COPPE/UFRJ

EXTRAÇÃO DE REGRAS *FUZZY* UTILIZANDO ANÁLISE ESPECTRAL E ALGORITMOS GENÉTICOS EM AMBIENTE COMPUTACIONAL DE ALTO DESEMPENHO

Vinícius da Fonseca Vieira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Civil, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Civil.

Orientador: Alexandre Gonçalves Evsukoff

Rio de Janeiro
Abril de 2009

EXTRAÇÃO DE REGRAS *FUZZY* UTILIZANDO ANÁLISE ESPECTRAL E ALGORITMOS GENÉTICOS EM AMBIENTE COMPUTACIONAL DE ALTO DESEMPENHO

Vinícius da Fonseca Vieira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

Prof. Alexandre Gonçalves Evsukoff, D.Sc.

Prof. Rodrigo Weber dos Santos, D.Sc.

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

Prof. Beatriz de Souza Leite Pires Lima, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2009

Vieira, Vinícius da Fonseca

Extração de Regras *Fuzzy* Utilizando Análise Espectral e Algoritmos Genéticos em Ambiente Computacional de Alto Desempenho/Vinícius da Fonseca Vieira. – Rio de Janeiro: UFRJ/COPPE, 2009.

XII, 86 p.: il.; 29, 7cm.

Orientador: Alexandre Gonçalves Evsukoff

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Civil, 2009.

Referências Bibliográficas: p. 79 – 86.

1. Classificador *Fuzzy*. 2. Computação de Alto Desempenho. 3. Algoritmo Genético Paralelo. 4. Análise Espectral Paralela. 5. Otimização de Estrutura. I. Evsukoff, Alexandre Gonçalves. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Civil. III. Título.

Agradecimentos

À minha mãe (Meire) e meu pai (Fernando) que sempre me incentivaram a buscar mais, sempre acreditaram em mim e sempre me ajudaram nas escolhas da vida.

Ao meu irmão Vítor pela grande amizade e fidelidade. Não é o sangue que o define como meu irmão.

À Carol, minha namorada, por tudo que representa na minha vida e por me ajudar sempre que eu preciso, quando eu peço e até quando eu nem peço. Além de uma namorada perfeita ainda me ajuda muito na computação. Sua presença faz o caminho ficar muito mais tranquilo.

À família da Carol, que sempre me fez sentir parte da família também, pelo carinho, amizade e confiança.

Ao Alexandre, meu orientador, pela amizade, pelos ensinamentos e pela paciência que me ajudaram a me encontrar na COPPE.

Ao Rodrigo, por ter aberto vários portas para mim e por ter me acolhido no Fisiocomp mesmo quando não estava mais na UFJF. Também por ter aceitado fazer parte da banca de avaliação. O considero um amigo e uma referência pessoal e profissional.

Aos amigos do Fisiocomp: Sachetto, Fernando, Caroline, Daves, Ricardo, Gustavo e outros, com quem passei bons momentos, dentro e fora do Fisiocomp.

Ao amigos Chuchu, Motoboy e Allan que fizeram minha ida para o Rio bem mais fácil.

Ao Custódio pela amizade, pelo incentivo à ida para a COPPE e por fazer essa ponte para mim.

Ao pessoal da sala da UFJF, que além de serem exemplo como profissionais e estudantes são amigos pra sempre.

Aos amigos de sempre por me fazerem rir até quando o mundo está acabando.

Ao Cristian pela amizade e por dividir os conhecimentos e as (muitas) preocupações durante o mestrado.

Aos Professores Nelson e Beatriz por aceitarem participar da banca de avaliação dessa dissertação e pelos ensinamentos durante o mestrado..

Ao CNPq pelo apoio financeiro que viabilizou este trabalho.

Ao Laboratório de Fisiologia Computacional e Computação de Alto Desempenho (Fisiocomp/UFJF) e ao Núcleo de Atendimento em Computação de Alto Desempenho (NACAD/UFRJ) por disponibilizarem recursos computacionais que viabilizaram este trabalho.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

EXTRAÇÃO DE REGRAS *FUZZY* UTILIZANDO ANÁLISE ESPECTRAL E ALGORITMOS GENÉTICOS EM AMBIENTE COMPUTACIONAL DE ALTO DESEMPENHO

Vinícius da Fonseca Vieira

Abril/2009

Orientador: Alexandre Gonçalves Evsukoff

Programa: Engenharia Civil

Este trabalho apresenta a implementação de um sistema para extração de regras *fuzzy* interpretáveis em um ambiente de computação de alto desempenho. O número de regras do modelo *fuzzy* é estimado através de análise espectral, que consiste na análise de autovalores de uma matriz baseada na similaridade entre os dados de entrada. Com o objetivo de melhorar a precisão de classificação do modelo, os pesos das regras são computados por um problema de otimização quadrática limitado e um algoritmo genético é utilizado para a seleção de estrutura que define simultaneamente o melhor subconjunto de variáveis e o número de conjuntos *fuzzy* na partição das variáveis selecionadas.

Foram implementadas duas soluções paralelas para o classificador *fuzzy*. Uma delas paraleliza apenas o algoritmo genético e outra paraleliza, além do algoritmo genético, a análise espectral, caracterizando uma implementação em dois níveis. O modelo resultante é avaliado em um conjunto de bases de dados e os resultados mostram que tal abordagem produz modelos *fuzzy* precisos e compactos. Do ponto de vista do paralelismo, os resultados mostram que ambas as abordagens permitem uma redução considerável do tempo de processamento global do método.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EXTRACTION OF FUZZY RULES USING SPECTRAL ANALYSIS AND GENETIC ALGORITHMS IN HIGH PERFORMANCE COMPUTING ENVIRONMENT

Vinícius da Fonseca Vieira

April/2009

Advisor: Alexandre Gonçalves Evsukoff

Department: Civil Engineering

This work presents an implementation of an interpretable fuzzy rule-based classifier in high performance environment. The number of rules in the fuzzy model is estimated by spectral analysis, which consists in the eigenstructure analysis of a matrix based on the similarity among the input data. In order to improve the model classification accuracy, the rule's weights are computed by a bounded quadratic optimization problem and a genetic algorithm is used for structure selection which defines simultaneously the best subset of variables and the number of fuzzy sets in the partition of the selected variables.

