UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

HIGOR ÍTALO DOS SANTOS

MODELOS MARKOVIANOS PARA SISTEMAS DINÂMICOS EMPREGANDO APRENDIZADO DE MÁQUINA E TEORIA DE AUTÔMATOS

Recife

2020

HIGOR ÍTALO DOS SANTOS

MODELOS MARKOVIANOS PARA SISTEMAS DINÂMICOS EMPREGANDO APRENDIZADO DE MÁQUINA E TEORIA DE AUTÔMATOS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Comunicações.

Orientador: Prof. Dr. Daniel Pedro Bezerra Chaves

Coorientador: Prof. Dr. Cecilio José Lins Pimentel

Recife

2020

Catalogação na fonte
Bibliotecário Gabriel Luz, CRB-4 / 2222

S237m Santos, Higor Ítalo dos
Modelos Markovianos para sistemas dinâmicos empregando aprendizado de máquina e teoria de autômatos / Higor Ítalo dos Santos – Recife, 2020.
56 f.: figs., tabs., abrev. e siglas, símbolos.

Orientador: Prof. Dr. Daniel Pedro Bezerra Chaves.
Coorientador: Prof. Dr. Cecilio José Lins Pimentel.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Elétrica, 2020.
Inclui referências.

1. Engenharia Elétrica. 2. Aprendizagem de máquina. 3. Autômatos probabilísticos de estados finitos. 4. Minimização de grafos. 5. Modelagem não supervisionada. 6. Sistemas dinâmicos. I. Chaves, Daniel Pedro Bezerra (Orientador). II. Pimentel, Cecilio José Lins (Coorientador). III. Título.

UFPE

621.3 CDD (22. ed.) BCTG / 2021 - 229

HIGOR ÍTALO DOS SANTOS

“MODELOS MARKOVIANOS PARA SISTEMAS DINÂMICOS EMPREGANDO APRENDIZADO DE MÁQUINA E TEORIA DE AUTÔMATOS”

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica.

Aprovada em: 20 de fevereiro de 2020.

BANCA EXAMINADORA

_____
Profº. Dr. Daniel Pedro Bezerra Chaves (Orientador e Examinador Interno)
Universidade Federal de Pernambuco

_____
Profº. Dr. Cecilio José Lins Pimentel (Coorientador e Examinador Interno)
Universidade Federal de Pernambuco

_____
Profº. Dr. Juliano Bandeira Lima (Examinador Interno)
Universidade Federal de Pernambuco

_____
Profº. Dr. José Sampaio de Lemos Neto (Examinador Externo)
Universidade Federal de Pernambuco

A todos que de alguma forma me ajudaram a seguir em frente.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus pelo suporte espiritual que me instrue e anima a seguir adiante em todos momentos de vida.

À minha amada Gabriela Chagas, maior incentivadora de minha pós-graduação, por todo apoio, compreensão, paciência e amor dedicados ao longo de tantos anos juntos e de duração do mestrado acadêmico.

Aos meus pais, Inácia e Adilson, por sempre terem buscado garantir a melhor educação para mim, mesmo com tantas dificuldades enfrentadas, me permitindo chegar onde estou hoje.

À minha irmã Amanda e demais familiares, pelo suporte material e moral proporcionados para alcançar meus objetivos.

Ao meu orientador Prof. Dr. Daniel Chaves e meu coorientador Prof. Dr. Cecilio Pimentel, por todas as horas dedicadas a esta pesquisa, pela atenção, paciência, disponibilidade e pelos ensinamentos que me proporcionaram a realização deste trabalho.

A Diego Hamilton, sem a ajuda do qual não teria tido êxito no desenvolvimento e execução dos códigos computacionais presentes neste trabalho.

A Helder, Túlio, Antônio, Carlos e tantos outros amigos e colegas de pós-gradução que acompanharam minha trajetória até aqui e de alguma forma contribuíram para meu progresso no mestrado.

A todos os professores que passaram por minha vida e que contribuíram com paciência, dedicação e ensinamentos para a minha formação.

À Universidade Federal de Pernambuco por, mesmo em meio às dificuldades enfrentadas pelo país atualmente, forneceu-me uma formação pós-acadêmica de qualidade.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro recebido durante o tempo de realização deste trabalho.

RESUMO

Uma abordagem difundida para a análise e modelagem de sistemas dinâmicos parte de um processo de discretização seguido por uma quantização do sinal, gerando uma série temporal discreta. Entre as vantagens propiciadas por esse método estão a redução na complexidade computacional no processamento do sinal e maior imunidade ao ruído. A dinâmica discreta da série resultante pode ser representada através de um Autômato Probabilístico de Estados Finitos (PFSA, Probabilistic Finite State Automata), comumente empregado em algoritmos de controle e técnicas de detecção de falha. Este trabalho apresenta um novo algoritmo para determinar um PFSA associado a um sistema dinâmico a partir de uma realização suficientemente longa da dinâmica. Por meio da análise da sequência discreta gerada, empregam-se métodos estatísticos, aprendizado de máquina e minimização de grafos para obter modelos PFSA compactos e fidedignos à dinâmica discreta oriunda do sistema de origem. Por fim, o método proposto é aplicado a sistemas dinâmicos com características distintas, demonstrando-se sua capacidade de gerar modelos com número reduzido de estados sem degradação de sua qualidade.

Palavras-chave: Aprendizagem de máquina. Autômatos probabilísticos de estados finitos. Minimização de grafos. Modelagem não supervisionada. Sistemas dinâmicos.

ABSTRACT

A widespread approach to the analysis and modeling of dynamical systems starts from a discretization process followed by a signal quantization, which generates a discrete time series. Among the advantages provided by this method are the reduction in computational complexity of the signal processing and greater noise immunity. The discrete dynamics of the resulting series can be represented by a Probabilistic Finite State Automata (PFSA), commonly employed in control algorithms and fault detection techniques. This thesis presents a new algorithm for determining a PFSA associated with a dynamical system from a sufficiently long stream of the dynamics. Techniques from statistical methods, machine learning and graph minimization are employed to obtain compact and reliable PFSA models for the discrete dynamics of the source system. Finally, the proposed method is applied to dynamic systems with different characteristics, demonstrating its ability to generate models with reduced number of states without degradation of their quality.

Keywords: Machine learning. Probabilistic finite state automata. Graph minimization. Unsupervised modeling. Dynamical systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PFSA	Probabilistic Finite State Automata
DCGraM	D-Markov with Clustering and Graph Minimization
VL-DCGraM	Variable Lenght D-Markov with Clustering and Graph Minimization
SNR	Signal to Noise Ratio
DFC	Discrete Fading Channel
OPTICS	Ordering Points to Identify the Clustering Structure
DBSCAN	Density-based Spatial Clustering of Applications with Noise

LISTA DE SÍMBOLOS

D	Profundidade de uma máquina D-Markov
S	Sequência simbólica gerada por um PFSA
ϵ	Sequência de comprimento nulo
L	Profundidade de uma árvore de contextos $\mathcal{CT}$
Σ	Alfabeto finito. Conjunto discreto de símbolos
Σ^n	Conjunto de sequências de símbolos de Σ de comprimento n
Σ^*	Conjunto de todas as sequências com símbolos de Σ de qualquer comprimento
σ	Símbolo pertencente ao alfabeto Σ
G	Grafo discreto rotulado
$F(q)$	Contexto à direita de uma estado $q \in Q$ de uma grafo G
Q	Conjunto de estados de uma grafo G
q	Estado pertencente a conjunto de estados Q de um grafo G
δ	Função de transição de um grafo
$\mathcal{P}$	Partição de um conjunto de estados Q
$\mathcal{L}$	Linguagem gerada por um grafo
$\mathcal{M}_h$	Partição definida pela equivalência Moore de profundidade h
π	Função probabilidade de transição de estados de um PFSA, $\pi : Q \times \Sigma \rightarrow [0, 1]$
$\mathbf{P}$	Matriz de transição de probabilidade $Q \times Q$ de um PFSA
$\boldsymbol{\mu}$	Vetor estacionário da matriz de transição de probabilidade $\mathbf{P}$
K	Número de *clusters* do Algoritmo k-means
$\mathbf{x}_n$	Ponto de dado associado a um *cluster*
$\mathbf{m}^{(k)}$	Conjunto de vetores I-dimensionais correspondente aos centroides de um *cluster*

J	Função objetivo que relaciona a distância entre pontos de dado $\mathbf{x}_n$ e um centroide $\mathbf{m}^{(k)}$
(G, π)	PFSA com grafo G e função probabilidade de transição π
$\mathcal{V}(q)$	*Morph* de um estado q de um PFSA (G, π), $\{\pi(q, \sigma), \forall \sigma \in \Sigma\}$
$\mathcal{V}(Q)$	Conjunto dos *morphs* de todos os estados $q \in Q$ de um PFSA (G, π)
$\mathcal{CT}$	Árvore de contextos
h	Taxa de entropia de um sistema dinâmico
h_ℓ	Taxa de entropia de ℓ-ésia ordem de um sistema dinâmico
D_ℓ	Divergência de Kullback-Leibler de ℓ-ésia ordem entre sequências S_1 e S_2
r	Parâmetro de um mapa logístico
X_k	Entrada de um canal binário com desvanecimento no instante k
Y_k	Saída de um canal binário com desvanecimento no instante k
Z_k	Símbolo de ruído de um canal binário com desvanecimento no instante k
R_k	Símbolo recebido no instante k
S_k	Símbolo transmitido no instante k
E_s	Energia do sinal transmitido
N_k	Variável aleatória Gaussiana de média zero com variância $N_0/2$
E_s/N_0	Relação sinal ruído
$\{A_k\}$	Processo de desvanecimento de canal
J_0	Função de Bessel de ordem zero
$f_D T$	Máxima frequência Doppler normalizada
T	*Threshold* de ramificação de uma árvore de contextos $\mathcal{CT}$
α	Valor real positivo adotado no cálculo da constante l_μ do algoritmo VL-DCGraM
M	Comprimento de palavra máximo adotado no algoritmo VL-DCGraM

SUMÁRIO

1 INTRODUÇÃO

Neste capitulo é feita uma introdução do presente trabalho, apresentando na Seção 1.1 as motivações que servem de base para o mesmo. Na Seção 1.2 é abordado o objetivo da pesquisa, com descrição das etapas seguidas para o desenvolvimento da mesma. Na Seção 1.3 é apresentada a estrutura organizacional desta dissertação.

1.1 MOTIVAÇÃO

O aumento contínuo de complexidade e do custo de sistemas de engenharia torna características como confiabilidade e segurança cada vez mais importantes, sendo fatores sujeitos às anormalidades de componentes e falhas de processo. Alguns exemplos desses sistemas são aeronaves, veículos autônomos, redes elétricas, máquinas elétricas, turbinas eólicas, equipamentos eletrônicos industriais, entre outros. Portanto, a adoção de medidas para detectar e identificar possíveis causas de mau funcionamento é essencial para minimizar a degradação de desempenho, a diminuição da produtividade e os riscos à segurança.

Entre as várias abordagens para resolver o problema de detecção de anomalias, existem duas categorias comuns: métodos baseados em modelo e métodos orientados a dados. O método baseado em modelo exige que o modelo do processo ou sistema considerado esteja disponível (GAO; CECATI; DING, 2015), (DING, 2008), que pode ser obtido usando princípios físicos ou técnicas de identificação de sistema. Uma vantagem associada aos métodos baseados em modelo é a interpretação física, no entanto a confiabilidade e a eficiência computacional tendem a diminuir à medida que a complexidade do sistema aumenta. Estes são usados em várias aplicações, incluindo: sistema de gerenciamento de bateria incorporado em tempo real (KIM et al., 2018); estruturas para diagnóstico e prognóstico de máquinas elétricas (NGUYEN et al., 2017); projeto de sensor ultrassônico (JIANG et al., 2018); esquema de detecção e isolamento de falhas em tempo real para aeronaves com motores a jato (MESKIN; NADERI; KHORASANI, 2011).

O método orientado a dados (CECATI, 2015) lida com o problema de representar os dados em um espaço com menor dimensão ou mais conveniente e deve permanecer confiável e computacionalmente eficiente para rastrear a relação entrada-saída de sinais obtidos de sensores confiáveis. Esse método possui um amplo escopo de aplicações, sendo caracterizado pela tarefa de aprender propriedades do mecanismo que gera os dados. Para esse fim, técnicas baseadas em análises estatística são comumente aplicadas, como análise de componentes principais (PCA) (DUNTEMAN, 1989), mínimos quadrados parciais (PLS) (BARKER; RAYENS, 2003), análise de componentes independentes (ACI) (HYVÄRINEN; OJA, 2000), classificadores de padrões estatísticos e máquina de vetores de suporte (SVM) (LORENA; CARVALHO, 2007). O desafio consiste no fato de que tudo isso exige uma grande quantidade de dados de treinamento para capturar as principais características do processo. Essa afirmação é enfatizada ao lidar com

séries temporais, pois o alto volume de dados implica numa complexidade computacional correspondente. No entanto, a análise de séries temporais é uma valiosa técnica de análise de dados, particularmente para sistemas inerentemente não lineares, quando é referida como análise de séries temporais não lineares (NTSA) (BRADLEY; KANTZ, 2015). As aplicações de métodos orientados a dados incluem: garras robóticas (BOHG et al., 2013); previsão dinâmica de destino para sistema de compartilhamento de carros sob demanda com base em sinal GPS (WANG; ZHONG; MA, 2018); estratégia não supervisionada de diagnóstico de falhas baseada em agrupamento espectral (LIANG et al., 2019); método de formação de feixe quase ideal para sistemas MIMO massivos (LONG et al., 2018).

A análise de séries temporais pode ser uma tarefa desafiadora. Detalhes experimentais como taxa de amostragem, resposta dinâmica dos instrumentos e relação sinal ruído podem afetar notavelmente a confiabilidade dos resultados. Um meio de minimizar o efeito do ruído é a adoção de técnicas de simbolização, que pode ser utilizado para melhorar a relação sinal ruído (GRABEN, 2001). A principal característica dessa técnica é a discretização das amostras que compõem a série temporal em uma sequência correspondente de símbolos definidos sobre um conjunto finito, um processo adaptado para reter as informações temporais importantes sobre a sequência simbólica (DAW; FINNEY; TRACY, 2003), (LI et al., 2017). Ganhos adicionais podem ser alcançados comprimindo adequadamente os conjuntos de dados de alta dimensão em estruturas de baixa dimensão, o que é fornecido por uma técnica chamada Filtragem por Dinâmica Simbólica (SDF), que demonstrou ser útil para a extração de informação da série temporal (JIN et al., 2010), (MUKHERJEE; RAY, 2014) e para capturar o comportamento dinâmico subjacente (LI et al., 2017).

A análise por dinâmica simbólica é caracterizada pela conversão de uma série temporal em uma sequência de símbolos (LIND; MARCUS, 1995), processo que se inicia pelo particionamento do intervalo de valores das amostras da série temporal em segmentos, atribuindo cada amostra a um segmento único. Cada segmento é atribuído a um símbolo distinto, mapeando cada amostra de um segmento em um mesmo símbolo. A sequência simbólica obtida é inspecionada em uma escala de tempo rápida, na qual supõe-se que o processo tenha dinâmica estatisticamente estacionária, e em uma escala de tempo lenta, na qual o processo pode exibir dinâmica não estacionária, e a partir disso são identificadas sequências finitas relevantes que possibilitem inferir estatísticas de sequências subsequentes (LI; RAY, 2017). Essas sequências são os blocos de construção para gerar modelos dinâmicos a partir da sequência simbólica (GRABEN, 2001), e esses modelos são usados por algoritmos para detecção de eventos (MUKHERJEE; RAY, 2014). As características desejadas de um modelo dinâmico são sua simplicidade e precisão analíticas para refletir a estatística da sequência simbólica; portanto, é uma ferramenta computacionalmente atraente para representar o comportamento de um sistema e é aplicada à simulação (BANDYOPADHYAY; BHATTACHARYA, 2014), análise de desempenho (PIMENTEL; ALAJAJI, 2009) e detecção de falhas (RAY, 2004).

Autômatos probabilísticos de estado finito (PFSA) fornecem um modelo eficiente para aprender a estrutura causal de um comportamento dinâmico observado (VIDAL et al., 2005a), (VIDAL et al., 2005b). Esses modelos são simples de construir e permitem uma implementação computacional eficiente. Um algoritmo útil para construir um PFSA é a máquina D-Markov (RAY, 2004), que é um processo Markoviano de ordem finita D, o que significa que usa as estatísticas de todas as sequências de comprimento D para formar seus estados. À medida que D aumenta, a máquina D-Markov é capaz de reproduzir o comportamento original do sistema mais precisamente à custa de aumentar exponencialmente o número de estados. Essas máquinas foram usadas para modelar o processo de ruído em sistemas de comunicação sem fio (PIMENTEL; ALAJAJI, 2009), (LIN et al., 2015), (ALTINEL; KURT, 2017), e detecção de anomalias (MUKHERJEE; RAY, 2014), (RAY, 2004), (REN; YE; LI, 2017).

