Andresso da Silva

Compressão de Sequências de Eventos com Ordem Parcial

Campina Grande, Brasil

2021

Andresso da Silva

Compressão de Sequências de Eventos com Ordem Parcial

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Universidade Federal de Campina Grande

Departamento de Engenharia Elétrica

Programa de Graduação em Engenharia Elétrica

Orientador: Francisco Marcos de Assis

Campina Grande, Brasil

2021

Andresso da Silva

Compressão de Sequências de Eventos com Ordem Parcial

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Trabalho aprovado. Campina Grande, Brasil, 19 de Abril de 2021:

Francisco Marcos de Assis
Orientador

Bruno Barbosa Albert
Convidado

Campina Grande, Brasil
2021

Aos meus pais

Agradecimentos

Aos meus pais por sempre terem dado todo o apoio, principalmente quando eu disse que iria sair de casa para me aventurar sozinho em Campina Grande. Eles me ensinaram tudo o que eu poderia precisar e continuam ensinando.

À PRAC e aos colegas e amigos da residência que me acolheram, em especial William e Julio pelos cafés e dias de filme, bolo ("solado") e lasanha. Também à Isac, Matheus e Cauan. Os dias foram bem melhores por conta de vocês.

Às minhas amigas que ajudaram a revisar o texto mesmo sendo uma área totalmente diferente das suas. Devo muito a vocês.

Aos colegas do LIMC pelos almoços juntos e aos colegas do IQuanta por me receberem e pelas conversas.

A todos os professores que contribuíram para a minha formação, em especial ao professor Francisco por me apresentar o tema e me orientar.

"These transient facts,
These fugitive impressions.
Must be transformed by mental acts,
To permanent possessions.
Then summon up your grasp of mind,
Your fancy scientific,
Till sights and sounds with thought combined
Become of truth prolific"
(James Clerk Maxwell)

Resumo

Em sistemas distribuídos (DS), alguns eventos podem ocorrer de forma independente e outros ocorrem em uma ordem determinada. Em teoria dos traços, os eventos são representados como símbolos de um alfabeto finito. A permutação de símbolos consecutivos que representam dois eventos independentes gera um conjunto de sequências equivalentes que provocam o mesmo resultado no sistema. O conjunto de sequências equivalentes do mesmo comprimento é chamado de classes de intercâmbio. A entropia de intercâmbio é definida a partir das classes de intercâmbio e fornece os limites teóricos de compressão de sequências de eventos com ordem parcial. Desta forma, busca-se apresentar dois novos algoritmos para compressão de sequências de eventos com ordem parcial usando classes de intercâmbio. Os desempenhos dos dois algoritmos são comparados em termos de tempo e taxa de redução no número de símbolos em relação à sequência original. Os algoritmos também são aplicados na estimação da entropia de intercâmbio a partir de sequências aleatórias. Os valores obtidos foram comparados com os valores teóricos, sendo possível obter valores abaixo da entropia de Shannon.

Palavras-chaves: Compressão, Ordem Parcial, Entropia de Intercâmbio, Complexidade de Intercâmbio, Traços.

Abstract

In distributed systems (DS), there are independent and dependent events. In trace theory, events are represented as symbols of a finite alphabet. Consecutive symbols that represent two independent events may be interchanged, generating a equivalent sequence. The set of equivalent sequences of the same length is defined as interchange class. Interchange entropy is based on interchange class for define information theoretic limits for sequence compression considering partial-order. It is presented two new algorithms aiming compression of sequences of events with partial-order. The performance of algorithms is analyzed in terms of time complexity and compression rate. Furthermore, the algorithms are used to estimate interchange entropy, obtaining values smaller then Shannon entropy for the same finite alphabet.

Key-words: Compression, Partial-Order, Interchange Entropy, Interchange Complexity, Traces.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DS	Sistemas distribuídos
AEP	Propriedade de Equipartição Assintótica
iid	Independente e Identicamente Distribuído
v.a	Variável Aleatória

Lista de símbolos

$\Sigma, \mathcal{X}$	Alfabeto finito
Σ^*	Conjunto de todas as palavras formadas a partir do alfabeto Σ
$\equiv_G$	Congruência de acordo com o grafo G
$\pi_A(w)$	Projeção da palavra w no conjunto A
$H(X)$	Entropia Shannon da variável X
$C_\iota(G, w)$	Complexidade de Intercâmbio associada ao grafo G da palavra w
$H_\iota(G, P)$	Entropia de Intercâmbio associada ao grafo G com distribuição P nos nós
$K_{m_1,m_2,\ldots,m_k}$	Grafo k-partido completo

Sumário

1 Introdução

Em Sistemas Distribuídos (DS), alguns eventos podem ocorrer de forma independente e outros ocorrem em uma ordem determinada. Isso leva à noção de eventos com ordem parcial (LAMPORT, 1978), em que certas sequências de eventos são equivalentes a outras sequências. Essa equivalência se refere ao fato de que algumas sequências de ocorrências produzem os mesmos resultados no sistema. Outra interpretação pode ser que alguns operações podem ser realizadas em paralelo, enquanto outras não podem.

Por exemplo, se existem as operações $(a)x := x + 2$, $(b)y := z + x$ e $(c)y := y + z$, executar a seguida de c produz o mesmo resultado de executar c seguida de a. Desta forma, é possível executar a e c de forma paralela, produzindo o mesmo resultado que seria obtido ao executar sequencialmente. Entretanto, mudar a ordem de ocorrência das operações a e b implica na mudança do resultado final. Isso pode ser verificado facilmente observando que executar a seguido de b produz $x := x + 2$ e $y := z + (x + 2)$ e executar b seguido de a produz $y := z + x$ e $x := x + 2$. Nesse caso, essas operações não podem ocorrer de forma paralela. Então, a e b são ditos operações dependentes ou não-comutativas. O mesmo acontece com b e c. Essa relação de não-comutatividade pode ser representada por um grafo como o da Fig. 1, em que cada operação é um vértice e as operações dependentes são conectadas por uma aresta.

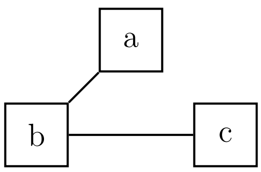


Figura 1 – Grafo G das relações de não-comutatividade entre eventos.

Outro exemplo de sequências equivalentes seria parcelas $s_1 s_2 \ldots s_n$ em que cada parcela s_i foi calculada em diferentes processadores e chegam em outro processador que executa a soma dessas parcelas. A soma final não depende da ordem de chegada das parcelas s_i. Desta forma, os cálculos das parcelas são independentes.

Pelos exemplos apresentados, é possível observar que eventos não sequenciais podem ser apresentados e analisados com a utilização de uma representação sequencial de símbolos. A representação de eventos concorrentes como uma cadeia sequencial de símbolos foi introduzida na teoria dos traços (MAZURKIEWICZ, 1977). A teoria dos traços foi desenvolvida com o intuito de fornecer ferramentas de linguagens formais à análise de sistemas concorrentes e, em particular, de redes de Petri (MAZURKIEWICZ, 1977). Com essa representação, é possível analisar o comportamento de sistemas concorrentes e sequenciais. Os chamados de traços são conjuntos de sequências equivalentes de eventos e quaisquer elementos do mesmo traço produzem o mesmo resultado no sistema. Como no primeiro exemplo, ac e ca pertencem ao mesmo traço.

Eventos com ordem parcial, portanto, permitem a geração de traços que são classes equivalentes de sequências, doravante classes de intercâmbio. As classes de intercâmbio são uma generalização das classes de tipo (SAVARI, 2004; COVER; THOMAS, 2006). Como na teoria dos tipos (COVER; THOMAS, 2006), as classes de intercâmbio podem ser utilizadas como ferramentas de compressão de sequências.

A utilização de técnicas que consideram as relações de ordem parcial podem levar a esquemas de compressão mais eficientes quando comparado à utilização de técnicas de compressões convencionais com ordem total. Técnicas de compressão de símbolos sequenciais baseadas em dicionários como o SEQUITUR (NEVILL-MANNING; WITTEN, 1997) conseguem comprimir de forma eficiente, além de descobrir padrões nessas sequências. Alur (ALUR et al., 2003) e colaboradores introduziram dois algoritmos de compressão para palavras com ordem parcial em que se utiliza o conceito de traços para comprimir palavras e, posteriormente, reconstruir uma palavra equivalente. Uma pesquisa nas bases de artigos parece indicar que não há muitos trabalhos nessa área. Em 2011, Reznik (REZNIK, 2011) propôs esquemas de codificação de palavras em que o autor não fala explicitamente em traços, mas considera conjunto de palavras com ordem parcial.

No contexto de compressão de eventos concorrentes, surge naturalmente a pergunta de como comprimir uma sequência de eventos em que há ordem parcial. A maioria das técnicas desenvolvidas para compressão de sequências de símbolos foca no processamento de eventos sequenciais (SAVARI, 2004). Além disso, há a necessidade de protocolos eficientes de transmissão de eventos parcialmente comutativos e uma medida de eficiência desses protocolos conforme o comprimento das sequências cresce.

A maior parte dos estudos relacionados à teoria da distorção (*distortion theory*) foca na medida de distorção por um símbolo introduzida por Shannon e não em medida de distorção associada às palavras definidas sob um alfabeto parcialmente comutativo. Nesse sentido, Savari (SAVARI, 2004) apresenta a entropia de intercâmbio que se baseia em uma generalização da complexidade de Kolmogorov, a complexidade de intercâmbio. A entropia de intercâmbio faz uso de ferramentas da teoria dos traços (MAZURKIEWICZ, 1977) para definir o limite teórico de compressão de sequências de eventos com ordem parcial em termos de números de *bits* por símbolo em média para representar as sequências (SAVARI, 2003). Esses limites podem ser utilizados para determinar a eficiência de esquemas de codificação e compressão de sequências de eventos com ordem parcial. Além disso, o limite superior da entropia de intercâmbio é a entropia de Shannon e, portanto, quando se considera a ordem parcial, a compressão obtida é maior ou igual a conseguida com ordem total. Segundo Alur (ALUR et al., 2003), até a publicação do artigo (SAVARI, 2003), também de Savari, não havia sinal de trabalhos focados no aspecto de teoria de distorção para sequências de eventos com ordem parcial.

Desta forma, busca-se apresentar os principais teoremas da propriedade de equipartição assintótica (AEP), teoria dos tipos e complexidade de Kolmogorov necessários para entender as generalizações feitas por Savari. São apresentados os conceitos propostos por Savari (SAVARI, 2004) e exemplos. Além disso, são propostos dois novos algoritmos para compressão de sequências de eventos com ordem parcial e um sub-ótimo com tempo polinomial e um ótimo com o tempo exponencial para efeitos de comparação. Esses algoritmos identificam subsequências pertencentes a uma mesma classe de intercâmbio e que são substituídas por um único símbolo que são apresentados em forma de um dicionário. Os desempenhos dos dois algoritmos são comparados em termos de tempo e taxa de redução no número de símbolos dos dicionários gerados.

Os algoritmos também são utilizados para estimar a entropia de intercâmbio associados a alguns grafos a partir de sequências aleatórias, sendo possível encontrar valores abaixo da entropia de Shannon. Essa abordagem, até onde vai o conhecimento do autor, é nova.

O documento está organizado da seguinte maneira: no Capítulo 2, são apresentados os principais conceitos e desigualdades referentes à propriedade de equipartição assintótica, teoria dos tipos, teoria dos traços e complexidade de Kolmogorov que servirão como base para compreender as generalizações feitas por Savari (SAVARI, 2004) e sobre compressão sem perdas. Nesse mesmo capítulo, são apresentados os conceitos de Complexidade de Intercâmbio e Entropia de Intercâmbio propostos por Savari (SAVARI, 2004). No Capítulo 3, são apresentados e descritos dois algoritmos desenvolvidos para comprimir sequências de eventos com ordem parcial e estimar a entropia de intercâmbio por meio da contagem de classes de intercâmbio. No Capítulo 4, são apresentadas as taxas de compressão obtidas em sequências aleatórias de eventos com diferentes comprimentos e diferentes parâmetros nos algoritmos. Além disso, são apresentadas as estimações da entropia de intercâmbio obtidas a partir dos algoritmos. No Capítulo 5, são feitas discussões acerca dos resultados e das limitações dos algoritmos. No Capítulo 6, são apresentadas as conclusões e trabalhos futuros.

