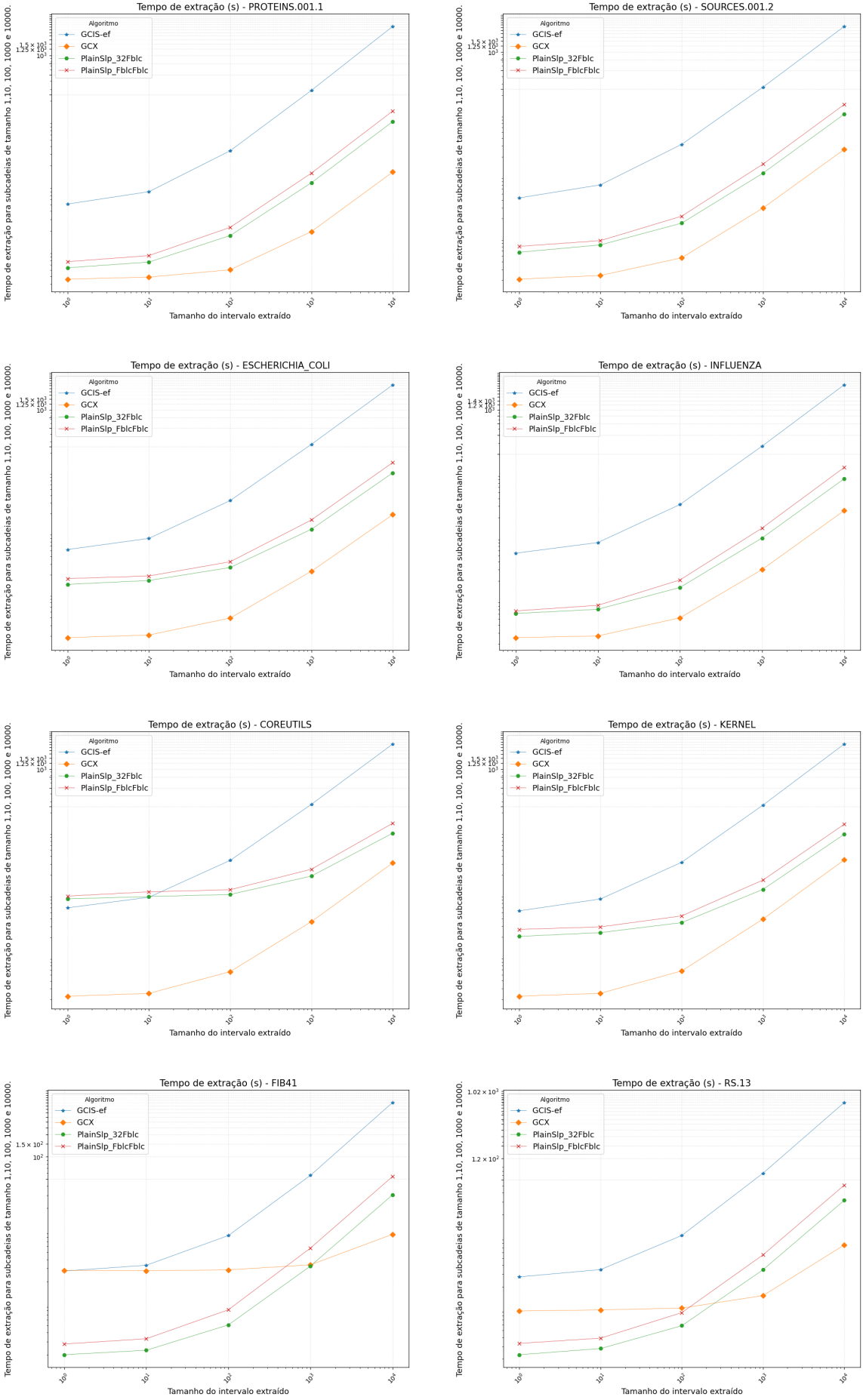


Figura 38 – Tempo de extração para textos repetitivos