1.2 OBJETIVOS

Neste trabalho, desenvolvemos um algoritmo para construir um PFSA a partir das estatísticas de uma sequência simbólica observada S. É chamado D-Markov de Comprimento Variável com Clusterização e Minimização de Gráficos (VL-DCGraM) e utiliza um método de agrupamento de dados associado a uma técnica de aprendizado de máquina, o algoritmo *k-means* (MACKAY, 2003) e técnicas de minimização de grafos (BERSTEL et al., 2010) da teoria de autômatos (como Moore) para obter modelos PFSA compactos e precisos. Funciona seguindo os passos:

- Construção de uma máquina D-Markov de comprimento variável.
- Formação de agrupamentos (*clusters*) de estados com probabilidade de transição (*morph*) semelhante, criando uma partição inicial $\mathcal{P}$ dos estados.
- Aplicação de algoritmo de minimização de grafo para refinar a partição inicial por divisão de estados, a fim de melhorar a precisão do modelo resultante.

O algoritmo de agrupamento, ou clusterização, é uma maneira eficiente de mesclar estados com estatísticas de primeira ordem semelhantes em classes de equivalência. No entanto, as estatísticas de ordem superior para estados da mesma classe podem ser diferentes. O algoritmo de minimização de grafo identifica essas diferenças e divide a classe correspondente em subclasses nas quais os estados preservam as estatísticas de ordem superior. Apresentamos um exemplo de modelagem para ilustrar que o VL-DCGraM produz modelos PFSA com menor número de estados do que as máquinas D-Markov e DCGraM (FRANCH, 2017) originais com precisão semelhante. Por meio da variação do comprimento dos estados das máquinas geradas e do número de *clusters*, demonstra-se que há um compromisso entre o número de estados do modelo e sua precisão.

1.3 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado em cinco partes. No Capítulo 2 analisa-se os fundamentos teóricos, incluindo PFSA, algoritmos de agrupamento e minimização de gráficos. No Capítulo 3 apresenta-se o algoritmo VL-DCGraM, detalhando-se cada uma de suas etapas. Algumas aplicações são apresentadas no Capítulo 4 para demonstrar a capacidade do VL-DCGraM de reproduzir a estatística da sequência original, além do seu desempenho em comparação ao modelo D-Markov e ao modelo gerado pelo algoritmo DCGraM. Finalmente, as considerações finais são apresentadas no Capítulo 5.

2 PRELIMINARES SOBRE GRAFOS, AUTÔMATO PROBABILÍSTICO DE ESTADOS FINITOS E CLUSTERIZAÇÃO

Neste capítulo, revisamos conceitos de grafos e PFSA (LIND; MARCUS, 1995) (VIDAL et al., 2005c) que são necessários nos capítulos subsequentes. O conceito de minimização de grafos é apresentado e um algoritmo convencional para essa finalidade, Moore, é descrito. O algoritmo de clusterização *k-means* e algumas variações são apresentadas, constituindo uma etapa essencial na proposta sugerida. Finalmente, dois algoritmos já conhecidos para sistemas de modelos dinâmicos com PFSA são apresentados, a saber, D-Markov e DCGraM (FRANCH, 2017).

2.1 SEQUÊNCIAS DE SÍMBOLOS DISCRETOS

Nesta seção introduzimos conceitos preliminares de linguagens formais (HOPCROFT, 2008). Uma sequência finita u de símbolos de um alfabeto Σ é denominada uma palavra e seu comprimento é denotado por $|u|$. A palavra vazia ε é definida como a sequência com comprimento igual a 0. O conjunto de todas as possíveis palavras de comprimento n, com símbolos em Σ é Σ^n; e o conjunto de todas as sequências com símbolos em Σ para qualquer comprimento n, incluindo a sequência vazia ε, é

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n.$$

Duas palavras u e $v \in \Sigma^*$ podem ser concatenadas para formar uma sequência uv. Por exemplo, utilizando uma alfabeto binário, $\Sigma = \{0, 1\}$, a concatenação de $u = 1100$ e $v = 100$ é $uv = 1100100$. Note-se que $|uv| = |u| + |v|$. Em Σ^*, a concatenação satisfaz o fechamento e a associatividade, o que significa que $u(vw) = (uv)w = uvw$, mas não é comutativa, uma vez que uv não é necessariamente igual a vu. A palavra vazia ε é um elemento neutro para a concatenação, isto é, $\varepsilon u = u\varepsilon = u$. Isso significa que Σ^* com a operação de concatenação é um monoide, uma vez que este constitui um conjunto com uma operação associativa e que possui um elemento identidade (LALLEMENT, 1979).

A palavra $v \in \Sigma^*$ é denominada sufixo da palavra $w \in \Sigma^*$ ($|w| > |v|$) se w pode ser escrita pela concatenação uv, para algum $u \in \Sigma^*$. Nesse mesmo sentido, a sequência u é chamada um prefixo de w.

2.2 GRAFOS

No decorrer desta dissertação, grafo é utilizado para denotar um gráfico direcionado rotulado conforme a Definição 2.1.

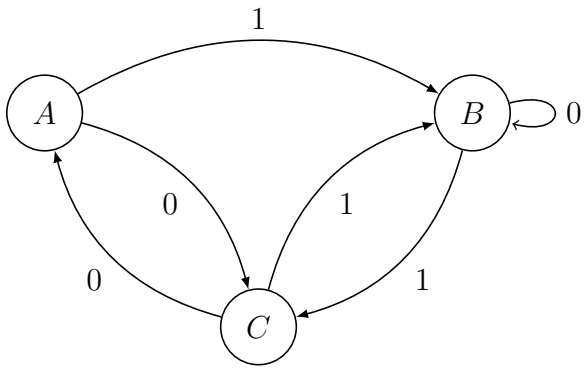
Definição 2.1 (**Grafo**). Um grafo *G* sobre o alfabeto Σ consiste de uma tupla (Q, Σ, δ):

- Q é um conjunto finito de estados com cardinalidade $|Q|$;
- Σ é um alfabeto finito com cardinalidade $|\Sigma|$;
- δ é a função de transição de estado $Q \times \Sigma \to Q$.

Cada estado $q \in Q$ pode ser representado como um ponto ou círculo e se $\exists\, \delta(q, \sigma) = q'$, para $q, q' \in Q$ e $\sigma \in \Sigma$, esta transição pode ser representada com uma seta direcionada do estado q para o estado q' com rótulo σ. A Figura 1 mostra um exemplo de um grafo com três estados sobre um alfabeto binário de onde é possível ver que há uma transição partindo do estado *A* em direção ao estado *B* com o símbolo 1, portanto $\delta(A, 1) = B$.

É possível estender a função de transição a fim de que ela aceite palavras e não apenas símbolos. Dado $\omega \in \Sigma^n$, em que $\omega = \sigma_1\sigma_2 \ldots \sigma_n$ com $\sigma_m \in \Sigma$, para $m = 1 \ldots n$, e dados os estados $q_0, q_1, \ldots, q_n \in Q$, definimos a função $\delta^*(q_0, \omega) = q_n$ se $\delta(q_0, \sigma_1) = q_1, \delta(q_1, \sigma_2) = q_2, \ldots, \delta(q_{n-1}, \sigma_n) = q_n$. Se para dois estados $q_1, q_2 \in Q$ há uma palavra $\omega \in \Sigma^*$ tal que $\delta^*(q_1, \omega) = q_2$, é dito existir um caminho entre q_1 e q_2 e que ω é gerada por *G*. Na Figura 1 o caminho iniciando no estado *A* e seguindo por *B*, *B*, *C*, *B*, *C*, *A* gera a palavra $\omega = 101110$, que é $\delta^*(A, 101110) = A$.

Figura 1 – Um grafo de três estados com *Q* = {A, B, C} e $\Sigma = \{0, 1\}$.



Fonte: O autor (2019)

Definição 2.2 (**Contexto à Direita**)**.** O contexto à direita de um estado $q \in Q$ é definido como o conjunto de todas as possíveis palavras geradas pelos caminhos que iniciam em q e findam em um estado de Q:

$$F(q) = \{\omega \in \Sigma^* | \delta^*(q, \omega) \in Q\}.$$

Definição 2.3 (**Linguagem de um grafo**)**.** A linguagem $\mathcal{L} \subset \Sigma^*$ de um grafo *G* é o conjunto formado pela união de contextos à direta de cada estado $q \in Q$:

$$\mathcal{L} = \bigcup_{q \in Q} F(q).$$

2.3 MINIMIZAÇÃO DE GRAFOS

Nesta seção, o tópico de minimização de grafos é discutido e o algoritmo Moore é mostrado como um exemplo de um algoritmo de minimização de grafos determinísticos. Como uma extensão dessa aplicação dos algoritmos de minimização, na dissertação (FRANCH, 2017), essa classe de algoritmos é empregada para identificar palavras com contextos à direita estatísticamente dissimilares.

2.3.1 Noções Preliminares

Para uma linguagem $\mathcal{L}$, é possível haver dois grafos $G_1 = (Q_1, \Sigma, \delta_1)$ e $G_2 = (Q_2, \Sigma, \delta_2)$ com $|Q_1| \neq |Q_2|$ capazes de gera-la. Isto decorre do fato que uma dada linguagem $\mathcal{L}$ pode ser representada por inúmeros grafos distintos. Entre elas, é desejável adotar aquela com o menor número de estados, pois isso promove a redução da complexidade computacional dos algoritmos que empregam essas representações e da memória requerida para o armazenamento dessas estruturas.

Definição 2.4 (Grafo Mínimo). Para uma dada linguagem $\mathcal{L}$ existe um grafo mínimo $G_{min} = (Q, \Sigma, \delta)$ capaz de gerá-la. O grafo mínimo é aquele para o qual, dados $q_1, q_2 \in Q$, $q_1 \neq q_2$, então $F(q_1) \neq F(q_2)$.

Se um grafo $G_1 = (Q_1, \Sigma, \delta_1)$ tem dois estados distintos q_1 e q_2 com o mesmo contexto à direita, um novo grafo $G_2 = (Q_2, \Sigma, \delta_2)$, que gera a mesma linguagem, pode ser obtido pela fusão desses estados, i.e., G_2 substitui os estados q_1 e q_2 por um estado q', tal que, se algum estado s de G_1 possui uma transição com rótulo $\sigma \in \Sigma$ para q_1 ou q_2, o estado s de G_2 tem transição com rótulo σ para q'. As linguagens geradas por G_1 e G_2 são as mesmas (HOPCROFT, 2008).

O processo de minimização de grafos está relacionado ao particionamento do conjunto de estados Q de um grafo empregando a relação de equivalência de Nerode (BERSTEL et al., 2010), descrita pela relação

$$p, q \in Q, p \equiv q \Leftrightarrow F(p) = F(q),$$

e posterior construção do grafo mínimo, empregando como estados desse representantes das partições obtidas.

Definição 2.5 (Partições e Relações de Equivalência). Dado um conjunto E, uma partição de E é uma família $\mathcal{P}$ de subconjuntos não vazios e disjuntos P de E tal que $\bigcup_{P \in \mathcal{P}} P = E$. O índice da partição corresponde ao seu número de elementos. A partição $\mathcal{P}$ define uma relação de equivalência sobre E e o conjunto de todas as classes de equivalência de uma relação de equivalência em E define uma partição do conjunto (HOPCROFT, 2008).

Um grafo é considerado mínimo se, e somente se, a partição gerada com a equivalência de Nerode é a identidade.

Seguem alguns conceitos necessários para apresentar o algoritmo de minimização. Dado um grafo G, dois algoritmos para obter um grafo mínimo a partir dele são: Moore e Hopcroft (BERSTEL et al., 2010). O primeiro é descrito nesta seção, mas algumas definições são necessárias antes de adentrar no algoritmo.

Quando um subconjunto F de E é a união de classes de $\mathcal{P}$, diz-se que F é saturado por $\mathcal{P}$. Dado $\mathcal{Q}$, outra partição de E, esta é dita ser um refinamento de $\mathcal{P}$ (ou que $\mathcal{P}$ é menos refinado que $\mathcal{Q}$) se cada classe de $\mathcal{Q}$ está contida em alguma classe de $\mathcal{P}$, o que é representado por $\mathcal{Q} \leq \mathcal{P}$. O índice de $\mathcal{Q}$ é maior ou igual ao índice de $\mathcal{P}$.

Dadas partições $\mathcal{P}$ e $\mathcal{Q}$ de $E, \mathcal{U} = \mathcal{P} \wedge \mathcal{Q}$ denota a partição menos refinada que refina $\mathcal{P}$ e $\mathcal{Q}$. Os elementos de $\mathcal{U}$ são conjuntos não vazios $P \cap Q$, em que $P \in \mathcal{P}$ e $Q \in \mathcal{Q}$. A notação é estendida para múltiplos conjuntos como $\mathcal{U} = \mathcal{P}_1 \wedge \mathcal{P}_2 \wedge \ldots \wedge \mathcal{P}_n = \bigwedge_{j=1}^{n} \mathcal{P}_j$.

Dado $F \subseteq E$, uma partição $\mathcal{P}$ de E induz uma partição $\mathcal{P}'$ de F por interseção. $\mathcal{P}'$ é composta pelos conjuntos $P \cap F$ com $P \subseteq \mathcal{P}$. Se $\mathcal{P}$ e $\mathcal{Q}$ são partições de E e $\mathcal{Q} \leq \mathcal{P}$, as restrições $\mathcal{P}'$ e $\mathcal{Q}'$ para F satisfazem $\mathcal{Q}' \leq \mathcal{P}'$.

Dado um conjunto de estados $P \subset Q$ e um símbolo $\sigma \in \Sigma$, denota-se $\sigma^{-1}P$ o conjunto de estados $q \in Q$ tal que $\delta(q, \sigma) \in P$. Considerando $P, R \subset Q$ e $\sigma \in \Sigma$, a partição de R

$$(P, \sigma)|R$$

é composta por dois subconjuntos possivelmente não vazios:

$$R \cap \sigma^{-1}P = \{r \in R \,\big|\, \delta(r, \sigma) \in P\} \tag{2.1}$$

e

$$R \backslash \sigma^{-1}P = \{r \in R \,\big|\, \delta(r, \sigma) \notin P\}. \tag{2.2}$$

Considere a extensão da função de transição sobre $P(Q) \times \Sigma$, em que $P(Q)$ é o conjunto de partes de Q, tal que $\delta(A, \sigma) = B = \{q \in Q \,\big|\, \exists p \in A,\, \delta(p, \sigma) = q\}$. O par (P, σ) é chamado divisor (*splitter*). Observe que $(P, \sigma)\big|R = R$ se $\delta(R, \sigma) \subset P$ ou $\delta(R, \sigma) \cap P = \emptyset$; e $(P, \sigma)|R$ é composto essencialmente de duas classes não vazias se ambos $\delta(R, \sigma) \cap P \neq \emptyset$ e $\delta(R, \sigma) \cap P^c \neq \emptyset$, ou equivalentemente, $\delta(R, \sigma) \not\subset P$ e $\delta(R, \sigma) \not\subset P^c$. Se $(P, \sigma)|R$ contém duas classes, então dizemos que (P, σ) divide R. Essa notação também pode ser estendida para sequências, utilizando $\omega \in \Sigma^*$ aos invés de $\sigma \in \Sigma$.

Proposição 2.1. *A partição correspondente a equivalência de Nerode é a partição menos refinada $\mathcal{P}$ tal que nenhum divisor (P, σ), com $P \in \mathcal{P}$ e $\sigma \in \Sigma$, divide uma classe em $\mathcal{P}$, ou seja, $(P, \sigma)\big|R = R$ para todo $P, R \in \mathcal{P}$ e $\sigma \in \Sigma$.*

2.3.2 Algoritmo Moore

Um importante algoritmo de minimização é o algoritmo Moore (MOORE, 1956). Este é baseado na ideia de tomar uma partição inicial com um critério bastante amplo e refiná-la até obter as classes de equivalência associadas à relação de equivalência de Nerode. O esboço do algoritmo é mostrado no Algoritmo 1, o qual encontra o grafo mínimo que gera, dada a partição inicial $\mathcal{P}$, a linguagem gerada pelo grafo G. A partição inicial usual para Moore é gerada agrupando estados que possuem transições com os mesmos rótulos.