2 Fundamentação Teórica

Nesse Capítulo serão apresentados os conceitos fundamentais necessários para compreender e interpretar as generalizações feitas por Savari (SAVARI, 2004). A propriedade de equipartição assintótica (AEP) e a teoria dos tipos possuem desigualdades fundamentais que permitem analisar a compressão com e sem perdas de sequências de símbolos, além do número mínimo de *bits* por símbolo em média que é necessário para representar essas sequências. Além disso, são apresentados os conceitos fundamentais da teoria dos traços que é utilizada por Savari (SAVARI, 2004) para propor uma generalização da complexidade de Kolmogorov. Desta forma, os conceitos fundamentais da complexidade de Kolmogorov também são apresentados. Por fim, são apresentadas as definições e teoremas essenciais da complexidade de intercâmbio e entropia de intercâmbio.

Os conceitos, teoremas e provas relacionados à AEP e à Teoria dos Tipos apresentados nessa Seção são baseados em (COVER; THOMAS, 2006, cap.3) e (COVER; THOMAS, 2006, cap.11). As definições e resultados sobre teoria dos traços são baseados em (MAZURKIEWICZ, 1977), (DIEKERT; ROZENBERG, 1995, cap.1) e (PERRIN, 1985). Por fim, os conceitos e resultados acerca de complexidade de Kolmogorov são baseados em (COVER; THOMAS, 2006, cap.14), (LI; VITNYI, 2008) e (HORIBE, 2003).

2.1 Propriedade de Equipartição Assintótica

A propriedade de equipartição assintótica (AEP) possibilita a classificação das sequências de variáveis aleatórias (v.a) em dois conjuntos, as *sequências típicas* e as *sequências não-típicas*. As sequências típicas são aquelas que possuem as entropias empíricas próximas à entropia da fonte e as sequências não-típicas são aquelas que não possuem essa propriedade. A definição de entropia é apresentada a seguir.

Definição 2.1.1. *(Entropia) A entropia $H(X)$ de uma v.a. discreta X com distribuição $p(x)$ é definida como*

$$H(X) = \sum_{x \in \mathcal{X}} -p(x) \log_2(p(x)) \tag{2.1}$$

A propriedade da equipartição assintótica (AEP) é apresentada no Teorema 2.1.1.

Teorema 2.1.1. *(AEP (COVER; THOMAS, 2006, p.58))*

Sejam $X_1, X_2, \ldots, X_n$ v.a. independentes e identicamente distribuídas (iid) com *distribuição p, então*

$$-\frac{1}{n}\log_2 p\left(X_1, X_2, \ldots, X_n\right) \to H(X) \tag{2.2}$$

em probabilidade. Ou seja, para todo $\epsilon > 0$ vale

$$\Pr\left\{\left|-\frac{1}{n}\log_2 p\left(X_1, X_2, \ldots, X_n\right) - H(X)\right| > \epsilon\right\} \to 0.$$

Uma das interpretações para as sequências típicas é que elas são quase igualmente surpreendentes, no sentido de apresentarem uma incerteza próxima à da entropia da fonte. A partir da AEP, é possível definir o chamado de conjunto típico que é formado por todas as sequências de comprimento n que apresentam uma probabilidade empírica compreendida em um intervalo definido.

Definição 2.1.2. *(Conjunto Típico) O conjunto típico $A_\epsilon^{(n)}$, $\epsilon > 0$ com repeito a $p(x)$ é o conjunto das sequências $(x_1, x_2, \ldots, x_n) \in \Sigma^n$ com a propriedade*

$$2^{-n(H(X)+\epsilon)} \leq p(x_1, x_2, \ldots, x_n) \leq 2^{-n(H(X)-\epsilon)} \tag{2.3}$$

em que Σ é o alfabeto de símbolos.

Algumas das propriedades do conjunto típico são apresentadas no Teorema 2.1.2.

Teorema 2.1.2. *(Propriedades do Conjunto Típico)*

1. *Se $(x_1, x_2, \ldots, x_n) \in A_\epsilon^{(n)}$, então $H(X) - \epsilon \leq -\frac{1}{n}\log_2 p(x_1, x_2, \ldots, x_n) \leq H(X) + \epsilon$.*
2. *$\Pr\{A_\epsilon^{(n)}\} > 1 - \epsilon$;*
3. *A cardinalidade do conjunto típico $A_\epsilon^{(n)}$ tem como limite superior $\left|A_\epsilon^{(n)}\right| \leq 2^{n(H(X)+\epsilon)}$;*
4. *A cardinalidade do conjunto típico $A_\epsilon^{(n)}$ tem como limite inferior $\left|A_\epsilon^{(n)}\right| \geq (1-\epsilon)\, 2^{n(H(X)-\epsilon)}$*

para n grande o suficiente.

A primeira propriedade segue diretamente da definição de conjunto típico. A segunda propriedade indica que pode-se definir o conjunto típico de forma que as sequências presentes nesse conjunto apareçam com uma probabilidade arbitrariamente próxima de 1. A terceira propriedade indica que a cardinalidade do conjunto típico é menor ou igual à cardinalidade do conjunto total de sequências de comprimento definidas sobre um alfabeto finito Σ que é igual a $|\Sigma|^n$.

Em outras palavras, existe um conjunto de sequências que ocorrem com alta probabilidade. Além disso, a cardinalidade desse conjunto é menor do que a cardinalidade do conjunto de todas as sequências, fazendo com sejam necessários $2^{n(H+\epsilon)}$ *bits* em média para representá-lo, como exemplificado na Fig. 2. Essas propriedades do conjunto típico podem ser utilizadas para a compressão com perdas de sequências.

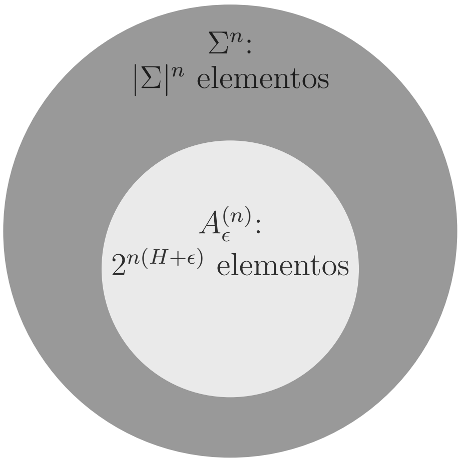


Figura 2 – Conjunto das sequências e conjunto típico.

O Teorema 2.1.3 apresenta que a entropia $H(X)$ é o limite inferior do número de *bits* necessários para representar símbolos de uma sequência com certa distribuição de probabilidade.

Teorema 2.1.3. *((COVER; THOMAS, 2006, p.62)) Seja $X^n = X_1, X_2, \dots, X_n$ uma sequência iid com distribuição $p(x)$ e seja $\epsilon > 0$, então existe um código que mapeia a sequência $x^n = x_1, x_2, \dots, x_n$ de comprimento n em palavras binárias tais que o mapeamento é inversível e*

$$\mathbb{E}\left[\frac{1}{n}l(X^n)\right] \leq H(X) + \epsilon \tag{2.4}$$

em que $l(X^n)$ denota o comprimento da palavra-código correspondente a X^n. Em outras palavras, é possível representar as sequências X^n usando $nH(X)$ bits em média.

2.2 Teoria dos Tipos

Enquanto a AEP possibilita a compressão por demonstrar que existe um subconjunto das sequências possíveis que apresenta alta probabilidade, a Teoria dos Tipos utiliza semelhanças entre as sequências para agrupar nos chamados de tipos. Esses tipos estão relacionados à frequência relativa de ocorrência de símbolos de um alfabeto nessas sequências.

Definição 2.2.1. *(Tipos (COVER; THOMAS, 2006, p.347)) O tipo $P_{\mathbf{x}}$ (ou distribuição empírica de probabilidade) de uma sequência $x_1, x_2, \ldots, x_n$ é a frequência de ocorrência relativa de cada símbolo do alfabeto Σ. Desta forma, para todo $a \in \Sigma$, tem-se $P_{\mathbf{x}}(a) = N(a|\mathbf{x})/n$, em que $N(a|\mathbf{x})$ denota o número de ocorrências do símbolo a na sequência $\mathbf{x} \in \Sigma^n$.*

Definição 2.2.2. *(Conjunto dos Tipos com Denominador n (COVER; THOMAS, 2006, p.348)) O conjunto dos tipos com denominador n é formado pelas $|\Sigma|$-tuplas de sequência relativas possíveis em uma sequência $x_1, x_2, \ldots, x_n$ e é denotado por $\mathcal{P}_n$.*

Exemplo 2.2.1. *(Exemplo de Conjunto dos Tipos com Denominador n) Seja $\Sigma = \{0, 1\}$ o conjunto dos tipos com denominador n é*

$$\mathcal{P}_n = \left\{ (P(0), P(1)) : \left(\frac{0}{n}, \frac{n}{n}\right), \left(\frac{1}{n}, \frac{n-1}{n}\right), \ldots, \left(\frac{n}{n}, \frac{0}{n}\right) \right\}$$

Definição 2.2.3. *(Classe de Tipo) Se $P \in \mathcal{P}_n$, então o conjunto de sequências de comprimento n e classe P é chamado de classe de tipo P, denotada por $T(P)$ e*

$$T(P) = \{\mathbf{x} \in \Sigma^n : P_{\mathbf{x}} = P\} \tag{2.5}$$

Em outras palavras, a classe de tipo associado a um tipo $P_{\mathbf{x}}$ é formada por todas as sequências que apresentam a mesma distribuição empírica de probabilidade, i.e., o mesmo tipo de $P_{\mathbf{x}}$.

Exemplo 2.2.2. *Seja $\Sigma = \{1, 2, 3\}$ e $\mathbf{x} = 123322$. Então, por definição, o tipo $P_{\mathbf{x}}$ é*

$$P_{\mathbf{x}}(1) = \frac{1}{6} \qquad P_{\mathbf{x}}(2) = \frac{3}{6} \qquad P_{\mathbf{x}}(3) = \frac{2}{6}$$

O tipo de classe de $P_{\mathbf{x}}$ é formado pelas sequências de comprimento $n = 6$ com uma ocorrência de 1, três ocorrências de 2 e duas ocorrências de 3, ou seja,

$$T(P_{\mathbf{x}}) = \{123322, 123232, \ldots, 332221\}$$

em que o número de elementos de T($P_{\mathbf{x}}$) é dado por

$$|T(P_{\mathbf{x}})| = \binom{6}{1, 2, 3} = 60$$

O número de elementos da classe de tipo pode ser calculada de forma geral com uma multinomial. O próximo teorema fornece um limite superior polinomial para o número de elementos do conjunto de tipos com denominador n.

Teorema 2.2.1. *(Teorema 11.1.1 (COVER; THOMAS, 2006, p.349))*

$$|\mathcal{P}_n| \leq (n+1)^{|\Sigma|} \tag{2.6}$$

Demonstração. A prova pode ser encontrada em (COVER; THOMAS, 2006, p.349). □

Enquanto o número de sequências cresce exponencialmente com n, o números de tipos distintos com denominador n é majorado por uma função polinomial de n. Desta forma, se referindo aos tipos ao invés das sequências individuais são necessários menos *bits* de informação para representar e construir sequências equivalentes.

A partir do Teorema 2.2.2 é possível associar o número de sequências com certo tipo à entropia.

Teorema 2.2.2. *(Tamanho de classe de tipo $T(P)$) Para qualquer tipo $P \in \mathcal{P}_n$, tem-se*

$$\frac{1}{(n+1)^{|\Sigma|}} 2^{nH(P)} \leq |T(P)| \leq 2^{nH(P)} \tag{2.7}$$

em que $H(P)$ de refere à entropia considerando a distribuição de probabilidade do tipo P.

Desta forma, o número de sequências de comprimento n que tem um tipo P vai sempre obedecer à desigualdade do Teorema 2.2.2.

2.3 Independência e Traços

Um conjunto Σ é chamado de alfabeto e é composto por um número finito de símbolos. Um símbolo desse alfabeto pode ser interpretado como uma ação ou evento em um sistema concorrente (MAZURKIEWICZ, 1977; DIEKERT, 1990). Uma sequência finita de símbolos é chamadas de palavras sobre Σ. Uma palavra vazia é denotada por ϵ e o conjunto de todas palavras sobre Σ é denotado por Σ^*.

Seja um grafo não-orientado chamado de *grafo de não-comutatividade* $G = (V, E)$ em que V está associado a um alfabeto finito Σ e em que os símbolos dos vértices $a, b \in V$ ligados por uma aresta, i.e., $(a, b) \in E$, não comutam.