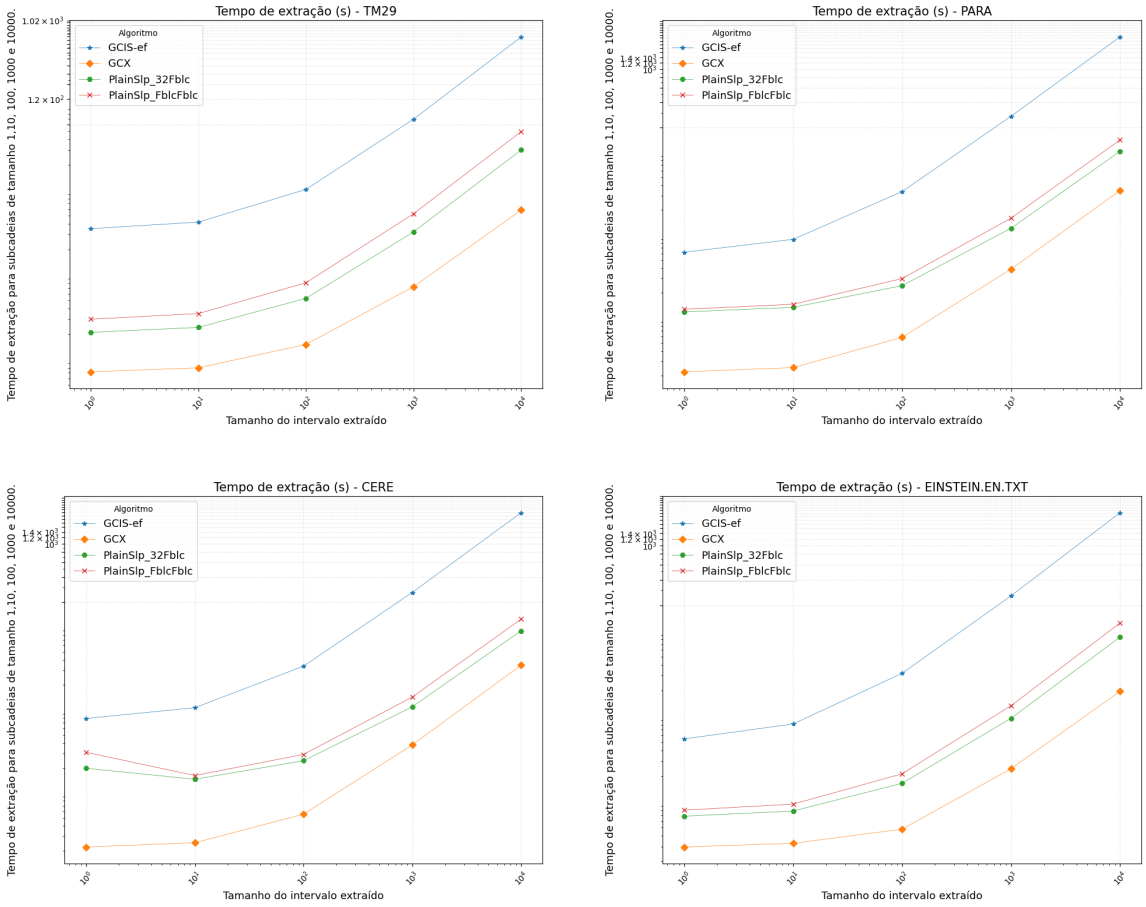


Figura 39 – Tempo de extração para textos repetitivos