Algoritmo 1 Moore($G, \mathcal{P}$)

1: **repita**
2: $\quad \mathcal{P}' \leftarrow \mathcal{P}$
3: $\quad$ **para todo** $\sigma \in \Sigma$ **faça**
4: $\quad\quad \mathcal{P}_\sigma \leftarrow \bigwedge_{P \in \mathcal{P}} (P, \sigma)|Q$
5: $\quad \mathcal{P} \leftarrow \mathcal{P} \wedge \bigwedge_{\sigma \in \Sigma} \mathcal{P}_\sigma$
6: **até que** $\mathcal{P} = \mathcal{P}'$

Dado um grafo $G = (Q, \Sigma, \delta)$ e uma partição inicial $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, o conjunto $L_q^{(h)}$ é definido como:

$$L_q^{(h)}(G) = \left\{ w \in \Sigma^* \;\middle|\; |w| \leq h, \delta^*(q, w) \in \mathcal{P}_j \right\},$$

em que $\mathcal{P}_j \in \mathcal{P}$ é composta por todas as palavras até comprimento h que podem ser geradas a partir de um certo $q \in Q$ e atingindo um estado na classe de equivalência de $\mathcal{P}_j$. A equivalência Moore de ordem h (denotada por $\equiv_h$) é definida como:

$$p \equiv_h q \Leftrightarrow L_p^{(h)}(G) = L_q^{(h)}(G).$$

Essa relação de equivalência afirma que dois estados são equivalentes se gerarem as mesmas palavras de comprimento até h que atingem um estado em $\mathcal{P}_j$. Quando a partição gerada pela relação de equivalência Moore $\equiv_h$ coincide com a obtida pela aplicação da relação de equivalência de Nerode $\equiv$ sobre $\mathcal{P}_j$, obtêm-se a profundidade do algoritmo Moore, sendo o menor número inteiro h tal que $\equiv_h$ é igual a $\equiv_{h+1}$, o que leva a um algoritmo que aplica sucessivamente a relação de equivalência Moore, até que a partição anterior não seja mais refinada.

Proposição 2.2. *Para dois estados $p, q \in Q$ e $h \geq 0$,*

$$p \equiv_{h+1} q \iff p \equiv_h q \text{ e } \delta(p, \sigma) \equiv_h \delta(q, \sigma),\ \forall \sigma \in \Sigma. \tag{2.3}$$

Utilizando-se esta formulação e definindo $\mathcal{M}_h$ como a partição definida pela equivalência Moore de profundidade h, a seguinte proposição é válida:

Proposição 2.3 (b). *Para $h \geq 0$,*

$$\mathcal{M}_{h+1} = \mathcal{M}_h \wedge \bigwedge_{\sigma \in \Sigma} \bigwedge_{P \in \mathcal{M}_h} (P, \sigma)|Q. \tag{2.4}$$

Esse cálculo significa que, para cada símbolo $\sigma \in \Sigma$ e para cada classe de equivalência $P \in \mathcal{M}_h$ (em que $\mathcal{M}_h$ é a partição da iteração anterior), um divisor (P, σ) é criado e aplicado ao conjunto original de estados Q. Isso criará partições que mostram quais estados de Q levam a estados em P com símbolo σ e quais não. Desse modo, a partição mais refinada $\bigwedge_{P \in \mathcal{M}_h}(P, \sigma)|Q$ é obtida, já que quebrará as classes de $\mathcal{M}_h$ em conjuntos de estados que levam às mesmas classes com ramos de rótulos σ. Feito isso, a partição mais refinada $\bigwedge_{\sigma \in \Sigma} \bigwedge_{P \in \mathcal{M}_h}(P, \sigma)|Q$ entre essas classes de equivalência são obtidas, separando Q em classes que levam à mesma classe em $\mathcal{M}_h$ com cada símbolo distinto de Σ. Isso calcula efetivamente a parte $\delta(p, \sigma) \equiv_h \delta(q, \sigma),\ \forall \sigma \in \Sigma$ de (2.3). Para finalizar (2.4), a partição mais refinada dessa última etapa e de $\mathcal{M}_h$ é obtida, resultando no incremento nas classes de equivalência de Moore.

Esse cálculo anterior é feito no Algoritmo 1 no qual a iteração refina a partição atual até que não ocorram alterações entre $\mathcal{M}_h$ e $\mathcal{M}_{h+1}$, o que significa que a equivalência Nerode é alcançada. Para construção de um grafo, a partição inicial é criada pelo agrupamento de estados em Q que possuem transições com mesmo rótulo.

O algoritmo Moore para o refinamento de k partições de um conjunto com n elementos pode ser realizado temporalmente em $O(kn^2)$. Cada iteração é processada em $O(kn)$, logo o tempo total é $O(mkn)$, em que m é o número total de etapas de refinamento necessárias para calcular a equivalência de Nerode.

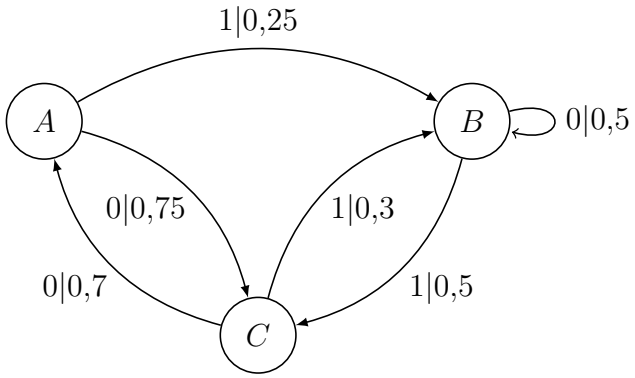
2.4 AUTÔMATO PROBABILÍSTICO DE ESTADOS FINITOS

Definição 2.6 (**Autômato Probabilístico de Estados Finitos**). Um autômato probabilístico de estados finitos (PFSA, *probabilistic finite state automata*) é definido como um grafo G e uma função π definida sobre seus ramos, representado pelo par (G, π). A Função $\pi : Q \times \Sigma \to [0, 1]$ é tal que para qualquer $q \in Q$, $\sum_{\sigma \in \Sigma} \pi(q, \sigma) = 1$, define uma distribuição de probabilidade associada com cada estado de G.

Definição 2.7 (**Morph**). Dado um estado $q \in Q$, a distribuição de probabilidade $\mathcal{V}(q) = \{\pi(q, \sigma); \forall \sigma \in \Sigma\}$ associada com q é chamada seu *morph*.

Um PFSA é desenhado com seu grafo com cada transição de estado rotulada com um símbolo σ e a probabilidade $\pi(q, \sigma)$. Um exemplo de PFSA é mostrado na Figura 2. Para cada Q = $\{A, B, C\}$, $\Sigma = \{0, 1\}$. Trata-se do mesmo grafo da Figura 1 com probabilidades associadas a suas transições para criar o PFSA.

Figura 2 – Um PFSA com mesmo grafo da Figura 1.



Fonte: O autor (2019)

Dado um PFSA (G, π), existe uma probabilidade associada a cada palavra $\omega \in \Sigma^*$ que pode ser gerada a partir de cada estado de G. Na Figura 2, partindo do estado A, é possível gerar a palavra $\omega = 10110010$ (uma vez que $\delta^*(A, \omega) = B$) ao se escolher o caminho através dos estados B, B, C, B, B, B, C e A e concatenando os rótulos do caminho para cada uma das transições entre estados. Multiplicando as probabilidades em cada transição de saída, vê-se que $\Pr(\omega|A) = 0,25 \times 0,5 \times 0,5 \times 0,3 \times 0,5 \times 0,5 \times 0,5 \times 0,7 = 0,001640625$.

Seja **P** a matriz de transição de probabilidades $Q \times Q$ de um PFSA, na qual sua entrada (q_1, q_2) é a probabilidade de transição do estado q_1 para o estado q_2, para $q_1, q_2 \in Q$. O vetor estacionário de **P**, denotado por $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_{|Q|}]$ é o seu autovalor correspondente ao autovalor 1. Este vetor satisfaz a relação $\boldsymbol{\mu}$**P** = $\boldsymbol{\mu}$. A matriz de transição de probabilidades e o vetor estacionário do PFSA da Figura 2 são dados por

$$\mathbf{P} = \begin{bmatrix} 0 & 0{,}25 & 0{,}75 \\ 0 & 0{,}5 & 0{,}5 \\ 0{,}7 & 0{,}3 & 0 \end{bmatrix}$$

$$\boldsymbol{\mu} = \begin{bmatrix} 0{,}2641 & 0{,}3585 & 0{,}3773 \end{bmatrix}.$$

Define-se duas matrizes $\mathbf{P}(0)$ e $\mathbf{P}(1)$, $\mathbf{P}(0) + \mathbf{P}(1) = \mathbf{P}$, tal que o elemento (i, j) da matriz $\mathbf{P}(\sigma), \sigma \in \{0, 1\}$ é a probabilidade condicional do PFSA transitar do estado i para o estado j e emitir o símbolo σ. Para o PFSA da Figura 2, as matrizes $\mathbf{P}(0)$ e $\mathbf{P}(1)$ são:

$$\mathbf{P}(0) = \begin{bmatrix} 0 & 0 & 0{,}75 \\ 0 & 0{,}5 & 0 \\ 0{,}7 & 0 & 0 \end{bmatrix}$$

$$\mathbf{P}(1) = \begin{bmatrix} 0 & 0{,}25 & 0 \\ 0 & 0 & 0{,}5 \\ 0 & 0{,}3 & 0 \end{bmatrix}.$$

2.5 CLUSTERIZAÇÃO

O algoritmo apresentado nesta seção, uma técnica de aprendizado de máquina chamada clusterização, é utilizado para gerar a partição inicial que é empregada no algoritmo Moore.

Clusterizar refere-se a uma série de técnicas utilizadas que tomam um conjunto de M objetos e reúne-os em K grupos, chamados *clusters*, que compartilham algum tipo de similaridade (MACKAY, 2003). Existem várias motivações e aplicações para clusterização; neste trabalho estamos interessados em aplicá-lo para agrupar estados que compartilham semelhança entre seus *morphs*.

2.5.1 K-Means

O Algoritmo *k-means*, também chamado Algoritmo do Lloyd (KANUNGO et al., 2002), tem como objetivo particionar um conjunto num número K de *clusters*, sendo K um valor fornecido. Intuitivamente, podemos pensar em um *cluster* como um grupo de pontos de dados cujas distâncias entre tais pontos são pequenas em comparação com as distâncias para pontos fora do *cluster*. Isso pode ser formalizado ao introduzir inicialmente um conjunto de vetores I-dimensionais $\mathbf{m}^{(k)}$, em que $k = 1, \ldots, K$, em que $\mathbf{m}^{(k)}$ é um protótipo associado ao k-ésimo grupo. Alternativamente, os pontos $\mathbf{m}^{(k)}$ podem ser interpretados como representando os centros dos agrupamentos, denominados centroides.

Deseja-se encontrar uma atribuição de pontos de dados para agrupamentos, bem como um conjunto de vetores $\mathbf{m}^{(k)}$, de modo que a soma dos quadrados das distâncias de cada ponto para o vetor mais próximo $\mathbf{m}^{(k)}$ seja mínima.

Para cada ponto de dados $\mathbf{x}_n$, associa-se uma variável de indicadores binários $r_{nk} \in 0, 1$, em que $k = 1, \ldots, K$ determinando a qual dos K grupos o ponto de dados $\mathbf{x}_n$ está associado, de modo que, se $\mathbf{x}_n$ estiver associado ao agrupamento k, então $r_{nk} = 1$ e $r_{nj} = 0$ para $j \neq k$. Então, pode-se definir uma função objetiva, às vezes chamada de medida de distorção, dada por

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \mathbf{m}^{(k)} \parallel^2 \tag{2.5}$$

que representa a soma dos quadrados das distâncias de cada ponto de dados para seu vetor atribuído $\mathbf{m}^{(k)}$. O objetivo é encontrar valores para r_{nk} e $\mathbf{m}^{(k)}$ que minimizem J. Isso pode ser realizado através de um procedimento iterativo no qual cada iteração envolve duas etapas sucessivas correspondentes a otimizações sucessivas em relação a r_{nk} e a $\mathbf{m}^{(k)}$. Primeiro, alguns valores iniciais para o $\mathbf{m}^{(k)}$ são escolhidos. Então, na primeira fase, minimiza-se J em relação a r_{nk}, mantendo o $\mathbf{m}^{(k)}$ fixo. Na segunda fase, minimiza-se J em relação a $\mathbf{m}^{(k)}$, mantendo r_{nk} fixo. Essa otimização em dois estágios é repetida até a convergência.

Considere primeiro a determinação dos r_{nk}. Como J em (2.5) é uma função linear de r_{nk}, essa otimização pode ser realizada para fornecer uma solução de forma fechada. Os

termos que envolvem n diferentes são independentes e, portanto, pode-se otimizar para cada n separadamente, escolhendo r_{nk} como 1 para qualquer valor de k que forneça o valor mínimo de $\| \mathbf{x}_n - \mathbf{m}^{(k)} \|^2$. Em outras palavras, simplesmente atribuímos o enésimo ponto de dados ao centro do agrupamento mais próximo. Isso pode ser expresso como

$$r_{nk} = \begin{cases} 1, & \text{se } k = \arg\min_j \| \mathbf{x}_n - \mathbf{m}^{(j)} \|^2 \\ 0, & \text{caso contrário.} \end{cases} \tag{2.6}$$

Segue a otimização do $\mathbf{m}^{(k)}$ com o r_{nk} mantido fixo. A função objetiva J é uma função quadrática de $\mathbf{m}^{(k)}$ e pode ser minimizada igualando a zero sua derivada em relação a $\mathbf{m}^{(k)}$

$$2\sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \mathbf{m}^{(k)}) = 0,$$

do que segue

$$\mathbf{m}^{(k)} = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}.$$

O denominador nesta expressão é igual ao número de pontos atribuídos ao *cluster* k e, portanto, esse resultado tem como interpretação que $\mathbf{m}^{(k)}$ é igual à média de todos os pontos $\mathbf{x}_n$ atribuídos ao *cluster* k.

As duas fases de redesignar pontos de dados para *clusters* e recalcular os centroides são repetidas sucessivamente até que não haja mais alterações nas atribuições (ou até que um número máximo de iterações seja excedido). Como cada fase reduz o valor de J, a convergência do algoritmo é garantida.

O algoritmo completo é mostrado no Algoritmo 2, seguido por uma explicação de cada etapa.

Etapa 1: Inicialização

O método de inicialização mais comum para as médias $\mathbf{m}^{(k)}$ é atribuir valores aleatórios de $\mathbb{R}^I$ a cada média. Existem outras heurísticas que podem ser utilizadas (HAMERLY; ELKAN, 2002), porém a inicialização aleatória geralmente é suficiente para a maioria das aplicações.

Etapa 2: Atribuição

Para cada $\mathbf{x}_n$ com $n = 1, \ldots, M$ e $k = 1, \ldots, K$, o valor de r_{nk} é calculado de acordo com (2.6) para ser utilizado na etapa de atualização.

Etapa 3: Atualização

Esta etapa atualiza os valores das médias, assumindo a posição média dos pontos contidos em cada *cluster*. Para um dado grupo k, $\mathbf{m}^{(k)}$ é atualizado com $\sum_n r_{nk}\mathbf{x}_n / R^{(k)}$, no qual

Algoritmo 2 K-Means

Entradas: $\{\mathbf{x}^{(n)}\}, K$
Saídas: K clusters contendo todos os pontos de dados $\{\mathbf{x}^{(n)}\}$
Etapa 1: Inicialização:
para $k \in 1, \ldots, K$ **faça**
    $\mathbf{m}^{(k)} \leftarrow$ *random*()
Etapa 2: Atribuição
para $n = 1, \ldots, M$ **faça**
    para $k = 1, \ldots, K$ **faça**
        se $k = \arg\min_j \parallel \mathbf{x}_n - \mathbf{m}^{(j)} \parallel^2$ **então**
            $r_{nk} \leftarrow 1$
        senão
            $r_{nk} \leftarrow 0$
Etapa 3: Atualização
para $k = 1, \ldots, K$ **faça**
    $R^{(k)} \leftarrow \sum_n^n r_{nk}$
    $\mathbf{m}^{(k)} = \frac{\sum_n r_{nk}\mathbf{x}_n}{R^{(k)}}$
Etapa 4: Repetição
se Atribuições modificadas **então**
    Repetir etapas 2 e 3
senão
    retorne $\{r_{nk}, \forall n \in 1, \ldots, M\}$

$\sum_n r_{nk}\mathbf{x}_n$ fornece apenas o ponto de dados no agrupamento k visto que r_{nk} é 0 para qualquer ponto de dados fora do *cluster* e $R^{(k)}$ é atualizado para $\sum_n r_{nk}$, que corresponde à quantidade total de pontos de dados no *cluster* k.

Etapa 4: Repetição

As etapas 2 e 3 são então repetidas até que os agrupamentos antes e após a iteração sejam os mesmos, o que significa que as médias atingiram o centroide de cada grupo e que a ação está concluída.

Complexidade temporal

Existem muitos métodos para obter a complexidade do algoritmo *k-means*, porém o mais comum é o dado por $O(MKIj)$ (ARTHUR; MANTHEY; RÖGLIN, 2009), em que M é o número de pontos de dados, K é o número de *clusters*, I é a dimensão do conjunto de pontos de dados e j é o número de iterações até a convergência, que é geralmente pequeno.

Na próxima seção abordadamos duas técnicas empregadas para a construção de PFSAs.

2.6 ALGORITMOS DE MODELAGEM PFSA

Nesta seção, dois algoritmos que constroem um PFSA a partir de uma sequência S de comprimento N sobre um alfabeto Σ são apresentados: Máquinas D-Markov (MUKHERJEE; RAY, 2014), e DCGraM (FRANCH, 2017).