Desta forma, se dois vértices são ligados por uma aresta, então eles são chamados de adjacentes e representam eventos que não são independentes. Se os símbolos a e b não comutam, então a relação é representada por $ab \not\equiv_G ba$. Se eles comutam, então a relação é representada por $ab \equiv_G ba$.

Por exemplo, se o *grafo de não-comutatividade* for $a - b - c$, então a não comuta com b e b não comuta com c, mas a comuta com c. Portanto, nesse caso, $ab \not\equiv_G ba$, $bc \not\equiv_G cb$

e $ac \equiv_G ca$. O complementar do grafo de não-comutatividade, $\overline{G}$, é chamado de grafo de comutatividade em que os símbolos ligados por uma aresta comutam.

Classes de palavras equivalentes de acordo com $\equiv_G$ são chamados de traços sobre G ou, ainda, classes de intercâmbio. O traço gerado pela palavra w é o conjunto formado pelas palavras que podem ser obtidas a partir de w por meio da troca de posição das letras consecutivas que comutam de acordo com o grafo G (MAZURKIEWICZ, 1977).

Cada palavra w sobre Σ representa uma classe de intercâmbio denotado por $[w]_G$. Se duas palavras pertencem à mesma classe de intercâmbio, elas são ditas congruentes. Em outras palavras, se $u \in \Sigma^*$ e $v \in \Sigma^*$ e $v \in [u]_G$, então elas são congruentes de acordo com G e essa relação é denotada por $u \equiv_G v$.

Exemplo 2.3.1. *(Adaptado de (DIEKERT, 1990, p.9)) Um banco de dados distribuído que recebe um conjunto de transações que podem ocorrer de forma concorrente. Nessas transações, são informados valores numéricos inteiros que podem ser w, x, y ou z e os valores anteriores salvos no banco de dados são atualizados. As regras de atualização dos valores no banco de dados são*

(a) $x := x + y$

(b) $x := w + x$

(c) $y := y + z$

(d) $w := 2y + z$

(e) $z := y + 2z$

(f) $x := w + x + y$

(g) $x := w + x + 2y$

Desta forma, podem haver atualizações que geram valores incorretos se forem executadas no mesmo momento. As dependências entre as transações podem ser representadas por um grafo de dependência em que cada regra é um nó. Caso uma regra utilize valores retornados por outras regras, então essa regra é dependente das outras e elas são ligadas por uma aresta. Isso significa que regras dependentes não podem ser executadas de forma concorrente. Por exemplo, como as regras (a) e (b) calculam e utilizam o valor de x, então elas são dependentes e não podem ser executadas simultaneamente, pois haveria uma inconsistência no valor de x. Seguindo esse raciocínio obtém-se o grafo de dependência apresentado na Fig. 3.

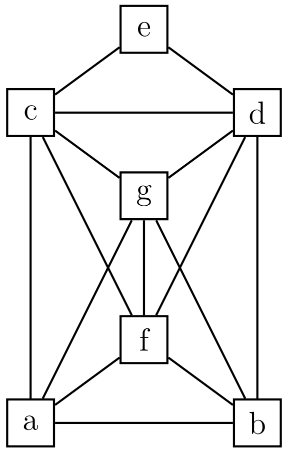


Figura 3 – Grafo de não-comutatividade de transações.

Considerando uma sequência de transações $p = uklv$ em que as transações k, l independentes, a sequência $p' = ulkv$ produzirá o mesmo resultado. De forma geral, qualquer palavra do traço gerado por p produzirá o mesmo resultado de p, de forma que a ordem das transações independentes não gera um resultado diferente no sistema.

Exemplo 2.3.2. *Considerando o grafo $G = (V, E)$ da Fig. 4, de forma que as palavras abcde, cabed e adbec estão definidas em V. É possível verificar que abcde é congruente a cabed, mas não é congruente a adbec. Uma forma de verificar é gerando o traço de abcde que é $[abcde]_G = \{abcde, abced, acbed, acbde, cabde, cabed, acebd, caebd\}$ e cabed $\in [abcde]_G$, mas adbec $\notin [abcde]_G$.*

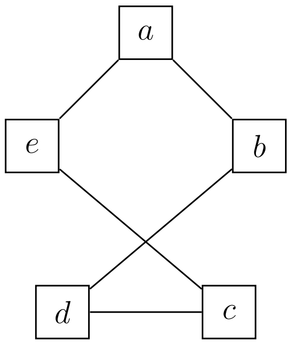


Figura 4 – Grafo de não-comutatividade G

A dificuldade de gerar o traço de palavras aumenta com o número de letras diferentes na palavra, o comprimento da palavra e as relações definidas pelo grafo. Para contornar esse problema, é possível utilizar um teorema provado por Perrin (PERRIN, 1985) em que são fornecidas as condições necessárias e suficientes para definir se duas palavras são congruentes ou não. Isso é feito por meio da definição da chamada de projeção de arestas.

Definição 2.3.1. *(Projeção) Para qualquer subconjunto A do conjunto de vértices V do grafo G e qualquer palavra w sobre V, a projeção $\pi_A(w)$ da palavra w em A é obtida deletando de w todos os símbolos que não estão presentes em A.*

O teorema de Perrin é anunciado a seguir.

Teorema 2.3.1. *(Perrin (PERRIN, 1985, p.330)) As condições necessárias e suficientes para que duas palavras w, u sejam congruentes é que elas estejam na mesma classe de tipo e que $\pi_{\{x,y\}}(w) = \pi_{\{x,y\}}(u)$ para todos os símbolos $x, y \in V$ que são ligados por uma aresta (adjacentes) em G.*

Demonstração. A prova pode ser encontrada em (PERRIN, 1985, p.330). □

Duas palavras estão na mesma classe de tipo quando elas possuem as mesmas frequências relativas de símbolos e, consequentemente, o mesmo comprimento. Por exemplo, *aabcde* e *bcadea* são do mesmo tipo de classe já que ambas possuem duas ocorrências do símbolo a, e uma ocorrência dos símbolos b, c, d e e. Já as palavras *aabcde* e *aaaabc* não são do mesmo tipo de classe, pois não possuem a mesma frequência relativa dos símbolos. Esse resultado torna a implementação de algoritmos de verificação mais simples do que a geração de todo o traço e depois a busca. Além disso, com as projeções de arestas de uma palavra de certo tipo de classe é possível gerar mais facilmente o seu traço.

Exemplo 2.3.3. *(Projeção de arestas) Considerando o grafo $G = (V, E)$ da Fig. 4 do exemplo anterior e as palavras abcde, cabed e adbec definidas em V. O primeiro passo é identificar os pares de símbolos adjacentes em G que são $a - b$, $a - e$, $b - d$, $c - d$, $c - e$. Desta forma, as projeções de abcde em relação aos pares de vértices adjacentes em G seriam $\pi_{\{a,b\}}(abcde) = ab$, $\pi_{\{a,e\}}(abcde) = ae$ e assim por diante. Fazendo o processo para todos os pares de símbolos adjacentes para as três palavras, montou-se a Tabela 1.*

$\{x, y\}$	$\pi_{\{x,y\}}(abcde)$	$\pi_{\{x,y\}}(cabed)$	$\pi_{\{x,y\}}(adbec)$
$\{a, b\}$	ab	ab	ab
$\{a, e\}$	ae	ae	ae
$\{b, d\}$	bd	bd	db
$\{c, d\}$	cd	cd	dc
$\{c, e\}$	ce	ce	ec

Tabela 1 – Projeções das palavras de acordo com os símbolos adjacentes em G

Como em todos os casos as projeções de abcde e cabed são iguais, então elas são congruentes, mas abcde não é congruente a adbec.

2.4 Complexidade de Kolmogorov

Nessa seção serão apresentados as principais propriedades da complexidade de Kolmogorov necessárias para entender os resultados obtidos por Savari em (SAVARI, 2004). Serão considerados um alfabeto finito Σ e o conjunto de todas as palavras definidas sobre Σ denotado por Σ^*, incluindo o símbolo vazio ϵ.

Definição 2.4.1. *(Complexidade de Kolmogorov (COVER; THOMAS, 2006, p.466)) A complexidade de Kolmogorov $C_{\mathcal{U}}(u)$ da palavra $u \in \Sigma^*$ com respeito a um computador universal $\mathcal{U}$ é definida como o menor comprimento $l(p)$ do programa p entre todos os programas que exibem u e param, i.e.,*

$$C_{\mathcal{U}}(u) = \min_{p:\mathcal{U}(p)=u} l(p). \tag{2.8}$$

Em outras palavras, a complexidade $C_{\mathcal{U}}(u)$ é a menor descrição de u entre as descrições possíveis de u realizadas por um computador $\mathcal{U}$.

Todas as complexidades serão consideradas como calculadas em relação a um computador universal $\mathcal{U}$, então será utilizando somente $C(x)$ para denotar a complexidade de Kolmogorov da palavra $x \in \Sigma^*$.

A complexidade de Kolmogorov é subaditiva (LI; VITNYI, 2008), i.e., a complexidade de duas palavras u e v, $u, v \in \Sigma^*$, concatenadas sem marcação obedece a relação apresentada em Eq. 2.9.

$$C(u, v) \leq C(u) + C(v) + 2\log\left(\min\left(C(u), C(v)\right)\right) \tag{2.9}$$

Definição 2.4.2. *(Complexidade Condicional de Kolmogorov (COVER; THOMAS, 2006, p.467)) Quando já se sabe o comprimento $l(u)$ de $u \in \Sigma^*$, então a complexidade condicional de Kolmogorov é definida como*

$$C(u|l(u)) = \min_{p:\mathcal{U}(p,l(p))=u} l(p)$$

Uma das propriedades da complexidade de Kolmogorov é que a complexidade condicional de Kolmogorov de uma palavra $u \in \Sigma^*$ é menor ou igual ao comprimento da palavra. Isso é anunciado no Teorema 2.4.1.

Teorema 2.4.1. *(Teorema 2.1.2 de (LI; VITNYI, 2008, p.100)) Para uma constante c, vale*

$$C(u) \leq l(u) + c \tag{2.10}$$

Demonstração. Dado um programa que copia os símbolos da palavra de entrada na saída, resultando numa complexidade igual a $l(u) + c$ para qualquer palavra. □

Corolário 2.4.1.

$$C(u) \leq l(u)\log|\Sigma| + c \tag{2.11}$$

Corolário 2.4.2. *Seja qualquer palavra $u \in \Sigma^*$ e alguma constante c, vale*

$$C(u) \leq l(u)\log|\Sigma| + 2\log|\Sigma| + c \tag{2.12}$$

Para entender alguns dos resultados relacionados à entropia de intercâmbio apresentados por (SAVARI, 2004), será necessário utilizar o teorema a seguir que relaciona a complexidade de Kolmogorov e a entropia.

Teorema 2.4.2. *(Relação entre Complexidade de Kolmogorov e Entropia (COVER; THOMAS, 2006, p.473)) Considerando o processo estocástico $\{X_i\}$ sendo iid de acordo com a função de massa de probabilidade $f(x)$, $x \in \Sigma$, em que Σ é o alfabeto finito. Denotando $f(\mathbf{x}) = \prod_{i=1}^{n} f(x_i)$, então, para todo n, existe uma constante c tal que vale*

$$H(X) \leq \frac{1}{n}\sum_{\mathbf{x}} f(\mathbf{x})C(\mathbf{x}|n) \leq H(X) + \frac{(|\Sigma|-1)\log n}{n} + \frac{c}{n}.$$

Por conseguinte,

$$\mathbb{E}\frac{1}{n}C(\mathbf{X}|n) \to H(X).$$

Demonstração. A demonstração pode ser encontrada em (COVER; THOMAS, 2006, p.473) e a versão da prova para sequências ergódicas pode ser encontrada em (HORIBE, 2003). □

Removendo a dependência do tamanho n da sequência, pode-se interpretar que no limite o computador consegue comprimir a sequência ao ponto de ser semelhante à entropia $H(X)$, ou seja,

$$\mathbb{E}\frac{1}{n}C(\mathbf{X}) \to H(X). \tag{2.13}$$

2.5 Complexidade de Intercâmbio e Entropia de Intercâmbio

Considerando uma palavra u sobre o conjunto de vértices V do grafo G, então V^* é o conjunto de todas as palavras definidas em V. Além disso, é $|V|$ o número de vértices de G e $C(u)$ é a complexidade de Kolmogorov de u. Assim, a complexidade de intercâmbio pode ser interpretada como o comprimento do menor programa que exibe a palavra $v \in V^*$ que é congruente a u em relação ao grafo G. (SAVARI, 2004), como é apresentado na Definição 2.5.1.