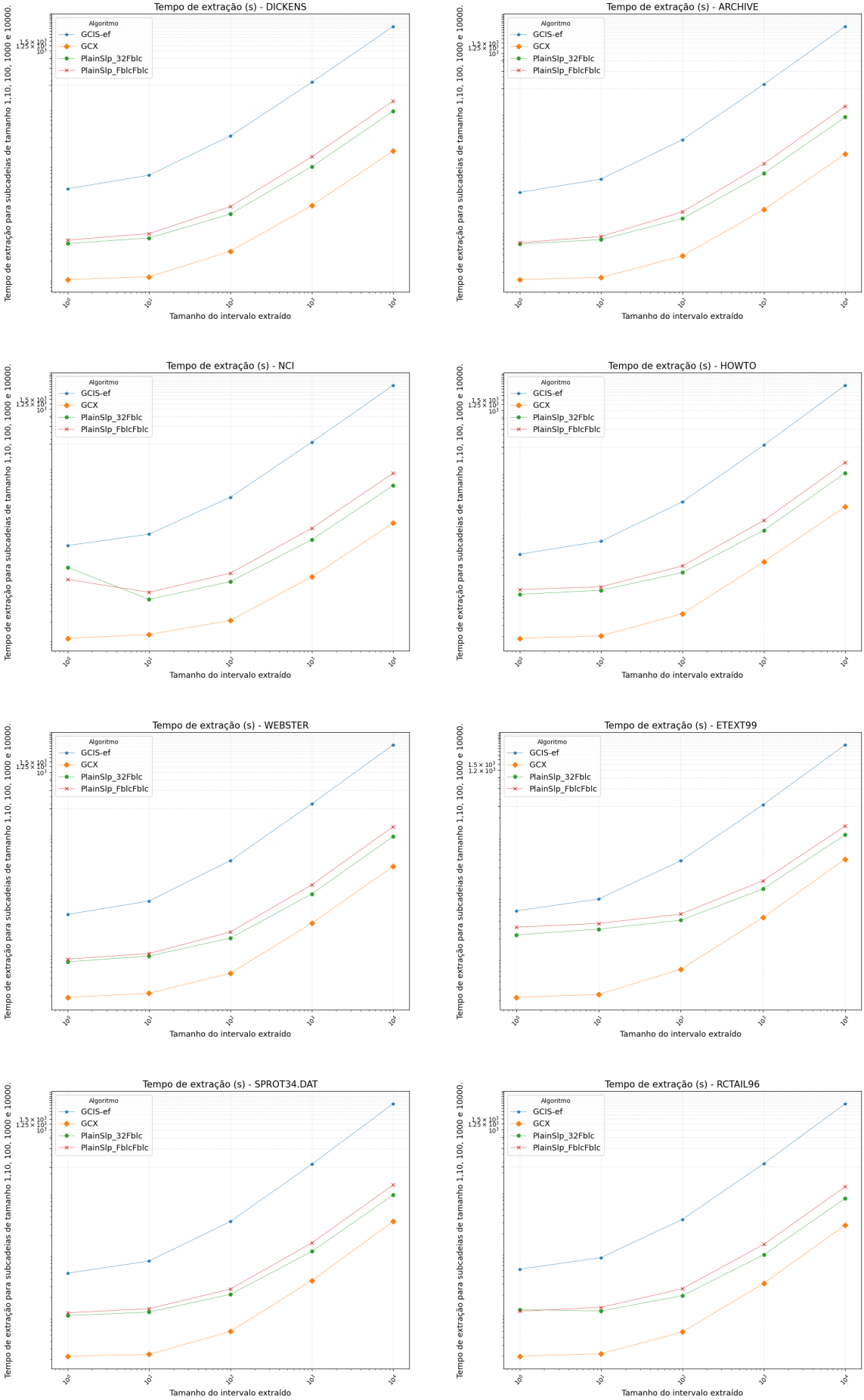


Figura 40 – Tempo de extração para textos não-repetitivos.