2.6.1 Máquinas D-Markov

Uma máquina D-Markov é um PFSA que gera símbolos que são dependentes apenas do histórico de no máximo D símbolos anteriores, onde D é a profundidade da máquina. Ela gera um processo Markoviano $\{s_n\}$ de ordem D,

$$Pr(s_n|\dots s_{n-D}\dots s_{n-1}) = \Pr(s_n|s_{n-D}\dots s_{n-1}).$$

Para construir uma máquina D-Markov, primeiramente as possíveis sequências sobre Σ de comprimento D são utilizadas como estados de Q, formando um total de $|\Sigma|^D$ estados. Para cada $\sigma \in \Sigma$, a transição de rótulo σ a partir do estado q é determinada como segue: Considere que o estado q rotulado como $q = \sigma_1\sigma_2\dots\sigma_D$ com $\sigma_n \in \Sigma$, para $n = 1, 2, \dots, D$. Cria-se uma transição a partir de q para $q' = \sigma_2\dots\sigma_D\tau$, ou seja, $\delta(q, \tau) = q'$, com probabilidade:

$$\Pr(\tau|q) = \frac{\Pr(q\tau)}{\Pr(q)}, \tag{2.7}$$

em que $\Pr(q)$ e $\Pr(q\tau)$ correspondem as probabilidades das sequências $\sigma_1\sigma_2\dots\sigma_D$ e $\sigma_2\sigma_3\dots\sigma_D\tau$, respectivamente.

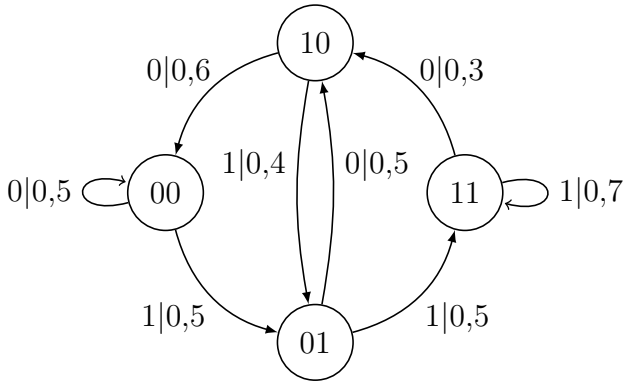
Por exemplo, considere uma sequência binária *S* com as probabilidades de palavras de comprimento $\ell \leq 3$ mostradas na Tabela 1. Para construir uma Máquina 2-Markov, os estados são 00, 01, 10 e 11. Utilizando (2.7), a máquina D-Markov mostrada na Figura 3 é gerada.

Tabela 1 – Probabilidades de palavras de comprimento até $\ell = 3$ obtidas a partir de uma sequência binária *S*.

$\ell = 1$	Prob.	$\ell = 2$	Prob.	$\ell = 3$	Prob.
0	0,51	00	0,27	000	0,15
1	0,49	01	0,23	001	0,12
		10	0,24	010	0,12
		11	0,25	011	0,11
				100	0,12
				101	0,12
				110	0,11
				111	0,14

Fonte: (FRANCH, 2017)

Figura 3 – Uma máquina D-Markov com D = 2.



Fonte: O autor (2019)

2.6.2 DCGraM

O Algoritmo DCGraM (FRANCH, 2017) corresponde a uma melhoria dos modelos D-Markov utilizando as técnicas de agrupamento de estados e minimização de grafos apresentadas anteriormente para gerar uma partição inicial e, em seguida, refiná-la recursivamente.

Para utilizar o algoritmo de clusterização, os *morphs* devem ser representados como pontos em um espaço $|\Sigma|$-dimensional. Para um dado estado q, seu *morph* é representado como um ponto $(\Pr(\sigma_1|q), \Pr(\sigma_2|q), \ldots, \Pr(\sigma_{|\Sigma|}|q))$ para $\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|} \in \Sigma$ no espaço $|\Sigma|$-dimensional. Como os *morphs* são distribuições de probabilidade, $\sum_{\sigma \in \Sigma} \Pr(\sigma|q) = 1, \forall q \in Q$, dadas essas restrições, os pontos de *morphs* são representados em um espaço $(|\Sigma| - 1)$-dimensional. Para criar eficientemente uma partição inicial $\mathcal{P}$ dos estados de uma máquina D-Markov (G_D, π_D), um algoritmo de agrupamento é usado para agrupar estados com *morphs* semelhantes.

A ideia central do DCGraM é usar uma técnica de minimização de grafo com a partição inicial $\mathcal{P}$ determinada pelo algoritmo de clusterização, produzindo um grafo reduzido com comportamento estatístico semelhante ao modelo D-Markov. Como os estados são agrupados de acordo apenas com seus *morphs*, estes possuem probabilidades semelhantes de gerar um símbolo, mas a probabilidade de gerar sequências mais longas pode ser diferente. Este é o critério usado para dividir esses *clusters* com o algoritmo de minimização de grafo. Depois que o algoritmo de minimização de grafo termina de refinar a partição inicial, os estados no mesmo *cluster* geram as sequências com probabilidades semelhantes.

O algoritmo DCGraM é exibido no Algoritmo 3 e cada uma das suas etapas são descritas a seguir.

Etapa 1: Criação de Máquina D-Markov

O DCGraM inicia pela criação de um modelo D-Markov (G_D, π_D) para um D específico, a partir de uma sequência de entrada S.

Etapa 2: Clusterização de estados

Seja $\mathcal{V}(q)$ o *morph* do estado q, como uma extensão direta, $\mathcal{V}(Q)$ representa o conjunto de *morphs* dos estados em Q. O algoritmo *k-means* aplicado ao DCGraM utiliza a métrica Euclidiana $\| x - \mathbf{m}^{(k)} \|^2$ para mensurar a distância entre pontos de dados.

O algoritmo funciona atribuindo primeiramente valores iniciais a $\mathbf{m}^{(k)}$. Em seguida, atribui cada ponto de dados x a um *cluster* j para o qual a distância $\| x - \mathbf{m}^{(j)} \|^2$ para $j \in \{1, \cdots, K\}$ é mínima. Os valores de $\mathbf{m}^{(k)}$ são então atualizados para o valor médio dos pontos de dados atribuídos a cada *cluster*. Essas etapas de atribuição e atualização são repetidas até que nenhuma atribuição seja alterada após uma iteração.

Os estados de G_D são agrupados usando a função auxiliar *kmeans*$(K, Q, \mathcal{V}(Q))$. Isso aplica o algoritmo *k-means* em $\mathcal{V}(Q)$ para criar K grupos e particionar o conjunto de estados Q de G_D seguindo os rótulos desses grupos, ou seja, os estados que recebem o mesmo rótulo por *k-means* são agrupados em uma mesma classe de equivalência. Isso resulta na partição inicial $\mathcal{P}$ do conjunto de estados de G_D e é usada como entrada para um algoritmo de minimização de grafo.

Etapa 3: Aplicação do algoritmo de minimização de grafo

A etapa final consiste em aplicar um algoritmo de minimização de grafo (Moore ou Hopcroft) às classes de equivalência na partição $\mathcal{P}$ para gerar a partição refinada final G_o na qual cada estado em cada classe de equivalência possui o mesmo contexto à direta e gera sequências com probabilidades semelhantes.

Uma vez obtido o G_o, π_o é calculado com a função *averageMorph*, e as classes de equivalência de G_o são transformadas em estados com *morphs* que são a média dos *morphs* dos estados na classe de equivalência. Finalmente, o PFSA reduzido (G_o, π_o) é retornado como a saída do DCGraM.

Algoritmo 3 DCGraM(S, D, K)

rotina DCGRAM
    ## Etapa 1: Criação de Máquina D-Markov
    $(G_D, \pi_D) \leftarrow$ *dmarkov*(S, D)
    ## Etapa 2: Clusterização de estados
    $\mathcal{P} \leftarrow$ *kmeans*$(K, Q, \mathcal{V}(Q))$
    ## Etapa 3: Aplicação do algoritmo de minimização de grafo
    $G_o \leftarrow$ *graphMinimization*$(G_D, \mathcal{P})$
    $\pi_o \leftarrow$ *averageMorphs*(G_o)
    retorne (G_o, π_o)

Complexidade Temporal

A complexidade temporal associada ao algoritmo DCGraM é dada pela soma das complexidades associadas a cada etapa, a saber: $O(N)$ para a Etapa 1, $O(|\Sigma|^{D+1}Ki)$ para a Etapa 2 e $O(D|\Sigma|^{D+1}\log|\Sigma|)$ para a Etapa 3. Assim, a complexidade do DCGraM é dada por $O(N+K|\Sigma|^{D+1}+|\Sigma|^{D}\log|\Sigma|) = O(N+K|\Sigma|^{D+1}i+D|\Sigma|^{D+1}\log|\Sigma|) = O(N+|\Sigma|^{D+1}(Ki+D\log|\Sigma|))$. O componente principal depende se o tamanho da sequência original ou o tamanho do D-Markov gerado é maior.

Embora o DCGraM seja necessariamente mais complexo que o D-Markov (visto que uma de suas etapas inclui a geração de uma máquina D-Markov), seus resultados finais são consideravelmente mais compactos do que os das máquinas D-Markov, com melhores resultados ou desempenho semelhante, justificando a vantagem de se ter um PFSA reduzido, mesmo sendo necessário mais tempo para gerá-lo.

3 ALGORITMO D-MARKOV DE COMPRIMENTO VARIÁVEL COM CLUSTERIZAÇÃO E MINIMIZAÇÃO DE GRAFO

Neste capítulo, apresentamos um algoritmo para modelagem de sistemas dinâmicos discretos, o algoritmo D-Markov de Comprimento Variável com Clusterização e Minimização de Grafo (VL-DCGraM). Na Seção 3.1 aborda-se a obtenção de sequências simbólicas a partir de processos de discretização de séries temporais. Na Seção 3.2 é apresentado um processo Markoviano denominado Cadeia de Markov de Comprimento Variável e sua representação sob uma estrutura em árvore, denominada árvore de contextos (BÜHLMANN, 2000), que apresenta relação com o algoritmo proposto. Finalmente, na Seção 3.3 o algoritmo VL-DCGraM é descrito detalhadamente.

3.1 SIMBOLIZAÇÃO DE SÉRIES TEMPORAIS

Realizar medições de séries temporais implica lidar com fatores como: taxa de amostragem, resposta dinâmica dos instrumentos e relação sinal-ruído; que podem afetar a qualidade dos resultados. Um meio de minimizar os efeitos negativos, como o ruído, por exemplo, é a adoção de técnicas de simbolização, cuja principal característica é a discretização das amostras que constituem a série temporal em uma sequência correspondente de símbolos, os quais pertencem a um conjunto finito e retém grande parte das informações temporais relevantes (DAW; FINNEY; TRACY, 2003).

O método de simbolização tem como primeira etapa particionar o intervalo de valores das amostras da série temporal em segmentos, de modo que cada amostra pertence a um segmento único. A simbolização é obtida mapeando cada segmento em um símbolo distinto, de modo que amostras com valores em um mesmo segmento são mapeadas no mesmo símbolo. A sequência simbólica obtida é inspecionada em uma escala de tempo rápida e em uma escala de tempo lenta. Na escala de tempo rápida, supõe-se que o processo tenha dinâmica estatisticamente estacionária; na escala de tempo lenta, o processo pode exibir dinâmica não estacionária, que pode ser associada a uma evolução gradual das propriedades estatísticas da dinâmica, em consequência de uma evolução gradual de anomalias (MUKHERJEE; RAY, 2014). A etapa seguinte consiste em identificar sequências relevantes de comprimento finito que refletem padrões dinâmicos subjacentes significativos (LI; RAY, 2017), ou seja, sequências finitas cuja observação possibilite captar a estatística de sequências subsequentes. Tais sequências compõem os blocos de construção para geração de modelos dinâmicos a partir da sequência simbólica. As características desejadas de um modelo dinâmico são sua simplicidade e precisão analíticas para refletir o comportamento estatístico da sequência simbólica, sendo por isso computacionalmente atraente como modelo de um sistema.

Dada uma série temporal discretizada, buscamos a construção de um modelo que a represente. Considerando uma sequência de símbolos S no alfabeto de quantização, pretendemos

modelá-la por meio de uma máquina Markoviana, a qual é abordada neste capítulo.

3.2 CADEIAS DE MARKOV DE COMPRIMENTO VARIÁVEL E ÁRVORES DE CONTEXTOS

Seja uma sequência $x_j^i = x_j, x_{j-1}. \cdots, x_i$, $i > j$, $i, j \in \mathbb{Z} \cup \{-\infty, \infty\}$. Denotando por X uma variável aleatória e por x seus valores específicos, temos a seguinte definição.

Definição 3.1 (Cadeia de Markov). Seja $(X_t)_{t\in\mathbb{Z}}$ um processo estacionário sobre um alfabeto finito Σ. O processo é denominado cadeia de Markov de ordem finita D se

$$\Pr(X_1 = x_1 \,|\, X_{-\infty}^0 = x_{-\infty}^0) = \Pr(X_1 = x_1 \,|\, X_{-D+1}^0 = x_{-D+1}^0), \tag{3.1}$$

para todo $x_{-\infty}^1$. Em uma cadeia de Markov de ordem finita D a probabilidade de ocorrência de um símbolo atual x_1 depende apenas da ocorrência dos D símbolos passados. Este processo pode ser representado por uma máquina D-Markov com $|\Sigma|^D$ estados descrita na Seção 2.6.1.

Estimar os parâmetro de cadeias de Markov é um tarefa árdua, uma vez que estas envolvem $|\Sigma|^D(|\Sigma| - 1)$ parâmetros livres, em que $|\Sigma|$ denota a cardinalidade de Σ. Contudo, há classes de processos em que (3.1) é satisfeita para valores menores que D, para algumas sequências passadas x_{-D+1}^0, e, portanto, podem ser representadas por PFSAs mais compactos, em que estados da máquina D-Markov podem ser colapsados. Para a formalização destes processos, definimos a seguir o conceito de função contexto.

Definição 3.2 (Função contexto). Seja $(X_t)_{t\in\mathbb{Z}}$ um processo estacionário sobre um alfabeto Σ. Denotamos por $c : \Sigma^\infty \to \cup_{\beta=0}^\infty \Sigma^\beta$ uma função que mapeia uma sequência infinita em uma sequência possivelmente finita (passado relevante) (BÜHLMANN, 2000)

$$c : x_{-\infty}^0 \mapsto x_{-l+1}^0, \tag{3.2}$$

em que l é definido por

$$l = \min\{k; \Pr(X_1 = x_1 \,|\, X_{-\infty}^0 = x_{-\infty}^0) = \Pr(X_1 = x_1 \,|\, X_{-k+1}^0 = x_{-k+1}^0)\}$$

para todo $x_1 \in \Sigma$. Assim, $c(\cdot)$ é chamada de função contexto.

O nome contexto refere-se à sequência de símbolos passados que influencia na ocorrência do símbolo atual. A partir de $c(\cdot)$, a função comprimento de contexto $l(\cdot) = |c(\cdot)|$ determina $c(\cdot)$ e vice-versa. A definição de l reflete implicitamente o fato de que o comprimento do contexto de uma variável x_t é $l = |c(x_{-\infty}^{t-1})| = l(x_{-\infty}^{t-1})$, que só depende do histórico $x_{-\infty}^{t-1}$.

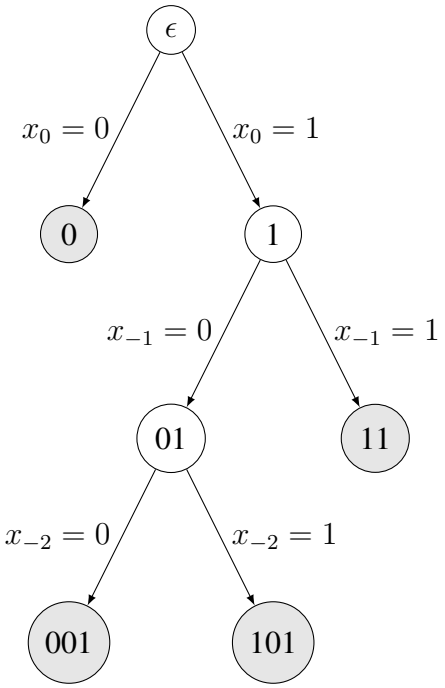
Definição 3.3 (Cadeia de Markov de comprimento variável). Seja $(X_t)_{t\in\mathbb{Z}}$ um processo estacionário sobre um alfabeto Σ e função contexto descrita conforme a Definição 3.2. Seja $0 \leq D \leq \infty$ o menor inteiro tal que

$$|c(x_{-\infty}^0)| = l(x_{-\infty}^0) \leq D, \, \forall x_{-\infty}^0 \in \Sigma^\infty.$$

Então $c(\cdot)$ é chamada uma função contexto de ordem D, e $(X_t)_{t\in\mathbb{Z}}$ é chamada uma cadeia de Markov estacionária de comprimento variável (VLMC) de ordem D (BÜHLMANN, 2000). Uma VLMC de ordem D corresponde a uma cadeia de Markov de ordem D.