Definição 2.5.1 (Complexidade de Intercâmbio)**.**

$$C_\iota(G,u) = \min\{C(v) : v \in V^*,\, v \equiv_G u\}$$

Conforme o Teorema 2.3.1, u e v serão congruentes se eles pertencem à mesma classe de tipo e possuem as projeções $\pi_{\{x,y\}}(v) = \pi_{\{x,y\}}(u)$ para todos os símbolos $x, y \in V$ que são adjacentes em G. Uma propriedade da complexidade de intercâmbio é que ela é quase subaditiva. Desta forma, para $u, v \in V^*$ tem-se

$$C_\iota(G,uv) \leq C_\iota(G,u) + C_\iota(G,v) + 2\log(\min(C_\iota(G,u), C_\iota(G,v))) + o(1) \tag{2.14}$$

Como $C_\iota(G,u) \leq C(u)$, usando o Corolário 2.4.1, chega-se a

$$C_\iota(G,u) \leq l(u)\log|V| + 2\log|C| + c$$

Então, a Eq. 2.9 implica em

$$C_\iota(G,uv) \leq C_\iota(G,u) + C_\iota(G,v) + 2\log(\min(l(u),l(v))) + o(1) \qquad (2.15)$$

Com o objetivo de analisar o comportamento de $n^{-1}C_\iota(G,u_1u_2\ldots u_n)$, $u_i \in V$, para valores de n grandes, pode-se utilizar o Teorema 2.5.1.

Teorema 2.5.1. *(Teorema 2.4 (SAVARI, 2004, p.1428)) Sejam* $\{X_{m,n}\}$ *e* $\{A_{m,n}\}, m \leq n$, *duas sequências de variáveis aleatórias que seguem as propriedades:*

1. $X_{0,n} \leq X_{0,m} + X_{m,n} + A_{m,n}$;
2. $X_{m,n}$ *é estacionário e ergódico;*
3. $\mathbb{E}[X_{0,1}] < \infty$ *e para cada* n, $\mathbb{E}[X_{0,1}] \geq c_0 n$, $c_0 > -\infty$;
4. $A_{m,n} \geq 0$ *e* $\lim_{n\to\infty} \mathbb{E}[A_{0,n}/n] = 0$

então

$$\lim_{n\to\infty} \frac{X_{0,n}}{n} = \lim_{n\to\infty} \frac{\mathbb{E}[\{X_{0,n}\}]}{n} = \inf_{m\geq 1} \frac{\mathbb{E}[\{X_{0,m}\}]}{m}$$

com alta probabilidade.

A partir do Teorema 2.5.1 e fazendo $U_i \in V$, e $X_{m,n} = C_\iota(G,U_{m+1}U_{m+2}\ldots U_n)$ e $A_{m,n} = 2\log(l(U_{m+1}U_{m+2}\ldots U_n)) + O(1)$ é possível obter o resultado apresentado no Teorema 2.5.2. A prova desse teorema pode ser encontrada em (SAVARI, 2004, p.1428).

Teorema 2.5.2. *(Propriedade de Equipartição Assintótica para Classes de Intercâmbio, Teorema 2.5 de (SAVARI, 2004, p.1428)) Seja* $U_1U_2\ldots$ *uma palavra aleatória de uma fonte com alfabeto finito, estacionária e ergódica ou a saída de uma fonte unifilar de Markov com alfabeto finito. Então,*

$$\begin{aligned}\lim_{n\to\infty} \frac{C_\iota(G,U_1U_2\ldots U_n)}{n} &= \lim_{n\to\infty} \frac{\mathbb{E}[\{C_\iota(G,U_1U_2\ldots U_n)\}]}{n} \\ &= \inf_{m\geq 1} \frac{\mathbb{E}[\{C_\iota(G,U_1U_2\ldots U_m)\}]}{m}\end{aligned}$$

com alta probabilidade.

O Teorema 2.5.2 é uma generalização do Teorema 2.1.1 para alfabetos em que há uma ordenação parcial, sendo possível verificar que existem sequências chamadas de típicas. Essas sequências típicas apresentam uma informação própria média por símbolo próxima da entropia da fonte. Enquanto as sequências típicas acorrem com alta probabilidade, as sequências não típicas acontecem com baixa probabilidade. Isso leva à definição da entropia de intercâmbio.

Definição 2.5.2 (Entropia de Intercâmbio)**.**

$$H_\iota(G,P) = \lim_{n\to\infty} n^{-1} C_\iota\left(G, U_1 \dots U_n\right). \tag{2.16}$$

em que P se refere a uma medida de probabilidade associada aos vértices de G. Há a convergência com alta probabilidade do limite para o valor da entropia de intercâmbio.

O Teorema 2.5.2 estabelece que para qualquer $\epsilon > 0$ sequências longas requerem no máximo $H_\iota(G,P) + \epsilon$ *bits* por símbolo para descrever uma palavra equivalente em relação ao grafo G. O Teorema 2.5.3 estabelece que $H_\iota(G,P)$ é o limite de compressão.

Teorema 2.5.3. *(Teorema da Codificação, Teorema 2.6 (SAVARI, 2004, p.1429)) Seja $U_1U_2\dots$ uma sequência aleatória de saída de uma fonte P com estados finitos ou com um alfabeto unifilar finito de Markov. Para $n \geq 1$, seja B_n o conjunto com cardinalidade $|B_n|$ de classes de intercâmbio de comprimento n em relação a um grafo de não-comutatividade G. Assume-se que*

$$\limsup_{n\to\infty} \frac{\log|B_n|}{n} \leq H_\iota(G,P)$$

Para qualquer palavra w, $C_G(w)$ denota a classe de intercâmbio que contém a palavra w em relação ao grafo de não comutação $G = (V,E)$, $w \in V$, que é definida como

$$C_G(w) = \{y : y \in V^*,\, y \equiv_G w\}$$

Desta forma,

$$\lim_{n\to\infty} P(C_G(U_1U_2\dots U_n) \in B_n) = 0.$$

A definição utilizada para a classe de tipo é semelhante à classe de intercâmbio que é traço gerado por w. O Teorema 2.5.3 é o paralelo do Teorema 2.1.3 que afirma que é possível representar as sequências X^n para X pertencente a um alfabeto finito $\mathcal{X}$ usando $nH(X)$ *bits* em média. Segue que é possível concluir que sequências típicas de comprimento n estão contidas em aproximadamente $2^{nH_\iota(G,P)}$ classes de intercâmbio típicas.

Classes de intercâmbio se assemelham ao conceito de classes de tipo (ver Definição 2.2.3) e configuram uma generalização desse último. Cabe notar que no caso de G

ser um grafo vazio (sem arestas) o número de tipos de classes de comprimento n para um conjunto V de vértices com cardinalidade $|V|$ é no máximo $(n+1)^{|V|}$ que é o número de classes de tipo convencionais já que todos os símbolos são congruentes e podem ser comutados.

É possível obter uma expressão exata para $H_\iota(G,P)$ no caso da fonte ser discreta e sem memória e o grafo G ser um grafo completo k-partido $K_{m_1,m_2,\ldots,m_k}$. Esse resultado é apresentado no Teorema 2.5.4.

Teorema 2.5.4. *(Teorema 2.11 (SAVARI, 2004, p.1432)) Assumindo uma fonte discreta e sem memória com distribuição de probabilidade no conjunto de vértices V. Suponto que $V = V_1 \cup V_2 \cup \ldots \cup V_k$ com $|V_i| = m_i$, $i \in \{1,2,\ldots,k\}$. O j-ésimo elemento de V_i será denotado por $v_{i,j}$, $i \in \{1,2,\ldots,k\}$, $j \in \{1,2,\ldots,m_i\}$. No caso do grafo k-partido $K_{m_1,m_2,\ldots,m_k}$, há uma aresta ligando cada par de vértices $(v_{i,j}, v_{l,n})$, $v_{i,j} \in V_i$, $v_{l,n} \in V_l$, $l \neq i$. Além disso, não há dois vértices em V_i, $i \in \{1,2,\ldots,k\}$, que sejam ligados por uma aresta. Para $i \in \{1,2,\ldots,k\}$, define-se*

$$Q_i = \sum_{j=1}^{m_1} P(v_{i,j})$$

$$\varphi_i(S) = \left(Q_i^S - \sum_{j=1}^{m_i} \left(\frac{P(v_{i,j})}{1 - Q_i + P(v_{i,j})} \right)^S \right) \log(S)$$

Então,

$$H_\iota(K_{m_1,m_2,\ldots,m_k}, P) = H(P) - \sum_{S=2}^{\infty} \sum_{i:m_i \geq 2} (1 - Q_i)\varphi_i(S).$$

A partir do Teorema 2.5.4 é possível definir

$$\begin{aligned} H_\iota(K_{m_1,m_2,\ldots,m_k}, P) &= H(P) - \sum_{S=2}^{\infty} \sum_{i:m_i \geq 2} (1 - Q_i)\varphi_i(S) \\ &= H(P) - L(P) \end{aligned}$$

Desta forma, há pelo menos $2^{nL(P)}$ sequências típicas de comprimento n que passam a pertencer a uma classe de intercâmbio devido às relações de dependência/independência. Quando o grafo é completo $L(P) = 0$ de forma que cada palavra só gera ela mesma como classe. Quando o grafo não é completo, há um menor número de classes de intercâmbio, possibilitando uma compressão maior do que a definida pela entropia de Shannon que não considera símbolos com ordem parcial. Quando o grafo é vazio, como foi visto, a entropia de intercâmbio é zero.

Corolário 2.5.1. *Assumindo uma fonte discreta e sem memória com distribuição de probabilidade sobre o conjunto de vértices $V(G)$. Se G não é um grafo completo em $V(G)$, então $H_\iota(G,P) < H(P)$.*

2.5.1 Exemplos

Exemplo 2.5.1. *Considerando o grafo apresentado na Fig. 5, é possível verificar que ele é um grafo bipartido completo, já que $V = V_1 \cup V_2 = \{a,c\} \cup \{b\}, \{a,c\} \cap \{b\} = \emptyset$ e todos os elementos da partição V_1 são conectados por uma aresta a todos os outros elementos da partição V_2.*

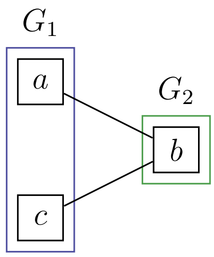


Figura 5 – Grafo $K_{2,1}$ bipartido completo.

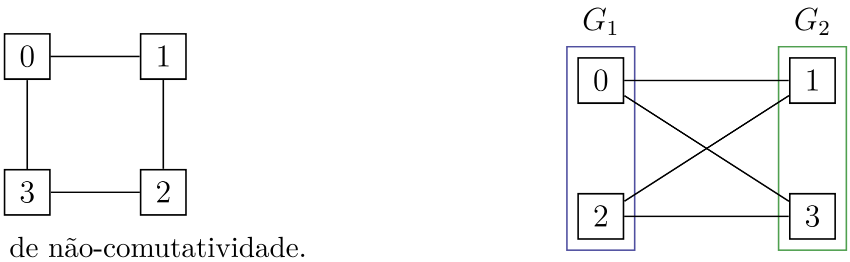
Desta forma, $V_1 = \{v_{1,1} = a, v_{1,2} = c\}$, $|V_1| = m_1 = 2$, $V_2 = \{v_{2,1} = b\}$, $|V_2| = m_2 = 1$, $Q_1 = P(a) + P(c) = 2/3$ *e* $Q_2 = P(b) = 1/3$.

$$\begin{aligned}
\varphi_1(S) &= \left(Q_1^S - \sum_{j=1}^{m_1}\left(\frac{P(v_{1,j})}{1 - Q_1 + P(v_{1,j})}\right)^S\right)\log(S) = \\
&= \left((2/3)^S - \left(\frac{1/3}{1-2/3+1/3}\right)^S - \left(\frac{1/3}{1-2/3+3}\right)^S\right)\log(S) \\
&= \left((2/3)^S - 2(1/2)^S\right)\log(S).
\end{aligned}$$

Assim, calculando $H_\iota(K_{2,1}, P)$*, tem-se*

$$\begin{aligned}
H_\iota(K_{2,1}, P) = H(P) - \sum_{S=2}^{\infty}(1 - Q_1)\left(\varphi_1(S)\right) \\
&= \log(3) - (1/3) \cdot \sum_{S=2}^{\infty}\left((2/3)^S - 2(1/2)^S\right)\log(S) \\
&\approx 1,58496 - 0,308518 \approx 1.27644\, bit.
\end{aligned}$$

Exemplo 2.5.2. *Considere o grafo da Fig. 6a. É possível verificar que esse grafo é um $K_{2,2}$ bipartido completo, como pode ser observado na Fig. 6b.*



(a) Grafo de não-comutatividade.