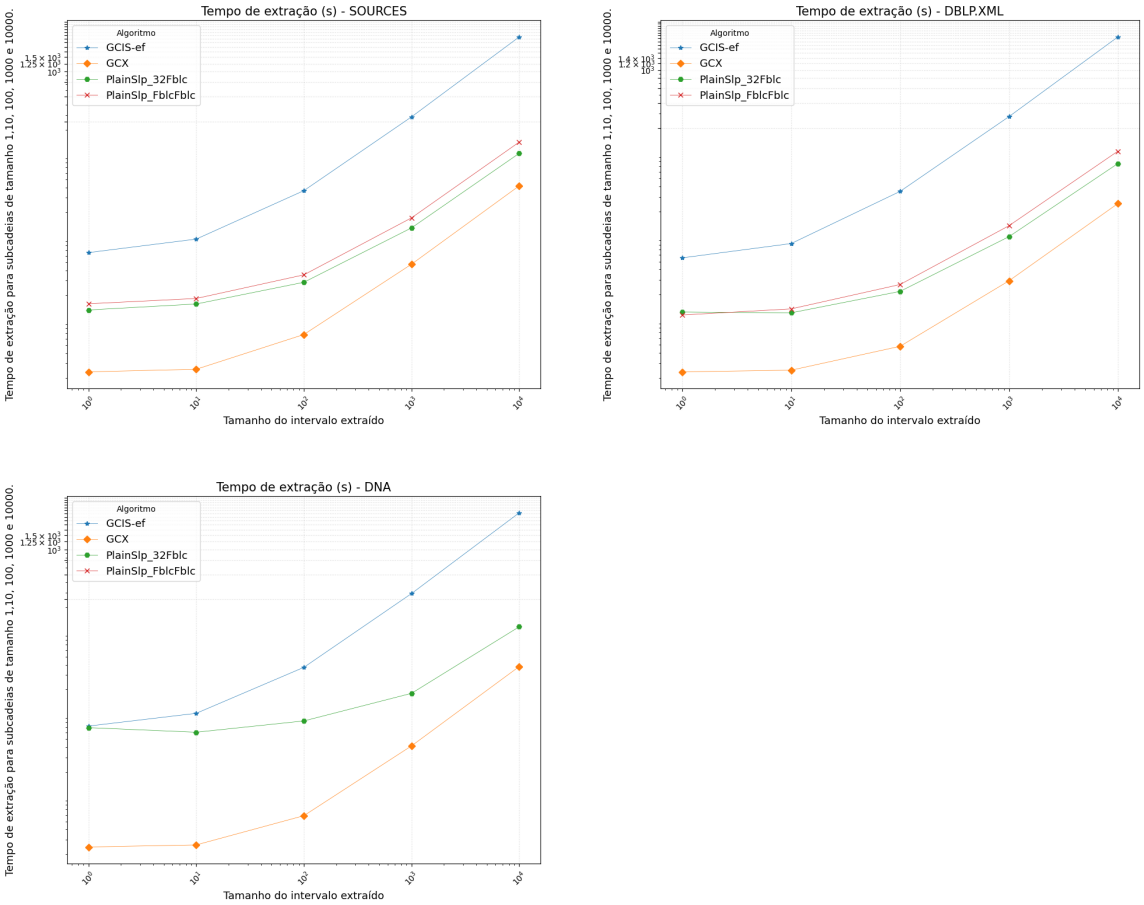


Figura 41 – Tempo de extração para textos não-repetitivos.

Capítulo 6

Conclusão

Apresentamos um novo compressor gramatical, chamado GCX. O GCX utiliza o princípio de fatoração de cadeias empregada no algoritmo de ordenação de sufixos DC3 (KäRKKäINEN; SANDERS; BURKHARDT, 2006) e o conceito de compressão gramatical baseada em ordenação de sufixos por indução introduzido pelo compressor GCIS (NUNES et al., 2022) para compor as regras de produção de uma gramática G livre de contexto capaz de gerar unicamente $S[0,n-1]$.

Nossos experimentos mostraram que o GCX apresenta uma velocidade de compressão e descompressão satisfatória em comparação com o GCIS e o RePair, no entanto, o GCX utiliza cerca de 1,6 vezes mais memória do que o GCIS.

Além disso as taxas de compressão do GCX foram inferiores às do GCIS e do RePair. A análise dos resultados evidencia que essas taxas foram mais próximas das taxas do GCIS e do RePair se considerarmos conjuntos de dados não-repetitivos. A taxa média de compressão considerando todos os conjuntos de dados foi de 11,63% para o RePair, 18,74% para o GCIS-ef, 14,57% para o GCIS-s8b e 35,77% para o GCX. Em média, os resultados do RePair foram 3 vezes melhores do que o GCX e 1,2 vezes melhor do que o GCIS-s8b.

O GCX foi construído sobre a hipótese de que regras de tamanho fixo aceleram o tempo de extração despendido pelo GCIS. Nossos experimentos confirmaram essa hipótese, demonstrando uma aceleração de 97 vezes. Além disso, observamos que essa melhoria na construção das regras reduz significativamente o tempo total de descompressão.

Tendo em vista o aumento progressivo na produção de dados, é fundamental para qualquer compressor alcançar uma taxa de compressão eficiente. Assim, pesquisas futuras serão dedicadas a entender por que a taxa de compressão piorou em nosso método, bem como ao aperfeiçoamento dessa taxa e à redução do consumo de memória necessário para realizar as operações de compressão e descompressão. Essas investigações podem incluir a análise da relação entre o tamanho das regras da gramática e a eficiência da compressão.

Referências

BELL, T. C.; WITTEN, I. H. The relationship between greedy parsing and symbolwise text compression. **J. ACM**, Association for Computing Machinery, New York, NY, USA, v. 41, n. 4, p. 708–724, jul 1994. ISSN 0004-5411. Disponível em: <https://doi.org/10.1145/179812.179892>.

CHARIKAR, M. et al. The smallest grammar problem. **IEEE Trans. Inf. Theory**, v. 51, n. 7, p. 2554–2576, 2005. Disponível em: <https://ieeexplore.ieee.org/document/1459058>.

CHILI, P. . **Gene DNA sequences**. 2005. 22 de novembro de 2023. Disponível em: <https://pizzachili.dcc.uchile.cl/texts/dna/>.

_____. **Source program code**. 2005. 22 de novembro de 2023. Disponível em: <https://pizzachili.dcc.uchile.cl/texts/code/>.