Uma forma de representar uma VLMC é por meio de uma árvore de contextos $\mathcal{CT}$. Um exemplo dessa estrutura para $\Sigma = \{0, 1\}$ e $D = 3$ é apresentado na Figura 4. Uma VLMC consiste de um conjunto de estados denominados nós, conectados por transições rotuladas com os símbolos do alfabeto. Cada nó possui exatamente um predecessor (com exceção do nó inicial, chamado nó raiz, rotulado pela sequência vazia ϵ, que não possui predecessor). Nós denominados folhas (0, 11, 001 e 101 na Figura 4) não apresentam sucessores, enquanto os demais possuem $|\Sigma|$ sucessores. Exceto o nó raiz, cada nó é rotulado com a sequência formada pelo rótulo do nó predecessor concatenado à direita com o símbolo que rotula a transição para si. Considere, por exemplo, o nó 01. Seu rótulo consiste da concatenação do rótulo do nó 1 (correspondendo ao símbolo mais recente) com o rótulo 0, referente ao rótulo da transição para o nó em questão. O rótulo de uma folha corresponde a um contexto da VLMC. Todo contexto da VLMC está associado a um caminho entre o nó raiz e uma folha.

Figura 4 – Exemplo de árvore de contextos com $\Sigma = \{0, 1\}$ e $D = 3$.



Fonte: O autor (2019)

Como exemplo de construção de uma árvore de contextos, consideremos $\Sigma = \{0, 1\}$ e

$D = 3$. A função

$$c(x_{-\infty}^{0}) = \begin{cases} 0, & \text{se } x_0 = 0, x_{-\infty}^{-1} \text{ arbitrário} \\ 001, & \text{se } x_0 = 1, x_{-1} = 0, x_{-2} = 0, x_{-\infty}^{-3} \text{ arbitrário} \\ 101, & \text{se } x_0 = 1, x_{-1} = 0, x_{-2} = 1, x_{-\infty}^{-3} \text{ arbitrário} \\ 11, & \text{se } x_0 = 1, x_{-1} = 1, x_{-\infty}^{-2} \text{ arbitrário} \end{cases} \tag{3.3}$$

pode ser representada pela árvore de contextos na Figura 4.

3.3 ALGORITMO VL-DCGRAM

Os modelos D-Markov geralmente atingem um bom desempenho à medida que D aumenta. O custo para esse desempenho aprimorado é um crescimento exponencial no número de estados. O algoritmo VL-DCGraM é proposto como um aprimoramento do modelo DCGraM, e consequentemente dos modelos D-Markov, associando técnicas de agrupamento e minimização de grafos com a redução do número de estados iniciais a serem analisados, com o objetivo de reduzir o tamanho da máquina final.

A diferença básica entre o VL-DCGraM e o DCGraM consiste em substituir a construção inicial de uma máquina D-Markov pela construção de uma VLMC. O objetivo é construir uma máquina composta por estados rotulados com palavras de comprimento variável, dando origem a uma VLMC e proporcionando com isso um menor número de estados iniciais a serem aplicados à técnica de minimização de grafos com partição inicial $\mathcal{P}$ determinada pelo algoritmo de clusterização. O resultado é um grafo reduzido com comportamento estatístico semelhante ou superior ao modelo DCGraM. Como os estados são agrupados de acordo apenas com seus *morphs*/probabilidades, eles possuem probabilidades semelhantes de gerar um símbolo, porém a probabilidade de gerar sequências mais longas pode ser distinta. Esta é a motivação para aplicar o algoritmo de minimização de grafos. Após o refinamento da partição inicial a partir do algoritmo de minimização, os estados nas mesmas classes geram as sequências em seu contexto à direita com probabilidades semelhantes.

3.3.1 Algoritmo *k-means* ponderado

Como contribuição deste trabalho, propomos uma variação do método *k-means* apresentado na Seção 2.5.1. Agora o centroide do *cluster* é calculado pela média ponderada de seus pontos, com o objetivo de minimizar a distância entre tais pontos em relação ao centroide $\mathbf{m}^{(k)}$.

A função objetiva é baseada numa média ponderada dada pela expressão

$$J_{pond} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} p_n \parallel \mathbf{x}_n - \mathbf{m}^{(k)} \parallel^2 \tag{3.4}$$

em que p_n representa a probabilidade associada ao ponto de dado $\mathbf{x}_n$. A fim de encontrar valores para os centroides $\mathbf{m}^{(k)}$ e as variáveis binárias r_{nk} que minimizem J_{pond}, realiza-se um processo

iterativo de otimizações sucessivas em relação a r_{nk} e $\mathbf{m}^{(k)}$ conforme descrito na Seção 2.5.1. Para a otimização de $\mathbf{m}^{(k)}$, mantemos r_{nk} fixo e igualamos a derivada da função J_{pond} a zero, obtendo

$$2\sum_{n=1}^{N} r_{nk}p_n(\mathbf{x}_n - \mathbf{m}^{(k)}) = 0,$$

do que segue

$$\mathbf{m}^{(k)}\sum_{n=1}^{N} r_{nk}p_n = \sum_{n=1}^{N} r_{nk}p_n\mathbf{x}_n,$$

e podemos concluir que

$$\mathbf{m}^{(k)} = \frac{\sum_{n=1}^{N} r_{nk}p_n\mathbf{x}_n}{\sum_{n=1}^{N} r_{nk}p_n}.$$

Desta última expressão verificamos que a cada um dos elementos $\mathbf{x}_i$ contidos no *cluster* k é adicionada sua probabilidade $p_i/\sum_{n=1}^{N} r_{nk}p_i$, o que corresponde a realizar uma média ponderada de todos os pontos $\mathbf{x}_n$ associados ao k-ésimo *cluster* para determinação de seu centroide.

Uma comparação entre os resultados obtidos com a aplicação dos algoritmos *k-means* tradicional e ponderado para os modelos DCGraM e VL-DCGraM é apresentada no capítulo seguinte, a fim de ilustrar a vantagem de adoção do algoritmo *k-means* ponderado em relação ao tradicional.

A descrição do VL-DCGraM utilizando o método *k-means* ponderado é mostrado no Algoritmo 4 e cada uma de suas etapas é discutida a seguir.

Algoritmo 4 VL-DCGraM $(L, M, \Sigma, \alpha, T)$

---

rotina VL-DCGRAM
  ## Passo 1: Criação de árvore de contextos $\mathcal{CT}$
  $S \leftarrow \{\epsilon\}$ # S armazena os nós da árvore
  $\mathcal{CT} \leftarrow (S, \delta, \mathcal{V}(S))$
  para $\omega \leftarrow NodesPreWalk(\mathcal{CT})$ **faça** # caminhada em pré-ordem pelos nós da árvore
    se $\Delta[\mathcal{V}(\alpha\omega), \mathcal{V}(\beta\omega)] > T$ para algum $\alpha, \beta \in \Sigma$ e $|\omega| < L$ **então**
      $S \leftarrow S \cup (\bigcup_{a\in\Sigma} a\omega)$
      para $a \in \Sigma$ **faça**
        $\delta(a, \omega) = \omega a$
    senão
      $Leafs \leftarrow \omega$
  ## Passo 2: Criação de máquina de estados
  $Q \leftarrow Leafs$
  $Conex[Q]$
  $Purging[Q, \delta]$
  $Maq \leftarrow (Q, \Sigma, \delta, \pi)$
  ## Passo 3: Análise da máquina gerada

$\mu \leftarrow OccupVect[Maq]$ # $OccupVect$ calcula o vetor de ocupação de Maq
$h \leftarrow -\sum_i \mu_i \log_{10} \mu_i$
$l_\mu \leftarrow 10^{-\alpha h}$
para $v \in Q$ e $|v| < M$ **faça**
  se $\mu_v > l_\mu$ **então** # μ_v é a probabilidade do estado, obtido a partir de μ
    $Q \leftarrow \left(Q \cup \bigcup_{a\in\Sigma} av\right) \setminus \{v\}$ # Inclusão dos filhos de v em Q e eliminação de v
    $Conex[Q]$
    $Purging[Q, \delta]$
    Retornar ao início do Passo 3
Passo 4: Clusterização de estados
$\mathcal{P} \leftarrow$ *kmeans*$(K, Q, \mathcal{V}(Q))$
Passo 5: Aplicação de minimização de grafo
$G_o \leftarrow$ *graphMinimization*$(Maq, \mathcal{P})$
$\pi_o \leftarrow$ *averageMorphs*(G_o)
retorne (G_o, π_o)
 # Função conexão de estados $Conex[Q]$**:**
para todo $\omega \in Q$ **faça**
  para todo $a \in \Sigma$ **faça**
    se $P(a|\omega) > 0$ **então**
      se $suffix(\omega a) \cap Q \neq \varnothing$ **então**
        $\delta(\omega, a) = \text{argmax}_{v\in Q} |v \cap suffix(\omega a)|$
      senão
        se $|\omega a| \leqslant M$ **então**
          $Q \leftarrow Q \cup \omega a$
          $\delta(\omega, a) = \omega a$
        senão
          $v \leftarrow suffmax(\omega a)$ # máximo sufixo próprio do argmax
          $Q \leftarrow Q \cap v$
          $\delta(\omega, a) = v$
 # Função eliminação de estados inacessíveis $Purging[Q, \delta]$**:**
repita
  $Q' \leftarrow Q$
  $Q \leftarrow Q' \cap \left(\bigcup_{a\in\Sigma} \delta(Q, a)\right)$
até que $Q' = Q$

Passo 1: Construção de árvore de contextos

A etapa inicial do algoritmo consiste na criação de uma árvore de contextos $\mathcal{CT}$ cujas ramificações são construídas seguindo um percurso em pré-ordem (WEISS, 2002), denominada de $NodesPreWalk(\mathcal{CT})$, para cada um de seus nós com rótulo ω. Para determinar se um nó será

ou não ramificado, empregamos uma extensão direta do conceito de *morphs* visto na Definição 2.7. Para todo $\omega \in \Sigma^*$, define-se o *morph* de ω como o conjunto

$$\mathcal{V}(\omega) = \{\pi(\omega, \sigma) = \Pr(\omega a)/\Pr(\omega), \forall \sigma \in \Sigma\}.$$

As ramificações são geradas a partir do nó raiz da seguinte maneira: se a distância euclidiana entre os *morphs* das correspondentes transições de ω com algum $\alpha, \beta \in \Sigma$, dada por

$$\Delta[\mathcal{V}(\alpha\omega), \mathcal{V}(\beta\omega)] = |\mathcal{V}(\alpha\omega) - \mathcal{V}(\beta\omega)|$$

é tal que $\Delta[\mathcal{V}(\alpha\omega), \mathcal{V}(\beta\omega)] > T$, ou seja, a distância entre os *morphs* tem valor superior a um dado valor T, e o comprimento de palavra ω for menor que L (profundidade estipulada para a árvore), nós são criados a partir de ω com as transições $\delta(a, \omega), \forall a \in \Sigma$, e $NodesPreWalk$ é aplicada ao nó subsequente. Este processo é realizado para cada novo nó criado até que esse não permita ramificações (o valor de $\Delta[\mathcal{V}(\alpha\omega), \mathcal{V}(\beta\omega)]$ é inferior a T) ou seu comprimento de palavra atinja a profundidade L da árvore, condição na qual é incluído no conjunto de folhas $Leafs$, caracterizando-o como um contexto da árvore de contextos.

Uma vez encerrada a caminhada, as folhas obtidas são utilizadas para a criação de uma máquina de estados, descrita no passo a seguir.

Passo 2: Criação de máquina de estados

Nesta etapa, o conjunto de folhas da $\mathcal{CT}$, $Leafs$, é utilizado para criação de uma máquina de estados Maq correspondente ao grafo descrito pela tupla (Q, Σ, δ, π). Inicialmente, Q recebe os contextos em $Leafs$ e é aplicada a estes a função $Conex[Q]$ para gerar a conexão entre os estados do seguinte modo: para todo $\omega \in Q$ e para todo $a \in \Sigma$, se $P(a \,|\, \omega) > 0$, existindo um estado em Q cujo rótulo corresponde a um sufixo para a concatenação de ω com a (representado por $\mathit{suffix}(\omega a)$ na descrição da função), a conexão será dada por

$$\delta(\omega, a) = \underset{v \in Q}{\operatorname{argmax}} \, |v \cap \mathit{suffix}(\omega a)|$$

ou seja, a transição ocorre para o maior sufixo de ωa que é rótulo de um elemento em Q. Caso não existam elementos em $\mathit{suffix}(\omega a) \cap Q$, um novo estado é acrescentado em Q de acordo com seu comprimento: se $|\omega a| \leqslant M$, adiciona-se a concatenação ωa; se $|\omega| > M$, adiciona-se o estado v correspondente ao máximo sufixo próprio da concatenação ωa, $\mathit{suffmax}(\omega a)$. Após a adição do novo estado, a conexão é estabelecida.

Uma vez estabelecidos todos os estados Q e conexões δ, é aplicada a estes a função $Purging[Q, \delta]$ para que sejam eliminados estados inacessíveis em Q, isto é, estados que não possuem transições de chegada. Nesta função, um conjunto Q' recebe os estados atuais de Q, e em seguida este último é atualizado para

$$Q = Q' \cap \left(\bigcup_{a \in \Sigma} \delta(Q, a) \right),$$

em que $\bigcup_{a\in\Sigma}\delta(Q,a)$ corresponde a todos os estados alcançados por algum estado de Q. Esta atribuição é realizada recursivamente, removendo-se os estados inacessíveis e suas transições associadas até que se tenha $Q' = Q$, indicando a obtenção de um conjunto contendo apenas estados alcançáveis.

Uma vez obtida a máquina de estados Maq, esta é submetida às ações de análise descritas no passo a seguir.

Passo 3: Análise da máquina gerada

Nesta etapa do algoritmo, os estados que compõem a máquina obtida no final do Passo 2 são analisados quanto a sua probabilidade de ocupação, a fim de verificar a necessidade de inclusão de novos estados. O objetivo é eliminar padrões estatísticos que venham a surgir na sequência gerada a partir da máquina, decorrente da visitação sucessiva de um estado que possua alta probabilidade de ocupação. Como uma consequência adicional, isso diminui a disparidade entre as probabilidades associadas aos estados da máquina, tornando mais homogênea a distribuição de probabilidade dos estados.

Dada a máquina Maq, calcula-se o vetor estacionário $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_{|Q|}]$ que armazena os valores das probabilidades de ocupação de seus estados, e a partir deste é calculado um limiar l_μ, dado por

$$l_\mu = 10^{-\alpha h},$$

em que α é um valor real positivo e h a entropia dada por

$$h = -\sum_{i=1}^{|Q|} \mu_i \log_{10} \mu_i.$$

Obtido o valor para l_μ, verifica-se para cada estado $v \in Q$ e com comprimento do rótulo menor que M ($|v| < M$) se sua probabilidade de ocupação μ_v é superior a l_μ. Em caso positivo, v é substituído por suas extensões com os símbolos do alfabeto, de modo a termos

$$Q = \left(Q \cup \bigcup_{a\in\Sigma} av \right) \setminus \{v\}.$$

Uma vez modificado o conjunto Q de Maq, esta é submetida às funções $Conex[Q]$ e $Purging[Q, \delta]$ para verificação de novas conexões entre estados e remoção de possíveis estados inacessíveis, e a máquina resultante é submetida ao início do Passo 3. Dessa forma, esta etapa do algoritmo é reexecutada até que seja obtida uma máquina na qual todos os estados v, $|v| < M$, possuam $\mu_v < l_\mu$, sendo esta utilizada nos Passos 4 e 5, descritos a seguir.

Passo 4: Clusterização de estados

Nesta etapa, a variação do algortimo *k-means* tradicional apresentada na Seção 3.3.1 é aplicada à máquina de estados atual para gerar uma partição inicial a ser utilizada pelo

algoritmo de minimização de grafos. Os estados de Maq são agrupados usando a função auxiliar $kmeans(K, Q, \mathcal{V}(Q))$, que aplica o algoritmo *k-means* ponderado no conjunto de *morphs* dos estados dá máquina, $\mathcal{V}(Q)$, para criar K *clusters* e particionar o conjunto de estados Q de Maq de acordo com os rótulos dos grupos formados, ou seja, os estados que recebem o mesmo rótulo via *k-means* são agrupados em uma mesma classe de equivalência. Isso resulta na partição inicial $\mathcal{P}$ do conjunto de estados de Maq, que é usada como entrada para um algoritmo de minimização de grafos, etapa descrita a seguir.

Passo 5: Aplicação de algoritmo de minimização

Como etapa final e de forma similar ao realizado na Etapa 3 do Algoritmo DCGraM, o algoritmo de minimização de grafos Moore é aplicado às classes de equivalência na partição $\mathcal{P}$, gerando uma partição final G_o na qual cada estado em cada classe de equivalência possui o mesmo contexto à direita e gera sequências no contexto à direita com probabilidades semelhantes.

Obtido G_o, calcula-se π_o para cada classe por meio da função *averageMorph*, e as classes de equivalência de G_o são convertidas em estados com *morphs* que são o *morph* médio dos estados nessa classe de equivalência. Finalmente, o PFSA reduzido (G_o, π_o) é retornado como saída do VL-DCGraM.