(b) Grafo $K_{2,2}$ bipartido completo.

Figura 6 – Modelo de fonte concorrente.

A partição é dada por $V = V_1 \cup V_2$ *com* $V_1 = \{v_{1,1} = 0, v_{1,2} = 2\}$ *e* $V_2 = \{v_{2,1} = 1, v_{1,3} = 2\}$*. Desta forma,* $|V_1| = m_1 = 2$ *e* $|V_2| = m_2 = 2$*.*

Realizando os cálculos dos termos Q_1*,* Q_2*,* $\varphi_1(S)$ *e* $\varphi_2(S)$*, tem-se*

$$Q_1 = \sum_{j=1}^{m_1} P(v_{1,j}) = P(0) + P(2) = 1/2$$

$$Q_2 = \sum_{j=1}^{m_2} P(v_{2,j}) = P(1) + P(3) = 1/2$$

$$\begin{aligned}\varphi_1(S) &= \left(Q_1^S - \sum_{j=1}^{m_1}\left(\frac{P(v_{1,j})}{1 - Q_1 + P(v_{1,j})}\right)^S\right)\log(S)\\ &= \left((1/2)^S - 2(1/3)^S\right)\log(S)\end{aligned}$$

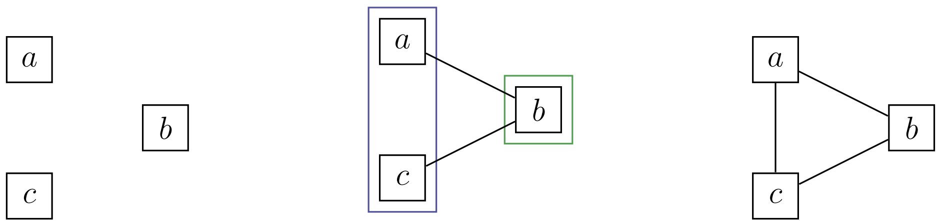
$$\begin{aligned}\varphi_2(S) &= \left(Q_2^S - \sum_{j=1}^{m_2}\left(\frac{P(v_{2,j})}{1 - Q_1 + P(v_{2,j})}\right)^S\right)\log(S)\\ &= \left((1/2)^S - 2(1/3)^S\right)\log(S).\end{aligned}$$

Assim, a entropia de intercâmbio, nesse caso, vale

$$\begin{aligned}H_\iota(K_{2,1}, P) &= H(P) - \sum_{S=2}^{\infty}\left((1 - Q_1)\left(\varphi_1(S)\right) + (1 - Q_2)\left(\varphi_2(S)\right)\right)\\ &= \log(4) - \sum_{S=2}^{\infty}\left((1/2)^S - 2(1/3)^S\right)\log(S)\\ &\approx 2 - 0{,}313461 \approx 1{,}68654\, bit.\end{aligned}$$

3 Algoritmos para Compressão de Sequências de Eventos com Ordem Parcial

São propostos dois novos algoritmos para comprimir sequências de eventos com ordem parcial e estimar a entropia de intercâmbio associada aos grafos de não-comutatividade. Para exemplificar o funcionamento dos algoritmos, será considerada inicialmente uma fonte com 3 símbolos equiprováveis e sem memória. Considerando essa fonte, os grafos de não-comutatividade que serão analisados são apresentados na Fig. 7. Todas as fontes consideradas nessa Seção serão discretas, sem memória com uma distribuição equiprovável nos vértices V do grafo de não-comutatividade.



(a) Grafo G_1. (b) Grafo $G_2 = K_{2,1}$ bipartido completo. (c) Grafo G_3

Figura 7 – Grafos de não-comutatividade com $|V| = 3$.

Uma sequência de eventos gerada pela fonte descrita pode ser:

$$\mathbf{x} = aaaaababaaccabbccabbabcbbcaabccbabcbaacbccaaccbcbaacbabcabcbabbbaccbbcacabbab \tag{3.1}$$

em que $|\mathbf{x}| = n = 77$, $N(a|\mathbf{x}) = 27$, $N(b|\mathbf{x}) = 28$, $N(c|\mathbf{x}) = 22$.

Para comprimir uma sequência de eventos, é necessário identificar as classes de intercâmbio mais recorrentes na sequência. Isso é feito porque, como todas as palavras de uma mesma classe de intercâmbio são equivalentes, é possível criar um dicionário de regras em que se refere à classe não mais à sub-sequência individualmente.

3.1 Algoritmo 1

A estratégia adotada nesse algoritmo é identificar as classes de intercâmbio de comprimento w que possuem mais elementos em uma sequência e substituir todas as ocorrência desses elementos por uma sub-sequência equivalente. Para verificar a congruência entre duas sub-sequências é utilizado um algoritmo baseado no Teorema 2.3.1.

Como se pode observar pelas frequências relativas dos símbolos obtidas na sequência 3.1, a forma que se faz a verificação de pertencimento de uma sub-sequência a uma classe de intercâmbio tem influência na compressão obtida. Para isso, é definida outra variável, s, que se refere ao passo de verificação das sub-sequências. Esse parâmetro tem efeito sobre o tempo de execução do algoritmo. Por exemplo, usando a sequência 3.1 com comprimento da sub-sequência $w = 4$ e o passo de $s = 3$, tem-se

$$1 : \boxed{\mathrm{aaaa}}\,ababaaccabbccabbabcbbcaabccbabcbaacbccaaccbcbaacbabcabcbabbbaccbbcacabbab$$

$$2 : aaa\boxed{\mathrm{aaba}}\,baaccabbccabbabcbbcaabccbabcbaacbccaaccbcbaacbabcabcbabbbaccbbcacabbab$$

$$3 : aaaaab\boxed{\mathrm{abaa}}\,ccabbccabbabcbbcaabccbabcbaacbccaaccbcbaacbabcabcbabbbaccbbcacabbab$$

Inicialmente, (1) seleciona-se a sub-sequência *aaaa* e (2) verifica-se se *aaba* é congruente à primeira usando como exemplo o grafo G_2 da Fig. 7b. A próxima sub-sequência a ser (3) verificada é *abaa* e o processo continua até o fim da sequência $\mathbf{x}$. O algoritmo não está otimizado porque as palavras que pertencem a mesma classe de intercâmbio não precisariam ser visitadas novamente. Se o número de ocorrências de palavras de uma mesma classe de intercâmbio na sequência for maior do que um, uma palavra desse traço é salva no dicionário. Cabe notar que o tamanho do dicionário vai ser menor ou igual ao número de classes de intercâmbio com comprimento w presentes na sequência original.

Quando todos os pares de sub-sequências são verificados, as sub-sequências presentes no dicionário são substituídas na sequência principal. Desta forma, qualquer sub-sequência da sequência original da mesma classe de intercâmbio de uma sub-sequência presente no dicionário vai ser substituída por esta última.

Essa substituição é feita começando pelas classes de intercâmbio do dicionário que possuem maior frequência de ocorrência na sequência original para que haja maior eficiência na redução do número de símbolos. Isso se deve ao fato de que quanto maior o número de ocorrência de palavras de uma certa classe, maior o valor da compressão da sequência.

Pode acontecer de algumas sub-sequências terem partes substituídas nesse processo e não mais serem congruentes a outro elemento do dicionário. Por exemplo, supondo que há uma sub-sequência *abcaabc* em que haja duas sub-sequências com o número de ocorrências maior do que um, $A = abca$ e $B = aabc$ só que *abca* pertence à classe de intercâmbio que ocorre mais do que a classe a qual *aabc* pertence. No processo de substituição,*abcaabc* se torna $Aabc$. Desta forma, podem haver sub-sequências no dicionário que não fazem parte da sequência comprimida, então essas sub-sequências são retiradas do dicionário. Sub-sequências do dicionário que só ocorrem uma vez também são retiradas e substituídas na sequência original para reduzir em um símbolo na representação.

Usando o algoritmo descrito, é possível verificar que algumas das sub-sequências que possuem mais elementos dos seus traços presentes na sequência 3.1 são apresentados a seguir.

$$aaaaababa\boxed{\text{acca}}b\boxed{\text{bcca}}bbabcb\boxed{\text{bcaa}}bccbabc\boxed{\text{baac}}bcc\boxed{\text{aacc}}bc\boxed{\text{baac}}babcabcbabb\boxed{\text{bacc}}bb\boxed{\text{caca}}bbab$$

As sub-sequências com a mesma cor estão contidas na mesma classe de tipo em relação ao grafo G_2 da Fig. 7b, então podem ser substituídas por qualquer palavra do traço gerado por elas. Dessa forma, o dicionário de regras é formado substituindo as sub-sequências que apresentam as mesmas classes de intercâmbio por só uma sub-sequência equivalente e criando uma regra pra isso.

Na Tabela 2, é apresentada o dicionário de regras obtido usando $w = 4$ e $s = 3$ na sequência 3.1.

Regra	**Expressão**	**Comprimento**
A	aaaaababaBbEbbabcbCbcDcCbccBbcCbabcabDbEbbBbbab	47
B	acca	4
C	bcaa	4
D	cbaa	4
E	bcca	4
Total		63

Tabela 2 – Dicionário gerado para a sequência 3.1 com $w = 4$ e $s = 3$.

O tamanho de um dicionário de regras é medido pelo número de símbolos das expressões (coluna 2 da tabela de regras). Desta forma, é possível observar que houve a redução de 77 símbolos da sequência inicial para 63 símbolos do dicionário que possibilita a construção de uma sequência equivalente a 3.1, em relação ao grafo G_2 da Fig. 7b. Na Tabela 2, é apresentado o dicionário obtido usando $w = 3$ e $s = 3$ na sequência 3.1, em que se obteve um dicionário usando 58 símbolos de forma que é possível construir uma sequência equivalente.

Regra	**Expressão**	**Comprimento**
A	aaaaababaBCBbbaDEabcFDaacbBBDaaFEDCbBbbBCab	43
B	acc	3
C	abb	3
D	bcb	3
E	bca	3
F	cba	3
Total		58

Tabela 3 – Dicionário gerado para a sequência 3.1 com $w = 3$ e $s = 3$.

De forma geral, considerando uma sequência S com n eventos com um passo de s e uma janela de w, o número de subsequências possíveis consideradas pelo Algoritmo 1 será

$$\#subseq = \left\lceil \frac{n-w+1}{s} \right\rceil.$$

Assim, o número de comparações entre essas sequências valerá no máximo

$$\#comparacoes = \left(\left\lceil \frac{n-w+1}{s} \right\rceil \right)^2.$$

Como é verificado se duas subsequências de tamanho w são congruentes em, no máximo $2w|E|$ operações, então o número máximo de operações do Algoritmo 1 é dado por

$$\#operacoes = 2w|E| \left(\left\lceil \frac{n-w+1}{s} \right\rceil \right)^2.$$

3.2 Algoritmo 2

O segundo algoritmo consiste em gerar as palavras possíveis e agrupá-las de acordo com suas classes de intercâmbio a partir de um grafo de não-comutatividade $G = (V, E)$ e do alfabeto definido por V. Tendo gerado as classes de intercâmbio, pode-se estimar a entropia de intercâmbio e comprimir sequências de forma ótima.

As palavras são geradas de acordo com o alfabeto V e existirão $|V|^w$ palavras com comprimento w. O primeiro passo é gerar essas palavras de acordo com V e w. A geração de todas as palavras corresponde ao arranjo com repetições dos símbolos do alfabeto V e foi utilizada uma função que utiliza uma estrutura baseada na árvore apresentada na Fig. 8.

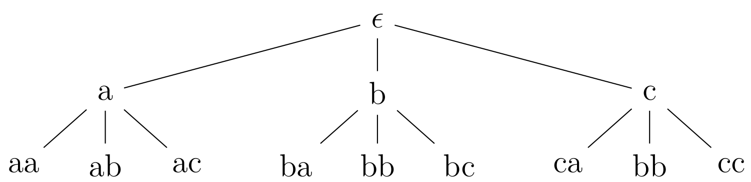


Figura 8 – Estrutura em árvore para a geração das sequências com $V = \{a, b, c\}$, $w = 2$.