_____. **XML**. 2005. 22 de novembro de 2023. Disponível em: <https://pizzachili.dcc.uchile.cl/texts/xml/>.

_____. **Artificial Collections**. 2010. 22 de novembro de 2023. Disponível em: <https://pizzachili.dcc.uchile.cl/repcorpus/artificial>.

_____. **Pseudo-Real Collections**. 2010. 22 de novembro de 2023. Disponível em: <https://pizzachili.dcc.uchile.cl/repcorpus/pseudo-real>.

_____. **Real Collections**. 2010. 22 de novembro de 2023. Disponível em: <https://pizzachili.dcc.uchile.cl/repcorpus/real>.

DOMO. **Data Never Sleeps 11.0**. 2023. Online; acesso em 29 de fevereiro de 2024. Disponível em: <https://www.domo.com/learn/infographic/data-never-sleeps-11>.

GAGIE, T. et al. Practical random access to slp-compressed texts. In: **Proc. String Processing and Information Retrieval**. [s.n.], 2020. p. 221–231. Disponível em: <https://doi.org/10.1007/978-3-030-59212-7_16>.

KäRKKäINEN, J.; SANDERS, P.; BURKHARDT, S. Linear work suffix array construction. **J. ACM**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 6, p. 918–936, nov 2006. ISSN 0004-5411. Disponível em: <https://doi.org/10.1145/1217856.1217858>.

LARSSON, N. J.; MOFFAT, A. Off-line dictionary-based compression. **Proc. IEEE**, v. 88, n. 11, p. 1722–1732, 2000.

LOUZA, F. A.; GOG, S.; TELLES, G. P. Background. In: _____. **Construction of Fundamental Data Structures for Strings**. Cham: Springer International Publishing, 2020. p. 9–21. ISBN 978-3-030-55108-7. Disponível em: <https://doi.org/10.1007/978-3-030-55108-7_2>.

MANBER, U.; MYERS, E. W. Suffix arrays: A new method for on-line string searches. **SIAM J. Comput.**, v. 22, n. 5, p. 935–948, 1993. Disponível em: <https://doi.org/10.1137/0222058>.

MANZINI, G. **Lightweight Corpus**. 2003. 22 de novembro 2023. Disponível em: <https://people.unipmn.it/manzini/lightweight/corpus/>.

NAVARRO, G. **Compact data structures:a practical approach**. 1. ed. New York, NY, USA: Sheridan Books, Inc, 2016. Disponível em: <https://doi.org/10.1017/CBO9781316588284>.

NAVARRO, G.; MäKINEN, V. Compressed full-text indexes. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 39, n. 1, p. 2–es, apr 2007. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/1216370.1216372>.

NONG, G.; ZHANG, S.; CHAN, D. W. H. Linear suffix array construction by almost pure induced-sorting. In: **Proceedings of the Data Compression Conference**. [s.n.], 2009. p. 193–202. Disponível em: <https://doi.org/10.1109/DCC.2009.42>.

NUNES, D. S. N. et al. Grammar compression by induced suffix sorting. **CoRR**, abs/2011.12898, 2020. Disponível em: <https://doi.org/10.48550/arXiv.2011.12898>.

_____. Grammar compression by induced suffix sorting. **ACM J. Exp. Algorithmics**, v. 27, p. 1.1:1–1.1:33, 2022. Disponível em: <https://doi.org/10.1145/3549992>.

PROJECT, S. **Large Corpus**. 1998. 22 de novembro de 2023. Disponível em: <https://www.samba.org/ftp/tridge/large-corpus/README>.

PUGLISI, S. J.; SMYTH, W. F.; TURPIN, A. A taxonomy of suffix array construction algorithms. **ACM Comput. Surv.**, v. 39, n. 2, p. 4, 2007. Disponível em: <https://doi.org/10.1145/1242471.1242472>.

REINSEL, D.; GANTZ, J.; RYDNING, J. **The Digitization of the World, From Edge to Core**. [S.l.], 2020.

SAYOOD, K. **Introduction to Data Compression, Fourth Edition**. 4th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN 0124157963.

TECHNOLOGY, S. U. of. **Silesia compression corpus**. 22 de novembro de 2023. Disponível em: <https://sun.aei.polsl.pl/~sdeor/index.php?page=silesia>.

WITTEN, I. H.; MOFFAT, A.; BELL, T. C. **Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition**. [S.l.]: Morgan Kaufmann, 1999. ISBN 1-55860-570-3.