No próximo capítulo, o algoritmo VL-DCGraM é aplicado a exemplos de sistemas dinâmicos para geração de PFSAs e os resultados obtidos são comparados utilizando modelos D-Markov e DCGraM descritos no Capítulo 2 para alguns quantificadores de desempenho.

4 RESULTADOS

Neste capítulo, a eficiência do algoritmo proposto no Capítulo 3 para construção de um PFSA é verificada por meio de alguns exemplos de sistemas dinâmicos usados para comparar os resultados obtidos pelos modelos VL-DCGraM, D-Markov e DCGraM. São considerados dois exemplos de sistemas dinâmicos: o mapa logístico, que é um mapa que apresenta comportamento caótico, e o canal de comunicação binário com desvanecimento, útil para modelamento de comunicações sem fio. O desempenho de cada modelo é avaliado utilizando os quantificadores descritos na Seção 4.1 e os resultados das comparações são expostos na Seção 4.2.

4.1 QUANTIFICADORES DE DESEMPENHO

Esta seção apresenta três quantificadores usados para comparar o desempenho do PFSA gerado pelos algoritmos supracitados. O primeiro é a entropia condicional usada para calcular a taxa de entropia, o que dá um indicativo da memória do sistema. O segundo é a divergência Kullback-Leibler, que compara a probabilidade de sequências geradas pelo modelo e pelo sistema dinâmico. O terceiro é a função autocorrelação.

4.1.1 Taxa de Entropia de ℓ-ésima ordem

Seja $\{X_k\}_{k=1}^{\infty}$ um processo aleatório discreto sobre Σ. Sua taxa de entropia é definida como (COVER; THOMAS, 2012):

$$h \triangleq \lim_{k\to\infty} H(X_k|X_1X_2\ldots X_{k-1}) = -\lim_{k\to\infty} \sum_{x\in\Sigma^k} \Pr(x)\log_{10}\Pr(x_k|x_1x_2\ldots x_{k-1}). \tag{4.1}$$

Para um processo estacionário, a entropia condicional $H(X_k|X_1\ldots X_{k-1})$ é não-crescente em k e converge para h conforme k tende a infinito (COVER; THOMAS, 2012). Como não é viável calcular (4.1), a entropia condicional de ℓ-ésima ordem é utilizada, e é definida como:

$$h_\ell \triangleq H(X_\ell|X_1X_2\ldots X_{\ell-1}), \tag{4.2}$$

que mede a incerteza de uma variável aleatória X_ℓ, considerando as $\ell-1$ amostras anteriores. A comparação de h_ℓ de um PFSA gerado a partir da sequência proveniente do sistema original é útil para testar se o primeiro captura corretamente a memória do segundo.

4.1.2 Divergência Kullback-Leibler de ℓ-ésima ordem

Para fins de comparação dos algoritmos, considere duas sequências S_1 e S_2 sobre um alfabeto comum Σ. Estas podem ser a sequência original S ou uma sequência gerada por um PFSA. Seja $\omega \in \Sigma^\ell$ uma palavra de comprimento ℓ e $P_1(\omega)$ e $P_2(\omega)$ sejam as probabilidades de ocorrência de ω em S_1 e S_2, respectivamente. Para um dado ℓ, a divergência de Kullback-Leibler

de ℓ-ésia ordem é dada por:

$$D_\ell(S_1||S_2) = \sum_{\omega \in \Sigma^\ell} P_1(\omega) \log \Big(\frac{P_1(\omega)}{P_2(\omega)} \Big). \tag{4.3}$$

Embora tecnicamente não seja uma distância, pois não obedece à desigualdade triangular nem é necessariamente comutativa, a divergência de Kullback-Leibler é útil para dar uma ideia de quão semelhantes são duas distribuições. Uma pequena divergência indica que o PFSA gera sequências cujas palavras, até um dado comprimento L, possuem distribuição de probabilidade próxima a da sequência original, o que pode ser interpretado como uma similaridade entre essas, sendo uma boa estimativa para o sistema original.

4.1.3 Função autocorrelação

A função autocorrelação $R[m]$ para um processo estocástico estacionário $\{Z_k\}_{k=0}^{\infty}$ é definida por

$$R[m] = E[Z_k Z_{k+m}],$$

em que $E[\cdot]$ indica valor esperado. Para um PFSA, existe uma expressão matricial para $R[m]$ dada por (PIMENTEL; BLAKE, 1999)

$$R[m] = \boldsymbol{\mu}\mathbf{P}(1)\mathbf{P}^{m-1}\mathbf{P}(1)\mathbf{1}, \tag{4.4}$$

em que o vetor estacionário $\boldsymbol{\mu}$, a matriz de transição de probabilidade $\mathbf{P}$, e a matriz $\mathbf{P}(1)$ são definidas na Seção 2.4 e $\mathbf{1}$ é um vetor coluna com todos os elementos iguais a 1. A função autocorrelação para uma sequência original S é computada por simulações.

4.2 APLICAÇÕES

Nesta seção, o modelo proposto VL-DCGraM é aplicado a dois tipos de sistemas: o mapa logístico e o canal binário com desvanecimento. Resultados de comparações com o D-Markov e DCGraM também são apresentados.

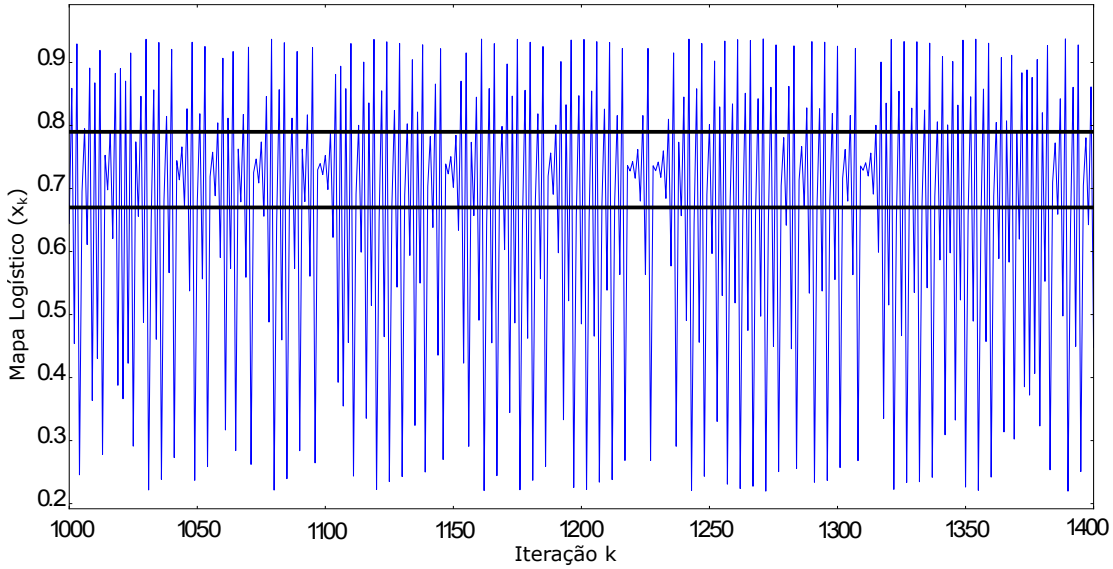
4.2.1 Mapa Logístico

O mapa logístico é um sistema dinâmico cujas amostras são obtidas iterativamente (a partir de um valor inicial x_0) da seguinte forma (STROGATZ, 2001):

$$x_{k+1} \triangleq r x_k(1 - x_k), \text{ for } k = 0, 1, 2, \ldots \tag{4.5}$$

com $x_k, r \in \mathbb{R}$. Este sistema apresenta comportamento caótico para vários valores de r. Como em (MUKHERJEE; RAY, 2014), x_0 é fixado em $0,5$ e $r = 3,75$. Uma sequência de comprimento 10^7 é gerada a partir de (4.5) e em seguida é quantizada com um alfabeto ternário da seguinte

Figura 5 – Amostras da sequência do mapa logístico ternário geradas por (4.5) com x_0 = 0,5 e r = 3,75.



Fonte: (FRANCH, 2017)

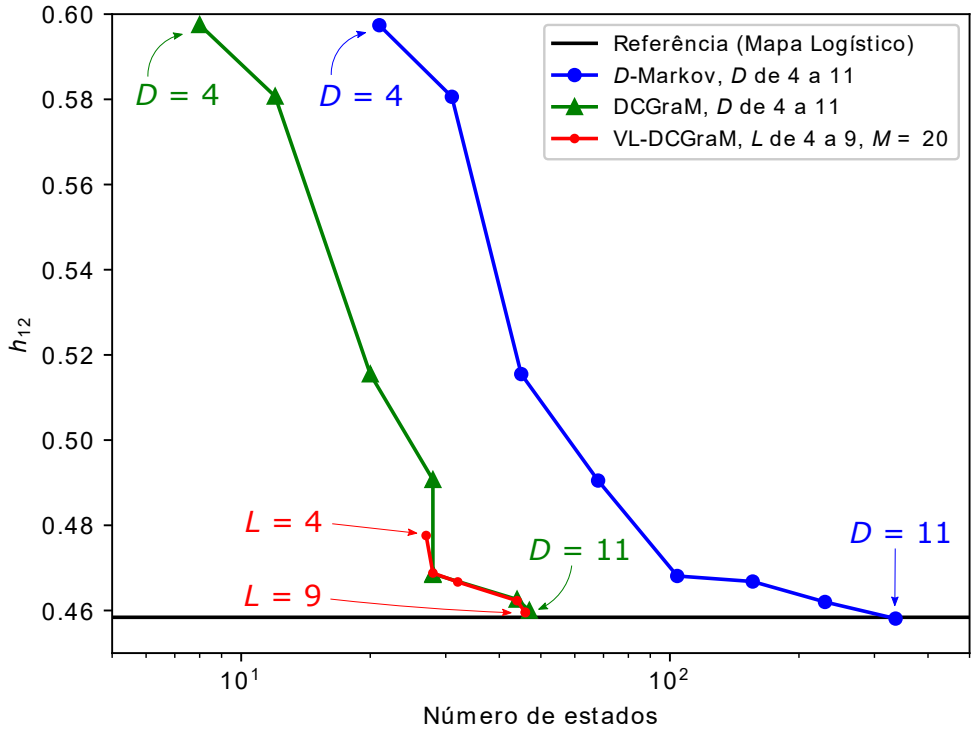
forma: valores $x_k \leq 0,67$ são mapeados para 0; quando $0,67 < x_k \leq 0,79$, é mapeado para 1 e quando $x_k > 0,79$ é mapeado para 2. Uma realização da sequência x_k e os limiares especificados são mostrados na Figura 5.

Máquinas D-Markov são geradas a partir das sequências ternárias S para D de 4 a 11 e utilizadas no algoritmo DCGraM com parâmetro $K = 5$ no *k-means*, enquanto as máquinas VL-DCGraM são geradas com L de 4 a 8 e utilizam o algoritmo *k-means* ponderado com $K = 5$. Vários valores para os parâmetros T, M e α foram testados, e os melhores resultados foram obtidos com $T = 0,4$, $M = 20$ e $\alpha = 2$.

Os resultados comparativos para entropia condicional com $\ell = 12$ versus o número de estados são mostrados na Figura 6. O valor de entropia h_{12} para a sequência ternária do mapa logístico também é mostrado, para servir como referência. Como a escala do eixo-x é logarítmica, é possível ver que o número de estados das máquinas D-Markov e do DCGraM cresce com D, embora o número de estados do DCGraM cresça mais lentamente que os das máquinas D-Markov. Para um dado D, é possível ver que a qualidade da máquina D-Markov e do DCGraM é semelhante. Ainda na Figura 6, observa-se que o modelo VL-DCGraM tende ao valor de entropia condicional da sequência original mais rapidamente que o modelo DCGraM, uma vez que o modelo D-Markov atinge a entropia do sistema original com $D = 11$, o DCGraM com $D = 10$ e o VL-DCGraM com $L = 8$.

Os resultados para a divergência de Kullback-Leibler para $\ell = 12$ versus o número de estados são mostrados na Figura 7. Observa-se nessa figura que a divergência da máquina VL-DCGraM com $L = 8$ tem número de estados e valor de divergência similar ao obtido com o modelo DCGraM para D = 11. Pode-se notar com isso que o modelo VL-DCGraM demonstra potencial para criar PFSAs capazes de gerar sequências semelhantes ao do sistema original.

Figura 6 – Entropia Condicional h_{12} versus número de estados da sequência gerada pelos modelos D-Markov, DCGraM e VL-DCGraM para o mapa logístico ternário com valores distintos de D e L.

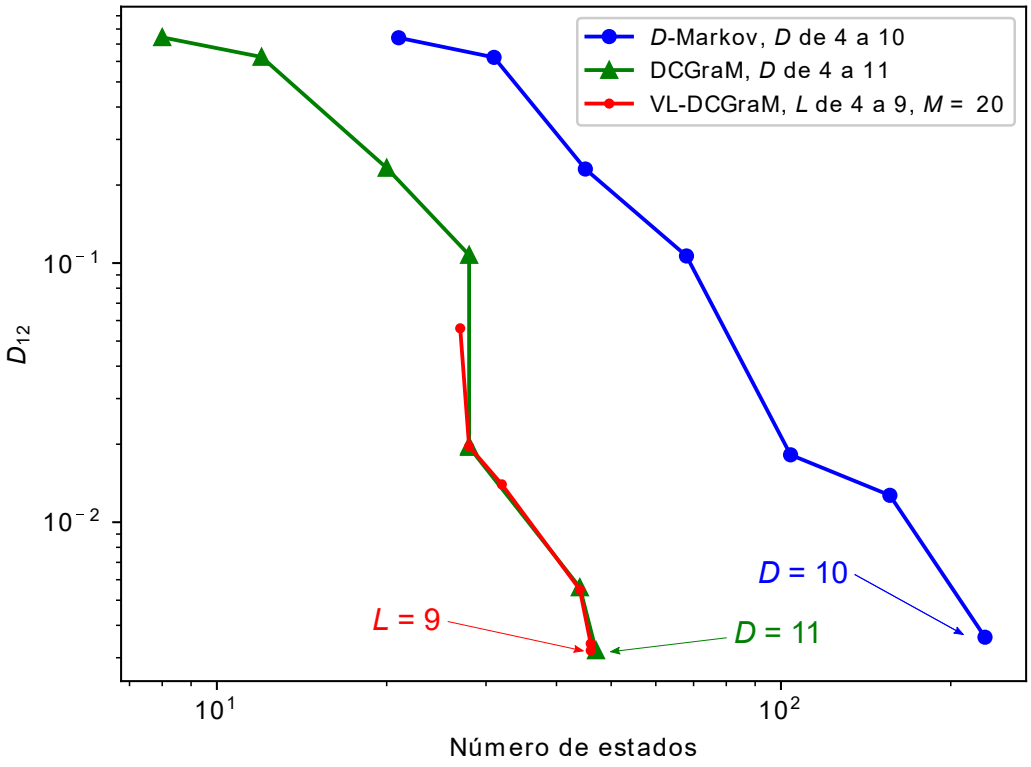


Fonte: O autor (2019)

Uma vantagem do modelo VL-DCGraM sobre o DCGraM é que, enquanto este último gera uma máquina inicial (D-Markov) com elevado número de estados que é a entrada dos algoritmos de clusterização e minimização de estados, a máquina inicial do VL-DCGraM é gerada por uma cadeia de Markov de comprimento variável que é mais compacta que a D-Markov. A Tabela 2 apresenta uma comparação do número de estados da máquina inicial e da máquina após as etapas de clusterização e minimização (máquina gerada por cada modelo). Por exemplo, máquinas VL-DCGraM com $L = 9$ e $M = 20$ e DCGraM com $D = 11$ têm aproximadamente os mesmos valores de h_{12} e D_{12}, mas a máquina inicial tem 131 estados e 336 estados, respectivamente.

Na Figura 8 é mostrado o gráfico da função autocorrelação referente às sequências geradas pelas máquinas D-Markov, DCGraM e VL-DCGraM. É possível notar que as curvas associadas ao modelo VL-DCGraM encontram-se mais próximas da curva do sistema original quando comparada às dos modelos D-Markov e DCGraM para D e L na faixa de 4 a 7. Na Figura 9 enfatiza-se que o modelo VL-DCGraM com $L = 5, M = 20$ apresenta maior aderência à curva de autocorrelação do sistema original e com menor número de estados que o modelo D-Markov e mesma quantidade de estados que o modelo DCGraM, ambos com parâmetro $D = 7$, o que representa uma vantagem do modelo proposto em relação aos demais.

Figura 7 – Divergência de Kullback-Leibler D_{12} versus número de estados da sequência gerada por máquinas D-Markov, DCGraM e VL-DCGraM comparada com a sequência original S do mapa logístico ternário para valores distintos de D e L.



Fonte: O autor (2019)

Tabela 2 – Número de estados das máquinas inicial e após minimização para os modelos DCGraM e VL-DCGraM para mapa logístico ternário.