Adiciona-se os elementos de V em uma lista $L = \{a, b, c\}$. Uma nova lista L' será formada pela concatenação dos elementos da primeira lista com cada um dos elementos de V. Então, os novos elementos a partir de L, serão aa, ab, ac a partir de a, ba, bb, bc a partir de b e assim por diante. Quando todos os ramos forem visitados, a lista L assume os valores de L'. O processo continua até as palavras formadas terem o comprimento w definido, gerando a lista L_w, requirindo um total de $\sum_{k=1}^{w} |V|^k$ operações.

A segunda parte do algoritmo é formar os traços a partir das palavras geradas na primeira parte. Define-se um dicionário que vai ter uma palavra de um traço como chave e as palavras desse traço como valores. Para cada palavra do conjunto de palavras gerado na primeira parte do algoritmo, verifica-se se ela pertence a alguma classe presente no dicionário. Isso é feita por meio da verificação de congruência entre a palavra e a chave. Se a nova palavra pertence a uma classe do dicionário, então é adicionada na lista apontada pela chave. Se essa palavra não pertence, então ela é adicionada ao dicionário como uma nova chave e o processo continua até todas as palavras de comprimento w pertençam a um traço.

O Algoritmo 2 foi aplicado à sequência 3.1 com $w = 4$ e obteve-se o dicionário de regras apresentado na Tabela 4. É possível observar que o dicionário gerado é semelhante no tamanho do gerado pelo Algoritmo 1 na Tabela 2, mas as regras se referem a classes distintas nas regras D e E.

Regra	**Expressão**	**Comprimento**
A	aaaaababBabbDbabcbCbccECbccBbcCEabcbabbbDbBbbab	47
B	aacc	4
C	baac	4
D	accb	4
E	babc	4
Total		63

Tabela 4 – Dicionário gerado para a sequência 3.1 usando $w = 4$ no Algoritmo 2.

O Algoritmo 2 foi aplicado à sequência 3.1 com $w = 3$ e obteve-se o dicionário de regras apresentado na Tabela 5. Nesse caso, o dicionário gerado foi 56, que é menor em dois símbolos em relação ao apresentado pelo Algoritmo 1 na Tabela 3. Além disso, o dicionário gerado pelo Algoritmo 2 possui 4 regras ao invés de 6 do Algoritmo 1 (ver Tabela 3).

Regra	**Expressão**	**Comprimento**
A	aaaaaDaBabbBbDcbbCbccDcbCbBBbcbCDacbcDbbBbbBabD	47
B	acc	3
C	aac	3
D	bab	3
Total		56

Tabela 5 – Dicionário gerado para a expressão 3.1 usando $w = 3$ no Algoritmo 2.

No pior caso, que corresponde ao grafo G ser completo, cada palavra corresponderá a um traço, exigindo que a verificação de uma nova palavra seja feita com cada uma das chaves do dicionário, então o algoritmo tem uma complexidade $O(|L_w|^2)$ ou ainda $O(|V|^{2w})$.

No fim da execução, o dicionário conterá todos os traços gerados por todas as palavras de comprimento w. Esse dicionário pode ser utilizado para gerar um dicionário de regras para comprimir sequências com eventos com ordem parcial ou para estimar a entropia de intercâmbio do grafo G usando a Eq. 4.2.

4 Resultados

Foram produzidas sequências de vários comprimentos e foi verificado qual a influência do comprimento e do deslocamento da sub-sequência no nível na redução do número de símbolos. O computador utilizado na geração dos resultados possui um Intel Core i7, relógio de 2,7 GHz e uma memória RAM de 8 GB.

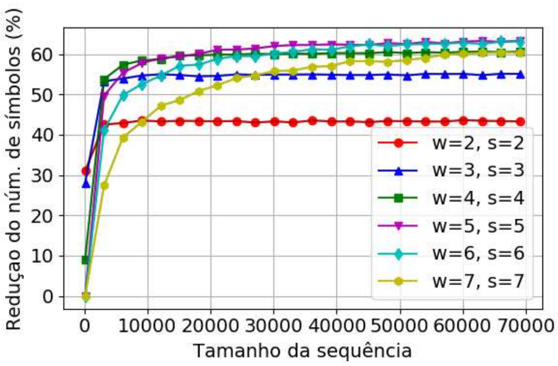
4.1 Algoritmo 1

4.1.1 Compressão de sequências de eventos com ordem parcial

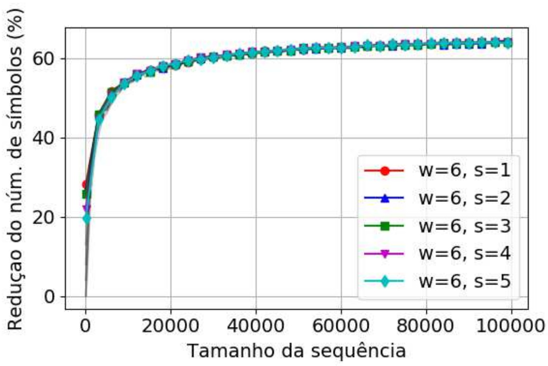
A avaliação de algoritmos que geram dicionários costumam comparar o número de símbolos da sequência original com o tamanho do dicionário gerado. O tamanho de um dicionário de regras é medido pelo número de símbolos das expressões de cada uma das regras. Isso porque usar teoria de distorção em alguns casos se torna uma tarefa difícil. Desta forma, a redução do número de símbolos pode ser expressa por meio da Eq. 4.1.

$$\text{Redução}(\%) = 100\left(1 - \frac{\text{Núm. de símbolos no dicionário}}{\text{Núm. de símbolos da sequência original}}\right) \tag{4.1}$$

Os comportamentos da redução em função do comprimento e do deslocamento podem ser observados na Fig. 9a e Fig. 9b, respectivamente.



(a) Redução do número de símbolos em função do comprimento w das sub-sequências.



(b) Redução do número de símbolos em função do deslocamento s das sub-sequências.

Figura 9 – Variação da redução para diferentes valores de w e s.

Os maiores valores de redução foram obtidos para $w = 5$ com o tamanho da sequência maior do que 20×10^3 e menor do que 70×10^3. Foram testados valores de

$w = 2, 3, 4, 5, 6, 7$. Há uma aparente insensibilidade ao deslocamento que se deve às características da fonte, como estacionariedade. Se os símbolos não fossem equiprováveis, provavelmente a redução teria uma correlação maior com o deslocamento, de forma que deslocamentos menores produziriam maiores reduções.

Observando os grafos G_1, G_2 e G_3 da Fig. 7, é esperado que reduções baseadas no grafo G_1 sejam as maiores, já que todos os eventos são comutativos em relação ao grafo e quaisquer par de sub-sequências pertencentes ao mesmo traço serão congruentes. As menores reduções serão para o grafo G_1 em que nenhum símbolo é independente dos outros e, portanto, só pares de sub-sequências iguais serão congruentes uma a outra, o que caracteriza o caso em que não há ordem parcial entre os eventos. A redução intermediária será obtida no caso de G_2, pois há uma ordem parcial. Sequências foram geradas e os valores de redução foram computados para os três grafos e apresentados na Fig.10.

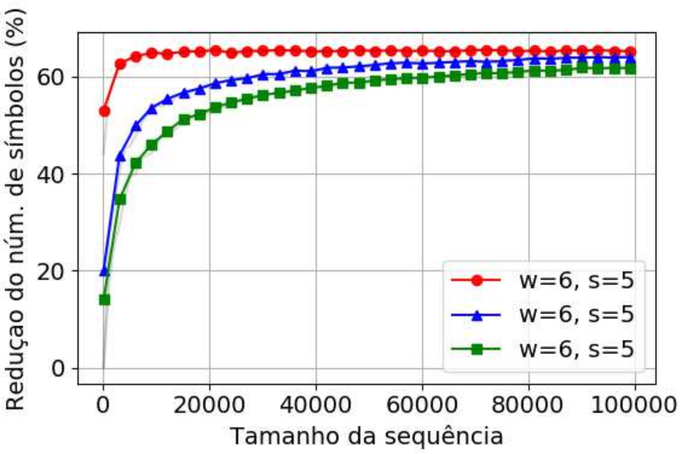


Figura 10 – Redução do número de símbolos para os grafos G_1 ($-\bullet$) G_2 ($-\triangle$) e G_3 ($-\square$).

Considerando uma fonte sem memória com 4 símbolos, todos equiprováveis, e o grafo G_4 da Fig. 11a, foram geradas sequências com o comprimento entre 100 símbolos e 70×10^3 e foram verificadas as reduções que são apresentadas na Fig. 12.

(a) Grafo $G_4 = K_{2,2}$ bipartido completo.

(b) Grafo G_5.

Figura 11 – Grafo de não comutatividade e modelo de uma fonte.

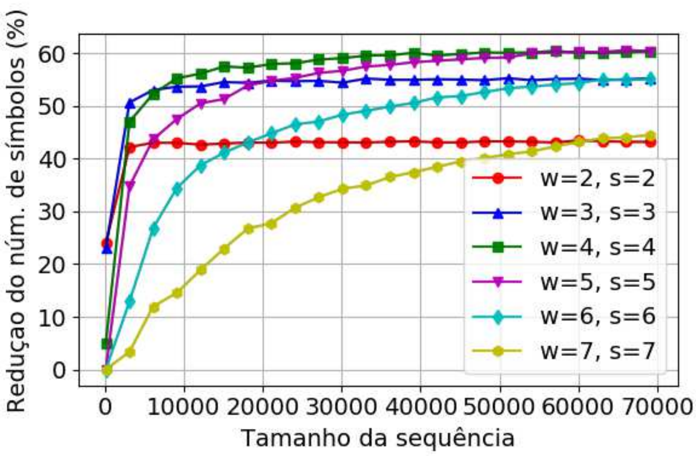


Figura 12 – Redução do número de símbolos em função do comprimento w das subsequências.

Os maiores valores de redução foram obtidos para $w = 5$ com o tamanho da sequência maior do que 50×10^3 e menor do que 70×10^3. Foram testados valores de $w = 2, 3, 4, 5, 6, 7$. Como há insensibilidade ao passo s, os passos escolhidos foram iguais aos valores de w para aumentar a velocidade de execução do algoritmo.

A abordagem desse algoritmo, apesar de não ser ótima, oferece vantagens como só adicionar ao dicionário classes de intercâmbio que ocorrem na sequência original, sem a necessidade de gerar todas as classes de intercâmbio. Isso pode influenciar no desempenho da compressão de sequências com símbolos não equiprováveis.

4.1.2 Estimação da Entropia de Intercâmbio

Como foi visto na Seção 2.5, sequências típicas de comprimento n estão contidas em aproximadamente $2^{nH_\iota(G,P)}$ classes de intercâmbio típicas. Desta forma, se for possível determinar quantas classes de intercâmbio distintas de comprimento w estão presentes em uma sequência, é possível obter uma estimativa da entropia de intercâmbio do grafo $H_\iota(G, P)$ a partir de

$$\hat{H}_\iota(G, P) \approx \frac{\log_2 (\#Classes - de - interc\hat{a}mbio)}{w}. \tag{4.2}$$

A hipótese inicial era de que o w associado à maior compressão observada também seria o valor com estimativa para a entropia de intercâmbio mais próxima da teórica, mas isso não se confirmou, pelo menos com w variando entre 2 e 13. Usando o algoritmo para determinar o w que produz a maior compressão com grafo G_3, foi obtido que $w = 7$ e o número de classes equivalentes foi 2187 de forma que $\log_2(2187)/7 \approx 1,58496$, valor que corresponde à entropia (de Shannon) para uma fonte com 3 símbolos equiprováveis. Isso decorre do fato que o grafo G_3 corresponde ao caso convencional em que os símbolos não podem ser trocados de posição, de forma que o número de classes de intercâmbio equivale ao número de classes de tipo que obedece à relação determinada no Teorema 2.2.2.

As entropias dos grafos G_1, G_2, G_3, G_4 e G_5 da Fig. 13 foram estimadas com o w que produziu os maiores valores de redução entre os valores de $w = 2, 3 \ldots, 12, 13$ e sequências com 2×10^6 símbolos. As estimativas são apresentadas na Tabela 6. Na Tabela 6, $|\hat{B}_n|$ representa o número de classes classes de intercâmbio de tamanho n em relação a um grafo de não-comutatividade G contadas nas sequências pelo algoritmo.