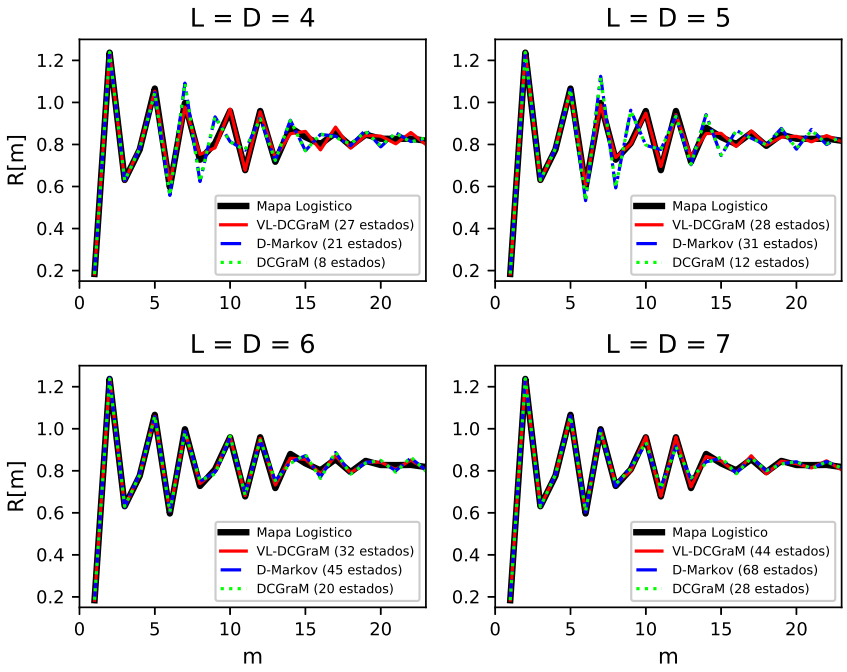
	DCGraM		VL-DCGraM	
D/L	inicial	pós-min.	inicial	pós-min.
4	21	8	50	27
5	31	12	63	28
6	45	20	73	32
7	68	28	92	44
8	104	28	116	46
9	156	28	131	46
10	230	44	154	46
11	336	47	174	46

Fonte: O autor (2019)

4.2.2 Canal Binário com Desvanecimento

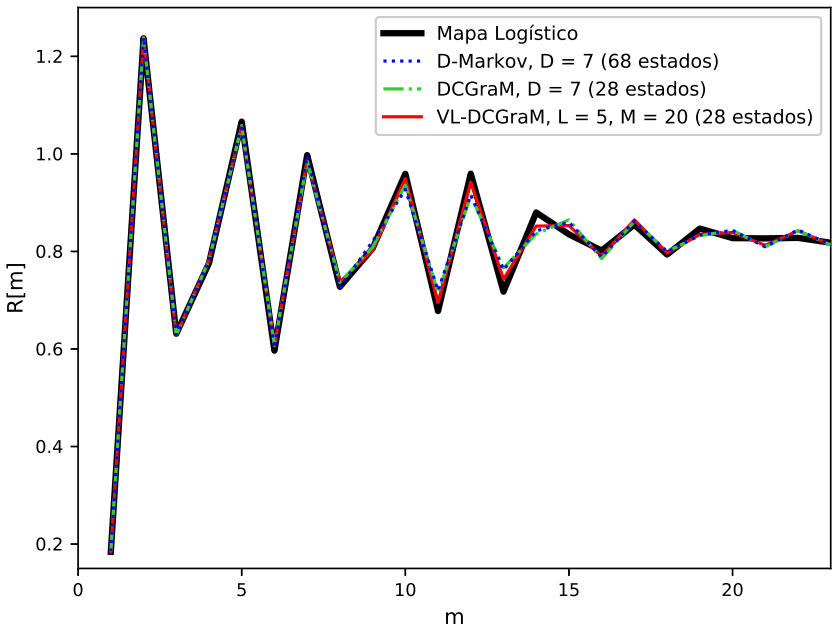
Considere um canal com desvanecimento discreto (DFC, *discrete fading channel*) com entrada binária e saída binária, com um processo de entrada binária i.i.d. $\{X_k\}_{k=1}^{\infty}$, $X_k \in \{0,1\}$, e saída binária $\{Y_k\}_{k=1}^{\infty}$, $Y_k \in \{0,1\}$, representado na Figura 10.

Figura 8 – Curvas da função autocorrelação $R[m]$ versus m da sequência gerada por modelos D-Markov, DCGraM e VL-DCGraM comparada com a sequência original S do mapa logístico ternário para D e L de 4 a 7.



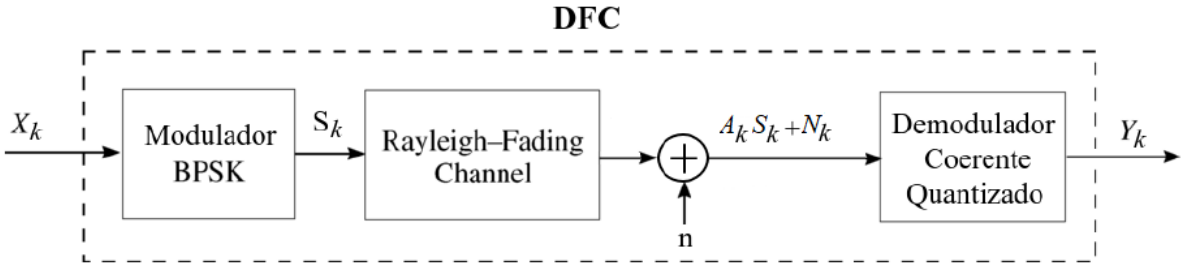
Fonte: O autor (2019)

Figura 9 – Curva da função autocorrelação $R[m]$ versus m da sequência gerada por modelos D-Markov e DCGraM ($D = 7$) e VL-DCGraM ($L = 5, M = 20$) comparada com a sequência original S do mapa logístico ternário.



Fonte: O autor (2019)

Figura 10 – Representação esquemática do Canal Binário com Desvanecimento.



Fonte: O autor (2019)

O DFC é composto por um modulador BPSK, um canal de desvanecimento plano Rayleigh correlacionado no tempo com AWGN e um demodulador coerente quantizado. O símbolo recebido no k-ésimo intervalo de sinalização é escrito como

$$R_k = \sqrt{E_s} A_k S_k + N_k, \qquad k = 1, 2, \cdots$$

em que $\{S_k\} = \{(2X_k - 1)\}$, E_s é a energia do sinal transmitido e $\{N_k\}$ é uma sequência de variáveis aleatórias Gaussianas de média zero independentes e identicamente distribuídas com variância $N_0/2$. A relação sinal-ruído (SNR) é definida como E_s/N_0. Além disso, $\{A_k\}$ é o processo de desvanecimento com $A_k = |G_k|$, em que $\{G_k\}$ é um processo Gaussiano complexo com média zero, variância unitária e função autocorrelação de Clarke (CLARKE, 1968) $R[m] = J_0(2\pi f_D T|m|)$, em que $J_0(x)$ é a função de Bessel de ordem zero do primeiro tipo e $f_D T$ é a máxima frequência Doppler normalizada. A variável aleatória A_k possui função densidade de probabilidade com segundo momento unitário, $p_A(a) = 2ae^{-a^2}$, para $a > 0$. O símbolo de saída é $Y_k = 0$ se $R_k \leq 0$, ou $Y_k = 1$ se $R_k > 0$.

Os símbolos de entrada e saída são expressos em termos de um símbolo de ruído $Z_k \in \{0, 1\}$ com $Y_k = X_k \oplus Z_k$, em que $\oplus$ denota adição módulo 2 e os símbolos X_k e Z_k são estatisticamente independentes. Para um determinado DFC especificado pelos parâmetros $f_D T$ e SNR, uma sequência de ruído binário $\{Z_k\}$ do DFC de comprimento $N = 5 \times 10^7$ é gerada por simulação e os parâmetros dos modelos D-Markov, DCGraM e do VL-DCGraM são estimados usando os algoritmos apresentados nos Capítulos 2 e 3.

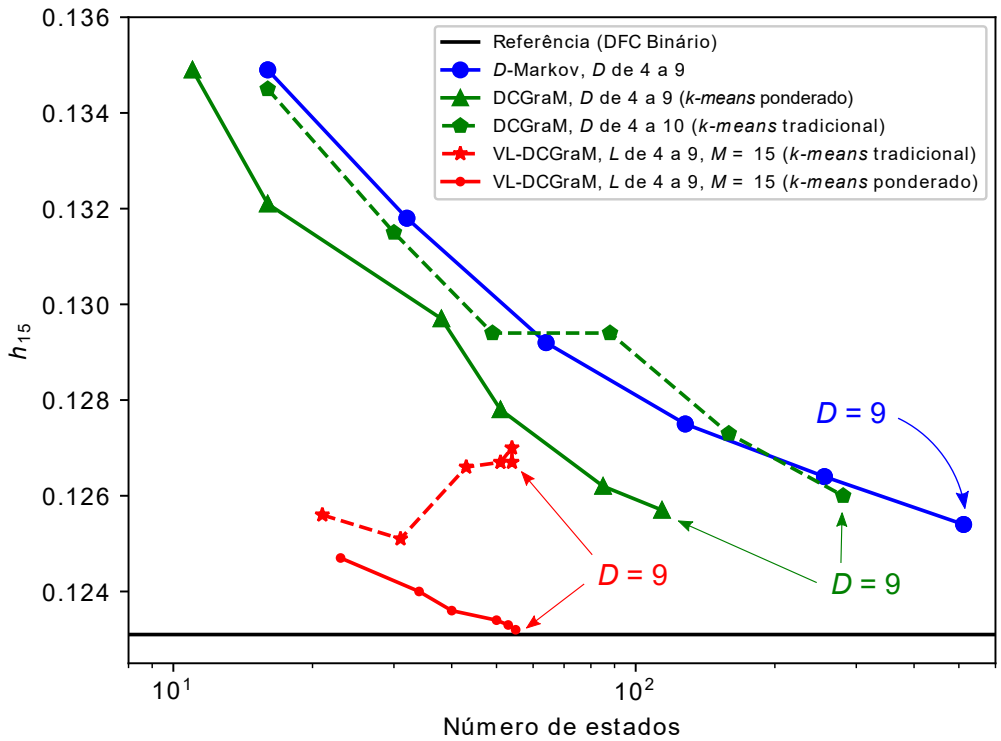
Os resultados a seguir são para um canal binário com desvanecimento com $f_D T = 0,005$ e SNR = 10 dB modelado com D-Markov e DCGraM para D variando de 4 a 9, além do modelo VL-DCGraM com L variando de 4 a 9 e $M = 15$. Testes realizados mostraram que os melhores resultados obtidos ocorreram para os parâmetros $T = 0,05$ e $\alpha = 2$. Os modelos DCGraM e VL-DCGraM utilizam o *k-means* ponderado com $K = 5$.

A Figura 11 mostra h_ℓ versus o número de estados para $\ell = 15$ para PFSAs gerados utilizando os modelos D-Markov, DCGraM e VL-DCGraM, apresentando ainda para estes dois últimos os resultados obtidos utilizando-se os algoritmos *k-means* tradicional e sua versão

ponderada proposta na Seção 3.3.1. É possível notar que a curva associada ao emprego do *k-means* ponderado apresenta melhores resultados em comparação àquela associada ao uso do *k-means* tradicional tanto para o modelo DCGraM quanto para o VL-DCGraM, sobretudo para este último, cujo comportamento da curva para o caso ponderado tende ao valor de referência com o aumento de L, fato que não ocorre quando da utilização do *k-means* tradicional. Esta observação reforçou a escolha do algoritmo *k-means* ponderado para etapa de clusterização nos modelos DCGraM e VL-DCGraM apresentados neste trabalho. Assim, apenas o *k-means* ponderado é considerado nas análises seguintes.

Prosseguindo a análise, observamos na Figura 11 que os resultados para entropia condicional obtidos para o modelo VL-DCGraM apresentam melhores resultados em comparação ao D-Markov e DCGraM, sendo o único modelo, com $L = 9$ e $M = 15$, a atingir o valor de entropia h_{15} do sistema original. Conclusão semelhante da superioridade do VL-DCGraM pode ser observada na Figura 12, que contém o resultado da divergência de Kullback-Leibler versus o número de estados para $\ell = 15$. Para um dado valor de divergência, o modelo VL-DCGraM produz um PFSA com menor número de estados quando comparados aos dos outros dois modelos.

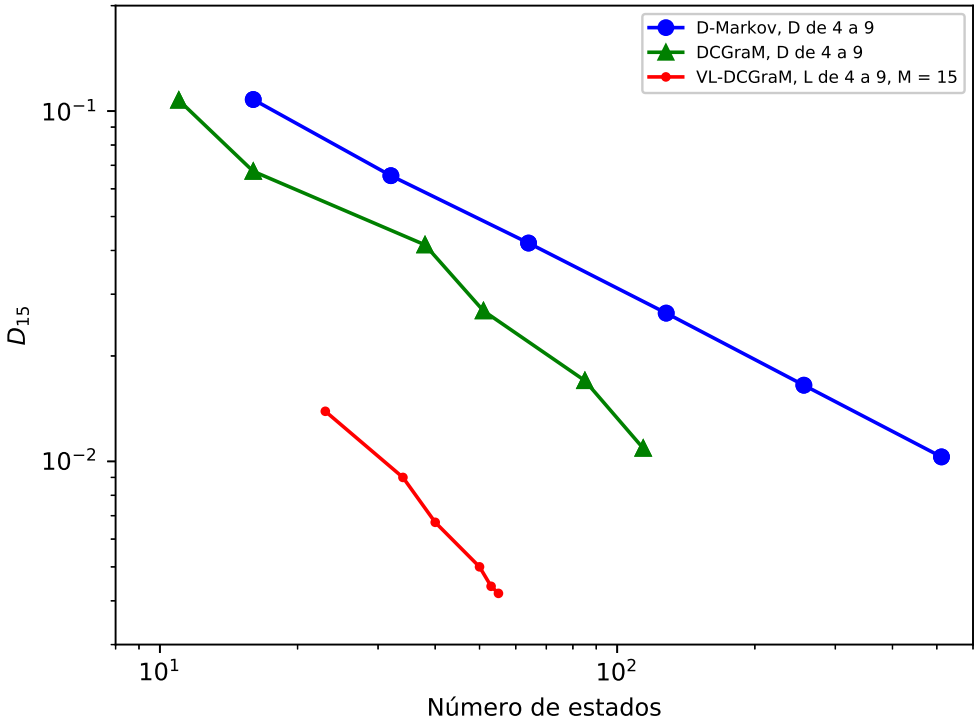
Figura 11 – Entropia condicional h_{15} versus número de estados da sequência gerada pelos modelos D-Markov e DCGraM com D de 4 a 9 e VL-DCGraM com L de 4 a 9 e M = 15 para o DFC binário com $f_D T = 0,005$ e SNR = 10 dB.



Fonte: O autor (2019)

A máquina inicial do modelo DCGraM, a D-Markov com $D = 9$, é constituída com

Figura 12 – Divergência de Kullback-Leibler D_{15} versus o número de estados da sequência gerada pelos modelos D-Markov e DCGraM (D de 4 a 9) e VL-DCGraM (L de 4 a 9, M = 15) para o DFC binário com $f_D T = 0,005$ e SNR = 10 dB.



Fonte: O autor (2019)

estados rotulados com sequência de tamanho 9, enquanto o VL-DCGraM tem estados com rótulo de comprimento variável conforme a Tabela 3, que mostra o número de estados da máquina inicial para um dado comprimento.

Tabela 3 – Número de estados por comprimento do rótulo dos estados da máquina inicial para o modelo VL-DCGraM com $L = 9$ e $M = 15$ para o DFC binário.

comprimento	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nº de estados	0	0	1	3	7	15	17	17	17	1	1	1	1	1	2

Fonte: O autor (2019)

A vantagem de redução do número de estados iniciais gerados pelo modelo VL-DCGraM também ocorre na sua aplicação ao DFC binário. Isso pode ser verificado na da Tabela 4, ao compararmos o número de estados das máquinas inicial e após as etapas de clusterização e minimização para cada um dos modelos analisados. Por exemplo, a máquina VL-DCGraM com $L = 9$ e $M = 15$ possui 84 estados inicialmente e 55 estados após a minimização, enquanto que aquela para o modelo DCGraM com D = 9 apresenta 512 estados inicialmente e 114 estados após a minimização, uma redução de 51,75% desse número de estados da máquina VL-DCGraM quando comparada à máquina DCGraM.

Tabela 4 – Número de estados das máquinas inicial e após minimização para os modelos DCGraM e VL-DCGraM para o DFC binário.

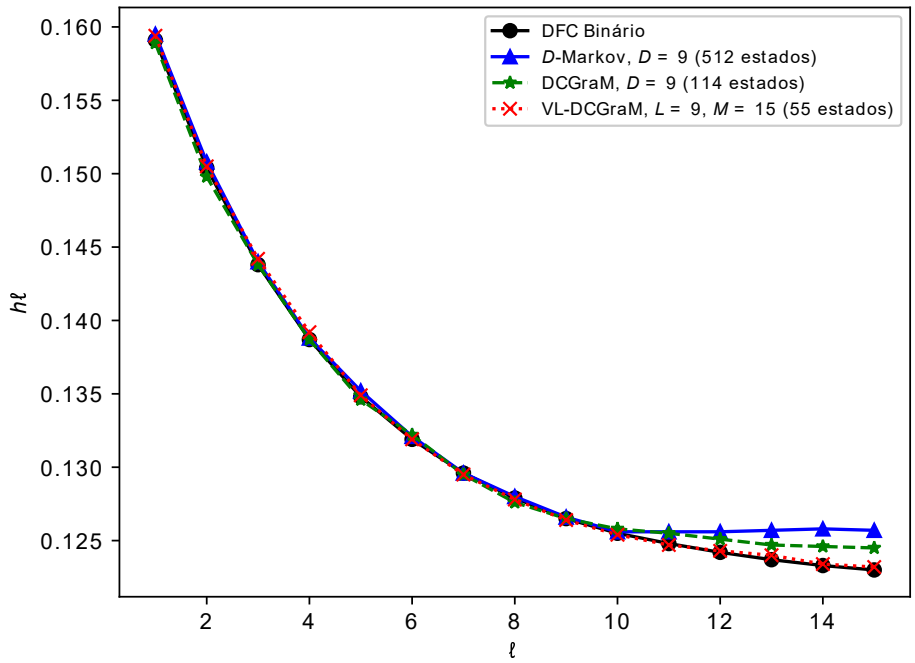
	DCGraM		VL-DCGraM	
D/L	inicial	pós-min.	inicial	pós-min.
4	16	11	26	23
5	32	16	36	34
6	64	38	50	40
7	128	51	64	50
8	256	85	76	53
9	512	114	84	55

Fonte: O autor (2019)

Na Figura 13 observa-se o comportamento de h_ℓ versus ℓ para os modelos D-Markov e DCGraM gerados para $D = 9$ e VL-DCGraM para $L = 9$ e $M = 15$. É possível notar que o modelo VL-DCGraM adere melhor à memória do sistema original quando comparado aos outros modelos. O valor de h_{15} é 0,1232, o que é uma boa aproximação para a taxa de entropia do DFC. O modelo VL-DCCGraM ($L = 9$ e $M = 15$) com 55 estados apresenta maior aderência à curva da entropia condicional do sistema original quando comparado aos outros modelos para ℓ e D maiores que 10.

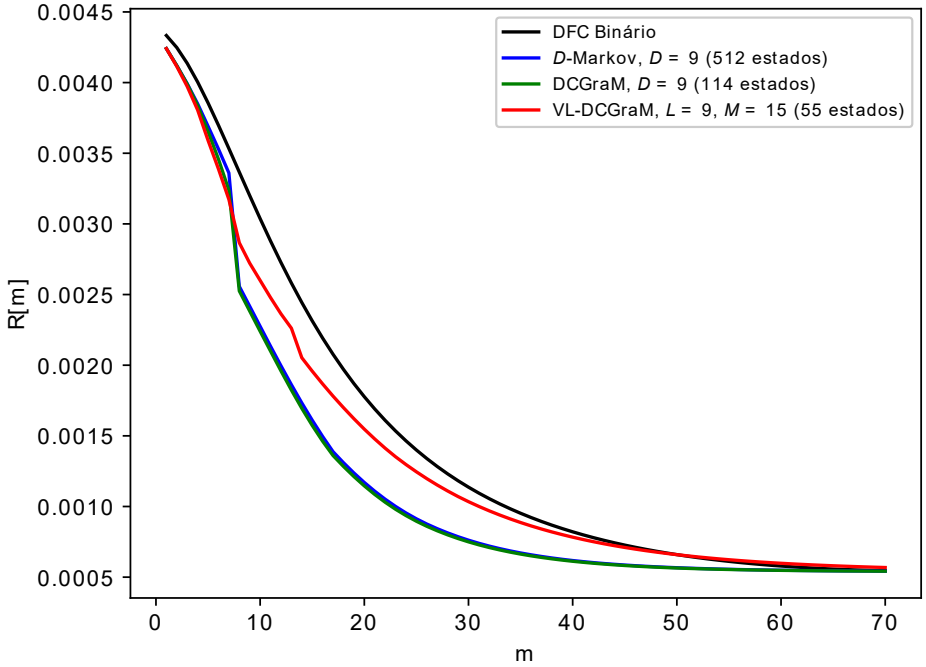
Finalmente, uma comparação da função autocorrelação é realizada na Figura 14 para valores selecionados de D e L. Para os modelos considerados, observa-se que o VL-DCGraM é o que tem maior proximidade com a curva do modelo original, além de apresentar um menor número de estados quando comparado aos modelos D-Markov e DCGraM.

Figura 13 – Entropia condicional h_ℓ versus ℓ associada aos modelos D-Markov e DCGraM para $D = 9$ e VL-DCGraM para $L = 9$ e $M = 15$ para o DFC binário com $f_D T = 0,005$ e SNR = 10 dB.



Fonte: O autor (2019)

Figura 14 – Autocorrelação $R[m]$ versus m da sequência gerada pela máquina D-Markov ($D = 9$), DCGraM ($D = 9$) e VL-DCGraM ($L = 9, M = 15$) para o DFC binário com $f_D T = 0,005$ e SNR = 10 dB.



Fonte: O autor (2019)

5 CONCLUSÃO

Neste trabalho foi apresentado o algoritmo VL-DCGraM para modelagem de sistemas dinâmicos discretos, utilizando PFSA para obter uma representação compacta. O algoritmo inicia analisando as estatísticas da sequência de saída de um sistema a ser modelado e usa técnicas de minimização de grafos para obter resultados mais compactos. Além disso, o VL-DCGraM também usa técnicas de aprendizado de máquina.

O algoritmo VL-DCGraM começa criando uma cadeia de Markov de comprimento variável a partir de uma árvore de contextos para um determinado L com base na sequência de saída de um sistema original. Em seguida, ele usa o algoritmo *k-means* ponderado, e finalmente aplica um algoritmo de minimização de grafo para obter um PFSA final.

O algoritmo VL-DCGraM foi aplicado a dois sistemas dinâmicos: o mapa logístico ternário e o canal binário com desvanecimento. Sendo esse algoritmo um método de refinamento sobre máquinas D-Markov usando técnicas de aprendizado de máquina, seus resultados foram comparados às máquinas D-Markov originais e ao modelo DCGraM.

Foi demonstrado que, para o mapa logístico, embora o PFSA gerado pelo VL-DCGraM seja menor que as máquinas D-Markov, a diferença não é tão perceptível quando comparado às máquinas DCGraM, embora essa diferença se torne mais evidente à medida que D e L aumentam. Ainda assim, o VL-DCGraM mostrou desempenho superior ao modelo D-Markov e equiparável às máquinas DCGraM, apresentando valores semelhantes de entropia condicional, divergência de Kullback-Leibler e função autocorrelação.

Para o canal de desvanecimento, o PFSA modelado pelo VL-DCGraM é notavelmente menor que as máquinas D-Markov e DCGraM, apresentando valores de entropia condicional, divergência de Kullback-Leibler e função autocorrelação mais próximos ao do sistema original do que os outros modelos.

5.1 TRABALHOS FUTUROS

Trabalhos futuros visam o aprimoramento do algoritmo apresentado nesta dissertação, abordando aplicações de interesse prático que possam ser beneficiadas com seu uso. Algumas melhorias de maior impacto para o algoritmo são:

- Explorar outros algoritmos de clusterização, como DBSCAN (ARLIA; COPPOLA, 2001) ou OPTICS (ANKERST et al., 1999) como substitutos do *k-means* adotado no VL-DCGraM.
- Aprimoramento do PFSA tradicional com a adoção de uma abordagem não determinística para as suas transições de estados com base na estatística da sequência original, ao invés de probabilidades fixas para essas transições.

- Modificação do algoritmo de criação de árvores de contextos, substituindo o uso da distância euclidiana entre *morphs* pelo divergente simétrico de Kullback-Leibler, visando caracterizar a divergência estatística entre *morphs* das sequências geradas e com isso observar o efeito sobre o conjunto de folhas obtido.
- Utilização dos modelos obtidos pelo algoritmo VL-DCGraM em aplicações como detecção de falhas nas quais máquinas com menor número de estados são capazes de indicar variações no comportamento estatístico da sequência de forma mais rápida e precisa (MUKHERJEE; RAY, 2014).

REFERÊNCIAS

ALTINEL, D.; KURT, G. K. Finite-state markov channel based modeling of rf energy harvesting systems. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 2, p. 1713–1725, 2017.

ANKERST, M. et al. OPTICS: ordering points to identify the clustering structure. In: ACM. **ACM Sigmod record**. [S.l.], 1999. v. 28, n. 2, p. 49–60.

ARLIA, D.; COPPOLA, M. Experiments in parallel clustering with DBSCAN. In: SPRINGER. **European Conference on Parallel Processing**. [S.l.], 2001. p. 326–331.

ARTHUR, D.; MANTHEY, B.; RÖGLIN, H. k-means has polynomial smoothed complexity. In: IEEE. **Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on**. [S.l.], 2009. p. 405–414.

BANDYOPADHYAY, S.; BHATTACHARYA, R. **Discrete and continuous simulation: theory and practice**. [S.l.]: CRC Press, 2014.

BARKER, M.; RAYENS, W. Partial least squares for discrimination. **Journal of Chemometrics: A Journal of the Chemometrics Society**, Wiley Online Library, v. 17, n. 3, p. 166–173, 2003.

BERSTEL, J. et al. Minimization of automata. **arXiv:1010.5318**, arXiv:1010.5318, 2010. Disponível em: <arXiv:1010.5318>.

BOHG, J. et al. Data-driven grasp synthesis—a survey. **IEEE Transactions on Robotics**, IEEE, v. 30, n. 2, p. 289–309, 2013.

BRADLEY, E.; KANTZ, H. Nonlinear time-series analysis revisited. **Chaos: An Interdisciplinary Journal of Nonlinear Science**, AIP Publishing, v. 25, n. 9, p. 097610, 2015.

BÜHLMANN, P. Model selection for variable length markov chains and tuning the context algorithm. **Annals of the Institute of Statistical Mathematics**, Springer, v. 52, n. 2, p. 287–315, 2000.

CECATI, C. A survey of fault diagnosis and fault-tolerant techniques—part ii: Fault diagnosis with knowledge-based and hybrid/active approaches. 2015.

CLARKE, R. A statistical theory of mobile-radio reception. **Bell Labs Technical Journal**, Wiley Online Library, v. 47, n. 6, p. 957–1000, 1968.

COVER, T. M.; THOMAS, J. A. **Elements of information theory**. [S.l.]: John Wiley & Sons, 2012.

DAW, C. S.; FINNEY, C. E. A.; TRACY, E. R. A review of symbolic analysis of experimental data. **Review of Scientific instruments**, AIP, v. 74, n. 2, p. 915–930, 2003.

DING, S. X. **Model-based fault diagnosis techniques: design schemes, algorithms, and tools**. [S.l.]: Springer Science & Business Media, 2008.

DUNTEMAN, G. H. **Principal components analysis**. [S.l.]: Sage, 1989.

FRANCH, D. K. **Dynamical System Modeling With Probabilistic Finite State Automata**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2017.

GAO, Z.; CECATI, C.; DING, S. X. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. **IEEE Transactions on Industrial Electronics**, IEEE, v. 62, n. 6, p. 3757–3767, 2015.

GRABEN, P. B. Estimating and improving the signal-to-noise ratio of time series by symbolic dynamics. **Physical Review E**, APS, v. 64, n. 5, p. 051104, 2001.

HAMERLY, G.; ELKAN, C. Alternatives to the k-means algorithm that find better clusterings. In: ACM. **Proceedings of the eleventh international conference on Information and knowledge management**. [S.l.], 2002. p. 600–607.

HOPCROFT, J. E. **Introduction to automata theory, languages, and computation**. [S.l.]: Pearson Education India, 2008.

HYVÄRINEN, A.; OJA, E. Independent component analysis: algorithms and applications. **Neural networks**, Elsevier, v. 13, n. 4-5, p. 411–430, 2000.

JIANG, Y. et al. A model-based hybrid ultrasonic gas flowmeter. **IEEE Sensors Journal**, IEEE, v. 18, n. 11, p. 4443–4452, 2018.

JIN, X. et al. Anomaly detection in nuclear power plants via symbolic dynamic filtering. **IEEE Transactions on Nuclear Science**, IEEE, v. 58, n. 1, p. 277–288, 2010.

KANUNGO, T. et al. An efficient k-means clustering algorithm: Analysis and implementation. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 24, n. 7, p. 881–892, 2002.

KIM, T. et al. An on-board model-based condition monitoring for lithium-ion batteries. **IEEE Transactions on Industry Applications**, IEEE, v. 55, n. 2, p. 1835–1843, 2018.

LALLEMENT, G. **Semigroups and combinatorial applications**. [S.l.]: John Wiley & Sons, Inc., 1979.

LI, F. et al. Anomaly detection in gas turbine fuel systems using a sequential symbolic method. **Energies**, Multidisciplinary Digital Publishing Institute, v. 10, n. 5, p. 724, 2017.

LI, Y.; RAY, A. Unsupervised symbolization of signal time series for extraction of the embedded information. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 19, n. 4, p. 148, 2017.

LIANG, G. et al. An unsupervised fault diagnosis method based on iterative multi-manifold spectral clustering. **IET Collaborative Intelligent Manufacturing**, IET, 2019.

LIN, S. et al. Finite-state markov modeling for high-speed railway fading channels. **IEEE Antennas and Wireless Propagation Letters**, IEEE, v. 14, p. 954–957, 2015.

LIND, D.; MARCUS, B. **An Introduction To Symbolic Dynamics and Codings**. Cambridge University Press, 1995. ISBN 978-0521559003. Disponível em: <http://ebooks.cambridge.org/ebook.jsf?bid=CBO9780511626302>.

LONG, Y. et al. Data-driven-based analog beam selection for hybrid beamforming under mm-wave channels. **IEEE Journal of Selected Topics in Signal Processing**, IEEE, v. 12, n. 2, p. 340–352, 2018.

LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. **Revista de Informática Teórica e Aplicada**, v. 14, n. 2, p. 43–67, 2007.

MACKAY, D. J. **Information theory, inference and learning algorithms**. [S.l.]: Cambridge university press, 2003.

MESKIN, N.; NADERI, E.; KHORASANI, K. A multiple model-based approach for fault diagnosis of jet engines. **IEEE Transactions on Control Systems Technology**, IEEE, v. 21, n. 1, p. 254–262, 2011.

MOORE, E. F. Gedanken-experiments on sequential machines. **Automata studies**, v. 34, p. 129–153, 1956.

MUKHERJEE, K.; RAY, A. State splitting and merging in probabilistic finite state automata for signal representation and analysis. **Signal Processing**, v. 104, p. 105–119, 2014. Disponível em: <http://dx.doi.org/10.1016/j.sigpro.2014.03.045>.

NGUYEN, V. et al. Model-based diagnosis and rul estimation of induction machines under interturn fault. **IEEE Transactions on Industry Applications**, IEEE, v. 53, n. 3, p. 2690–2701, 2017.

PIMENTEL, C.; ALAJAJI, F. Packet-based modeling of reed–solomon block-coded correlated fading channels via a markov finite queue model. **IEEE Transactions on Vehicular Technology**, IEEE, v. 58, n. 7, p. 3124–3136, 2009.

PIMENTEL, C.; BLAKE, I. F. Enumeration of markov chains and burst error statistics for finite state channel models. **IEEE Transactions on Vehicular Technology**, IEEE, v. 48, n. 2, p. 415–428, 1999.

RAY, A. Symbolic dynamic analysis of complex systems for anomaly detection. **Signal Processing**, Elsevier, v. 84, n. 7, p. 1115–1130, 2004.

REN, H.; YE, Z.; LI, Z. Anomaly detection based on a dynamic markov model. **Information Sciences**, Elsevier, v. 411, p. 52–65, 2017.

STROGATZ, S. **Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering**. [S.l.]: Westview Press, 2001.

VIDAL, E. et al. Probabilistic finite-state machines-part i. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 27, n. 7, p. 1013–1025, 2005.

VIDAL, E. et al. Probabilistic finite-state machines-part ii. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 27, n. 7, p. 1026–1039, 2005.

VIDAL, E. et al. Probabilistic finite-state machines - part I. **IEEE Transactions On Pattern Analysis And Machine Intelligence**, v. 27, n. 7, p. 1013–1025, July 2005. ISSN 0162-8828.

WANG, L.; ZHONG, Y.; MA, W. Gps-data-driven dynamic destination prediction for on-demand one-way carsharing system. **IET Intelligent Transport Systems**, IET, v. 12, n. 10, p. 1291–1299, 2018.

WEISS, M. A. Data structures and problem solving using java. In: ____. 2. ed. [S.l.]: Addison Wesley, 2002. cap. 18, p. 592–595.