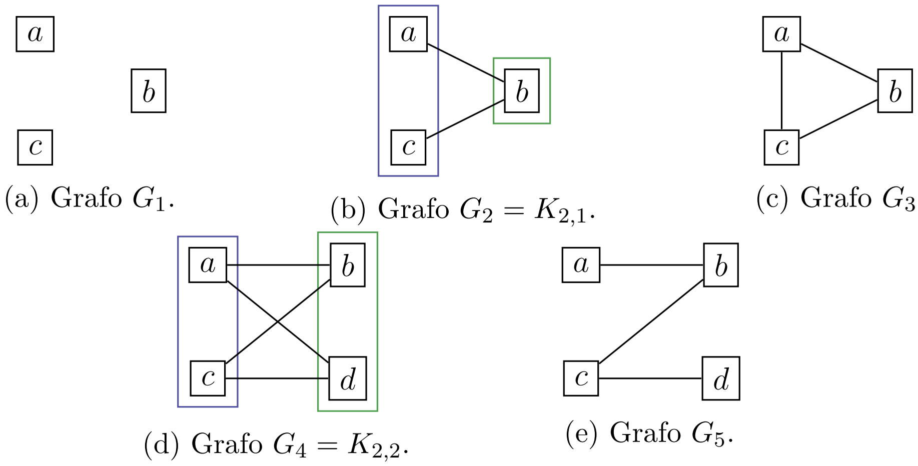


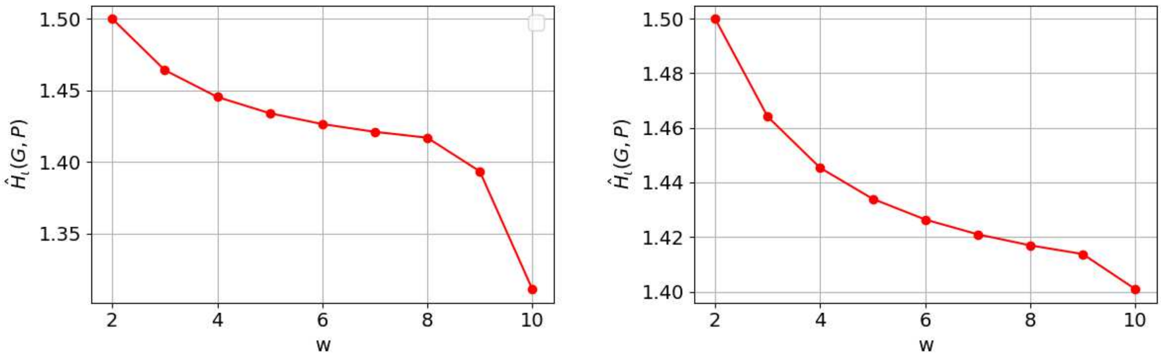
Figura 13 – Grafos de não-comutatividade usados para estimar a entropia de intercâmbio.

Grafo	$H(H)$	$H_\iota(X)$	$\lvert B_n\rvert$	w	$\hat{H}_\iota(G,P)$	$\hat{H}_\iota(G,P)$ $w=2$
G_1	1,58496	0,00	1	10	0,00	0,00
G_2	1,58496	1,27644	987	7	1,4209	1,50
G_3	1,58496	1,58496	2187	7	1,58496	1,58496
G_4	2,00	1,68654	1905	6	1,81681	1,90367
G_5	2,00	$< 2,00$	3279	7	1,66843	1,85021

Tabela 6 – Estimativas de entropia para os grafos.

Como G_1 é um grafo vazio, então qualquer par de sub-sequências com o mesmo comprimento é congruente. Desta forma, só existe uma classe de intercâmbio e a entropia de intercâmbio é zero, independente do comprimento w. Para G_2, que é um grafo bipartido completo, obteve-se uma estimativa de 1,4209 *bits*. A estimativa inicial com $w = 2$ já é menor do que a entropia de Shannon. Para G_3, que é um grafo completo, a entropia de intercâmbio se iguala à entropia de Shannon e desde a primeira estimativa com $w = 2$ o resultado é igual à entropia de Shannon. O grafo G_4 é um bipartido completo com entropia de intercâmbio igual a 1,68654 *bits* e a estimativa foi de 1,81681 *bits*. A estimativa inicial com $w = 2$ também forneceu um valor menor do que a entropia de Shannon. O grafo G_5 tem uma relação de dependência a menos em relação à G_4. Assim, é esperado que haja menos classes de intercâmbio distintas em relação à G_4 para o mesmo comprimento w.

Na Fig. 14 são apresentadas as estimativas para o grafo G_2 baseadas em sequências com comprimentos iguais a 1×10^6 e 2×10^6.



(a) Estimação usando sequências de comprimento igual a 1×10^6.

(b) Estimação usando sequências de comprimento igual a 2×10^6.

Figura 14 – Estimação da Entropia de Intercâmbio para G_2 para diferentes valores de w.

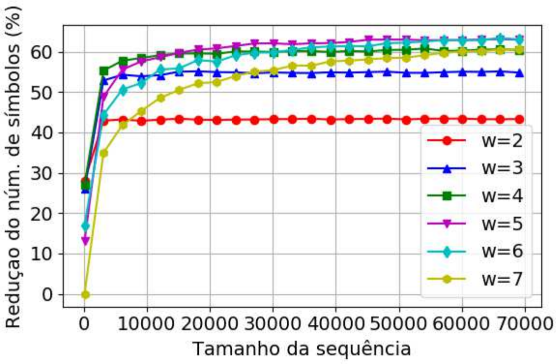
A queda brusca da estimativa da entropia de intercâmbio para $w = 9$ e $w = 10$ apresentada na Fig. 14a se deve ao fato que, conforme o comprimento w cresce, aumenta também o número de classes de intercâmbio distintas. Desta forma, a probabilidade de ocorrência de cada uma das classes diminui. Considerando o valor esperado do número de ocorrências de cada classe, o comprimento da sequência não é suficiente para fazer a identificação do número correto das classes. Assim, os números de classes de intercâmbio identificado é menor do que os números corretos, fazendo a estimativa diminuir. A solução pra isso é crescer o tamanho da sequência, o que acarreta o aumento do tempo de processamento. Pela Fig. 14b, dobrar o comprimento da sequência fez com que fosse possível estimar corretamente o número de classes de intercâmbio para $w = 9$, mas para $w = 10$ já houve um erro significativo.

4.2 Algoritmo 2

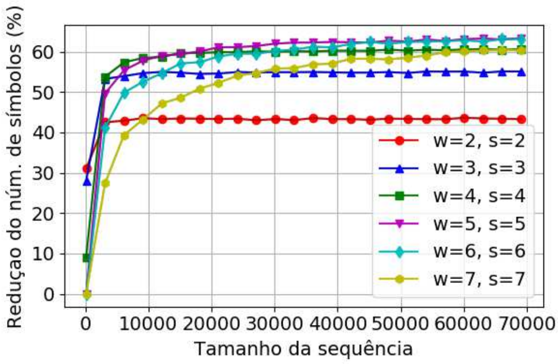
4.2.1 Compressão de sequências de eventos com ordem parcial

Para fazer a compressão de sequências utiliza-se o dicionário de classes gerado. Para cada classe desse dicionário, verifica-se o número de ocorrências dessa classe na sequência a ser comprimida. Desta forma, a classe que tem mais ocorrências é substituída na sequência original e a chave do dicionário de classes é adicionada ao dicionário de regras. Esse processo se repete para todas as classes. Como pode haver casos em que uma classe só aparece uma vez na sequência após a substituição das outras, é necessário substituir a regra na sequência original e apagar essa regra do dicionário de regras, como no caso do Algoritmo 1 (ver Seção 3.1) para economizar alguns símbolos.

Foram geradas sequências com comprimento variando entre 100 e 70×10^3 e foi verificado o comportamento da redução do número de símbolos para valores de $w = 2, 3, \ldots, 7$. As compressões obtidas para o grafo G_2 são apresentadas na Fig. 15a.



(a) Algoritmo 2 (cerca 26 segundos)



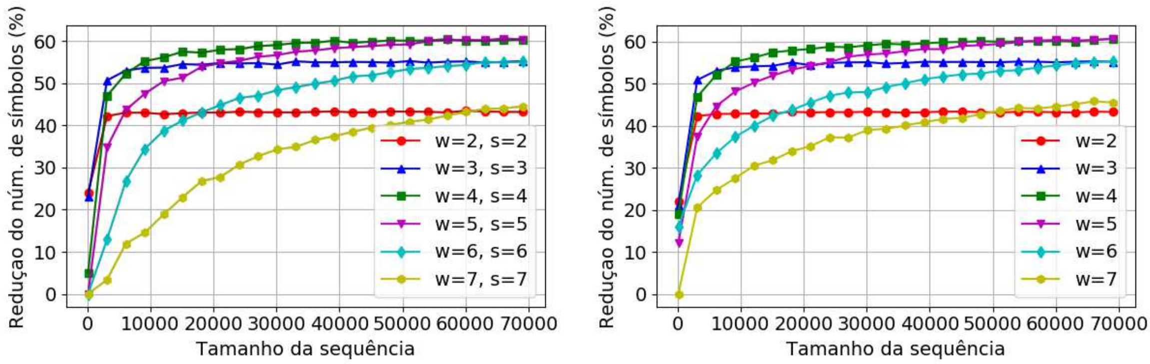
(b) Algoritmo 1 (cerca de 200 segundos)

Figura 15 – Redução do número de símbolos de sequências usando os Algoritmos 1 e 2 no Grafo G_2.

De forma geral, os resultados do Algoritmo 2 (Fig. 15a) foram semelhantes aos resultados do Algoritmo 1 (Fig. 15b). Para sequências menores, o Algoritmo 2 apresentou uma maior taxa de compressão do que o Algoritmo 1 devido à geração de todas as classes de intercâmbio ao invés de utilizar os passos entre as comparações.

Em termos de tempo de execução, o Algoritmo 2 fez as reduções para todos os casos descritos em cerca de 26 segundos, enquanto o Algoritmo 1 levou cerca de 200 segundos. Para valores pequenos de w, a geração das classes de intercâmbio usando o Algoritmo 2 é rápida, sendo o maior tempo associado à verificação da frequência de ocorrência das classes e à geração do dicionário de regras. Enquanto o Algoritmo 1 precisa fazer as verificações de todos os pares de sub-sequências especificadas a partir de w e s.

Foram geradas sequências com comprimento variando entre 100 e 70×10^3 e foi verificado o comportamento da redução do número de símbolos para valores de $w = 2, 3, \ldots, 7$. As compressões obtidas para o grafo G_4 são apresentados na Fig. 16b.



(a) Algoritmo 1 (cerca 711 segundos) (b) Algoritmo 2 (cerca de 155 segundos)

Figura 16 – Redução do número de símbolos de sequências usando os Algoritmos 1 e 2 no Grafo G_4.

Como no caso do G_2, o Algoritmo 2 produziu melhores resultados do que o Algoritmo 1 para sequências menores. Para sequências com comprimentos superiores a 60×10^3, os resultados são semelhantes. O Algoritmo 1 fez as reduções para todos os casos descritos em cerca de 155 segundos, enquanto o Algoritmo 1 levou cerca de 711 segundos. Isso se deve aos motivos explicados anteriormente. Cabe notar que aumentando o número de símbolos no alfabeto de 3 para 4 fez com que o tempo de execução do Algoritmo 2 subisse de 26 segundos para 155 segundos, enquanto o tempo de execução do Algoritmo 1 subiu de 200 segundos para 711 segundos. Isso decore do fato de que o tempo de execução do algoritmo cresce exponencialmente com o número de símbolos do alfabeto. Desta forma, com comprimentos w e alfabetos maiores, o Algoritmo 2 se torna impraticável.

4.2.2 Estimação da Entropia de Intercâmbio

Utilizou-se a Eq. 4.2 para estimar a entropia de intercâmbio para o grafo G_2, para os dicionários de classes de intercâmbio gerados para diferentes valores de w. As estimações são apresentadas na Fig. 17.

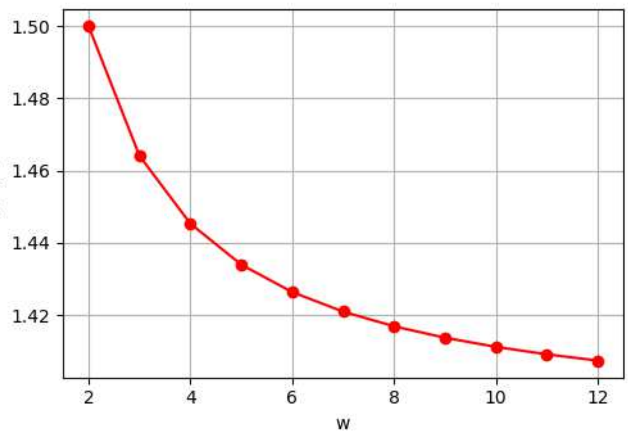


Figura 17 – Estimação da entropia da intercâmbio para G_2 usando o Algoritmo 2.

Como todas as classes de intercâmbio são geradas, a estimação não depende de sequências e não acontece o problema apresentado na Fig. 14 do Algoritmo 1.

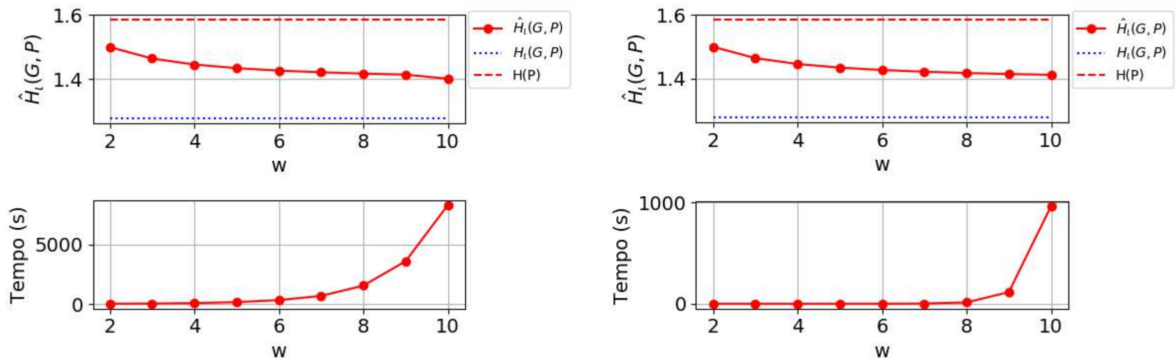
5 Discussões

Foram apresentados dois algoritmos para realizar a compressão de sequências de eventos com ordem parcial. O primeiro consiste na identificação de classes de intercâmbio presentes em sequências e na criação de um dicionário de regras com essas classes. O segundo algoritmo consiste na geração das classes de intercâmbio e na geração de um dicionário de regras de acordo com a sequência a ser comprimida.

Apesar do Algoritmo 1 não ser ótimo para a compressão de sequências no sentido de não ter garantia de identificar todos os elementos da classe de intercâmbio de comprimento da janela utilizada, dependendo do passo selecionado, ele obteve um desempenho semelhante ao Algoritmo 2 conforme o comprimento das sequências aumentava. Como o dicionário de regras é formado diretamente das sequências, no caso em que os símbolos não são equiprováveis, esse algoritmo pode apresentar um desempenho melhor em relação ao Algoritmo 1 em termos de tempo conforme o comprimento w aumenta. Entretanto, para a estimação da entropia de intercâmbio de um grafo, é necessário escolher o tamanho das sequências para que não haja uma identificação de classes de intercâmbio menor do que a real.

O Algoritmo 2 é ótimo, mas tem um custo exponencial em w e polinomial em $|\Sigma|$, o que o torna inviável para aplicações com valores w grandes. Entretanto, para valores pequenos de w, o Algoritmo 2 tem um tempo de execução menor que o Algoritmo 1, além de não depender do comprimento das sequências.

Na Fig. 18 são apresentados os tempos para a estimação da entropia de intercâmbio para G_2 com diferentes valores de w usando os dois algoritmos. As estimações usando o Algoritmo 1 foram feitas com sequências de comprimento igual a 2×10^6.

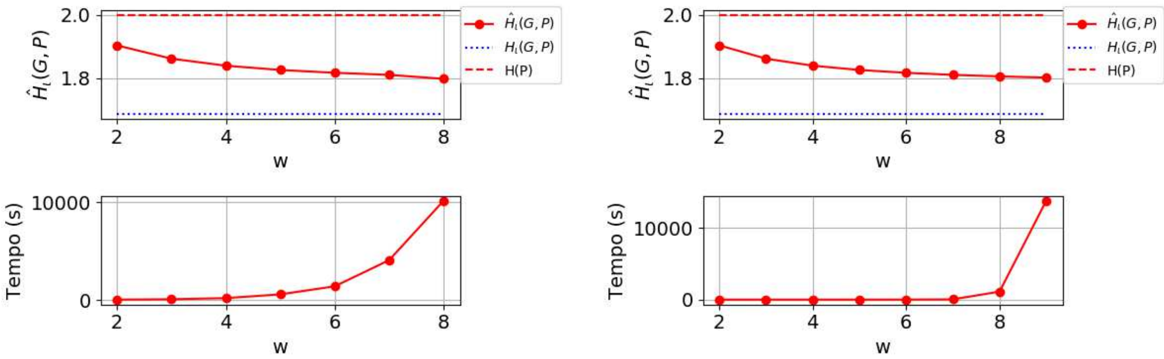


(a) Algoritmo 1. (b) Algoritmo 2

Figura 18 – Tempos para a estimação da entropia de intercâmbio para o grafo G_2.

O tempo para estimar a entropia de intercâmbio com $w = 10$ passou dos 5×10^3 segundos para o Algoritmo 1, como pode ser visto na Fig. 18a, além de produzir um valor um pouco abaixo do correto para a estimação da entropia de intercâmbio. No mesmo caso, o Algoritmo 2 obteve um resultado acurado em 1×10^3 segundos, um tempo 5 vezes menor.

Na Fig. 19 são apresentados os tempos para a estimação da entropia de intercâmbio para G_4 com diferentes valores de w usando os dois algoritmos. As estimações usando o Algoritmo 1 foram feitas com sequências de comprimento igual a 2×10^6.



(a) Algoritmo 1. (b) Algoritmo 2

Figura 19 – Tempos para a estimação da entropia de intercâmbio para o grafo G_4.

Para o caso de $w = 9$, o Algoritmo 2 produziu a estimação corretamente, mas levou mais tempo em relação ao Algoritmo 1. Isso pode ser observado na Fig. 19a e Fig. 19b. Como o Algoritmo 2 gera todas as sequências de comprimento w e depois agrupa, o tempo aumenta rápido conforme w aumenta, já que o número de sequências é cresce exponencialmente.

É possível modificar os algoritmos de acordo com outros algoritmos de compressão como o SEQUITUR (NEVILL-MANNING; WITTEN, 1997) ou algum da família do Lempel-Ziv para que haja ainda mais compressão nas sequências. Além disso, é possível usar outros algoritmos para que a transmissão de dicionários seja mais eficiente. Foi demonstrado que a utilização de algoritmos que consideram a ordem parcial têm melhores desempenhos que os algoritmos convencionais para sequências de eventos com ordem parcial (ALUR et al., 2003). É preciso investigar mais os algoritmos propostos por Alur (ALUR et al., 2003), ensejando a compressão de sequências com ordem parcial.

6 Conclusões

Foram apresentados conceitos e teoremas relacionados à compressão de sequências quando não há ordem parcial usando AEP e Teoria dos Tipos. Por meio da teoria dos traços, foi possível definir as classes de intercâmbio que é uma generalização das classes de tipo e assim apresentar os conceitos e teoremas da complexidade de intercâmbio e entropia de intercâmbio.

Foram propostos dois novos algoritmos para compressão de sequência de eventos com ordem parcial, um sub-ótimo com tempo polinomial e um ótimo com o tempo exponencial, respectivamente. Os desempenhos dos dois algoritmos foram comparados em termos de tempo e taxa de redução no número de símbolos dos dicionários gerados. Para a compressão de sequências de eventos, o desempenho do Algoritmo 1 é semelhante ao obtido usando o Algoritmo 2, dependendo dos parâmetros selecionados. Quando se considera sub-sequências com tamanho maior que 10, o tempo de execução do Algoritmo 2 se torna impraticável.

Savari (SAVARI, 2004) apresenta desigualdades e um teorema que permite o cálculo exato da entropia de intercâmbio para o caso dos grafos que são k-partidos completos. Entretanto, a entropia de intercâmbio é difícil de ser determinada para grafos mais gerais. Desta forma, os algoritmos também foram utilizados para estimar a entropia de intercâmbio associados a alguns grafos a partir de sequências aleatórias. Pelo conhecimento do autor, essa é uma abordagem nova para estimar a entropia de intercâmbio. Os valores obtidos foram comparados com os valores teóricos dos grafos k-partidos completos, sendo possível obter valores abaixo da entropia de Shannon. Apesar do Algoritmo 1 apresentar a necessidade de um comprimento determinado para as sequências para estimar a entropia de intercâmbio adequadamente, em alguns casos, ainda é uma alternativa dado ao tempo exponencial do Algoritmo 2.

Como trabalhos futuros, os algoritmos apresentados podem ser aprimorados para melhorar o tempo de execução e comparados com outros algoritmos para a compressão de sequência de eventos com ordem parcial. Além disso, pode-se desenvolver uma aplicação para esses algoritmos e comprimir sequências de comandos ou mensagens de protocolos.

Referências

ALUR, R. et al. Compression of partially ordered strings. In: *CONCUR 2003 - Concurrency Theory*. Springer Berlin Heidelberg, 2003. p. 42–56. Disponível em: <https://doi.org/10.1007%2F978-3-540-45187-7_3>. Citado 2 vezes nas páginas 24 e 58.

COVER, T. M.; THOMAS, J. A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006. ISBN 0471241954. Citado 7 vezes nas páginas 24, 27, 29, 30, 31, 35 e 36.

DIEKERT, V. *Combinatorics on Traces*. Berlin, Heidelberg: Springer-Verlag, 1990. ISBN 0387530312. Citado 2 vezes nas páginas 31 e 32.

DIEKERT, V.; ROZENBERG, G. *The Book of Traces*. USA: World Scientific Publishing Co., Inc., 1995. ISBN 9810220588. Citado na página 27.

HORIBE, Y. A note on kolmogorov complexity and entropy. *Applied Mathematics Letters*, v. 16, n. 7, p. 1129 – 1130, 2003. ISSN 0893-9659. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0893965903901054>. Citado 2 vezes nas páginas 27 e 36.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 21, n. 7, p. 558–565, jul. 1978. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/359545.359563>. Citado na página 23.

LI, M.; VITNYI, P. M. *An Introduction to Kolmogorov Complexity and Its Applications*. 3. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. ISBN 0387339981. Citado 2 vezes nas páginas 27 e 35.

MAZURKIEWICZ, A. Concurrent program schemes and their interpretations. *DAIMI Report Series*, v. 6, n. 78, Jul. 1977. Disponível em: <https://tidsskrift.dk/daimipb/article/view/7691>. Citado 5 vezes nas páginas 23, 24, 27, 31 e 32.

NEVILL-MANNING, C. G.; WITTEN, I. H. Identifying hierarchical structure in sequences: A linear-time algorithm. AI Access Foundation, El Segundo, CA, USA, v. 7, n. 1, p. 67–82, set. 1997. ISSN 1076-9757. Citado 2 vezes nas páginas 24 e 58.

PERRIN, D. Words over a partially commutative alphabet. In: *Combinatorial Algorithms on Words. NATO ASI Series (Series F: Computer and Systems Sciences)*. [S.l.]: Springer, Berlin, Heidelberg, 1985. v. 12. Citado 3 vezes nas páginas 27, 33 e 34.

REZNIK, Y. A. Coding of sets of words. In: *2011 Data Compression Conference*. [S.l.: s.n.], 2011. p. 43–52. Citado na página 24.

SAVARI, S. A. On compressing interchange classes of events in a concurrent system. In: *Data Compression Conference, 2003. Proceedings. DCC 2003.* [S.l.: s.n.], 2003. p. 153–162. Citado na página 24.

SAVARI, S. A. Compression of words over a partially commutative alphabet. *IEEE Transactions on Information Theory*, v. 50, n. 7, p. 1425–1441, 2004. Citado 9 vezes nas páginas 24, 25, 27, 34, 36, 37, 38, 39 e 59.