Two parallel solutions for the fuzzy classifier were implemented. One of them parallelizes only the genetic algorithm ant the other parallelizes, besides the genetic algorithm, the spectral analysis, characterizing a two-level approach. The resulting model is evaluated by a set of data bases and the results show that such approach produces accurate and compact fuzzy models. In terms of parallelism, the results show that both approaches allow a considerable reduction in the processing global time.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com a crescente disponibilidade de dados referentes a observações sobre processos nas mais diversas áreas do conhecimento, houve também, naturalmente, um crescente estudo sobre o desenvolvimento de métodos que possibilitem a extração de informações úteis a partir dessas observações. Através da descoberta de padrões nos dados coletados, abre-se novas possibilidades para o estudo de fenômenos tão complexos que não há modelos matemáticos que os descrevam, ou que a não-linearidade desses modelos faz com que a análise seja extremamente dificultada.

Através da mineração de dados é possível extrair conhecimento a partir da análise de observações de entrada e saída para um determinado fenômeno, possibilitando que um sistema computacional crie modelos que simulem determinada tarefa e que sejam capazes de predizer, com eficiência, o valor de saída para uma determinada entrada.

As duas formas mais tradicionais de análise de dados para extração de modelos que descrevam um fenômeno são [1]:

- Classificação: a partir da observação de um conjunto de variáveis de entrada, deve-se predizer apenas uma variável de saída de valor nominal, que se refere a uma classe que categoriza o valor de saída. Assim, o objetivo da classificação é atribuir uma categoria a uma observação cuja classe ainda é desconhecida.

- Regressão: a partir da observação de um conjunto de variáveis de entrada, deve-se predizer um conjunto variáveis de saída numéricas e de valor real. Geralmente uma aproximação independente $\widehat{y} = \widehat{f}(x)$ é construída para cada variável de saída.

Diversos métodos para extração de conhecimento de bases de dados podem ser encontrados na literatura, cada qual com uma série de vantagens e desvantagens peculiares, tais como: classificadores Bayesianos, redes neurais, máquinas de vetores de suporte, árvores de decisão e sistemas *fuzzy*.

Atualmente, a mineração de dados é uma área bastante madura e consolidada e por isso tem sido aplicada, não só em ambientes acadêmicos mas também em ambientes corporativos. Hoje em dia a mineração de dados é utilizada tanto em aplicações voltadas para o governo [2] quanto para empresas privadas [3].

Com a ampliação da utilização da mineração de dados, tem havido também uma crescente necessidade de se extrair informações de bases de dados cada vez maiores, exigindo o desenvolvimento de métodos capazes de lidar com grandes massas de dados e de ambientes computacionais de *software* e *hardware* que suportem a execução dessas tarefas.

Em problemas cuja solução demanda grande poder computacional, como é o caso dos problmas de extração de conhecimento de grandes bases de dados, uma abordagem que é bastante utilizada é a computação paralela, isto é, a divisão da computação entre múltiplos processadores. Dessa forma, pode-se tirar proveito do poder computacional de vários computadores simultaneamente e a aplicação tem sua execução dividida de forma transparente, como se em um nível mais alto apenas um processador estivesse executando tal tarefa. Entretanto, por muito tempo o uso dessa abordagem esteve limitado a ambientes acadêmicos, já que computadores distribuídos não eram de fácil aquisição para usuários comuns ou até para empresas.

Porém, atualmente tem havido uma crescente popularização de computadores *multicore* e uma constante redução no custo de *clusters* de computadores pessoais. Por isso tem havido uma grande disseminação no desenvolvimento de *softwares* que sejam capazes de aproveitar os benefícios da computação paralela. Além da demanda previamente existente por *softwares* paralelos, a expansão no desenvolvimento de aplicações segundo essa abordagem tem sido impulsionada por iniciativas como o *Universal Parallel Computing Research Center* da Microsoft e Intel, que tem como objetivo acelerar os benefícios da computação paralela para consumidores e empresas.

Esse trabalho tem como objetivo desenvolver uma implementação robusta e uma

solução paralela para a *Fuzzy Symbolic Machine* (FSM), que teve sua base proposta por Evsukoff em [4] e estendida por Evsukoff et al. em [5].

Esse trabalho compreende a uma implementação e à análise da *Fuzzy Symbolic Machine* (FSM), um método de classificação baseado em regras *fuzzy* que teve sua base proposta por Evsukoff em [4] e estendida por Evsukoff et al. em [5]. Nesse trabalho, a FSM lida apenas com problemas de classificação, entretanto o procedimento poderia ser adaptado para problemas de regressão.

Métodos baseados em regras *fuzzy* para análise e modelagem de dados são reconhecidos por permitirem a representação da dependência através de variáveis numéricas pela relação entre seus conceitos linguísticos. Esses métodos permitem que se construa modelos com boa capacidade preditiva mas também levando em consideração a interpretabilidade. Entretanto, a medida que a complexidade de um problema cresce, interpretabilidade e precisão tornam-se características incompatíveis [6].

A FSM é um sistema *fuzzy* para classificação composto de três tarefas principais: seleção de estrutura, identificação da base de regras e estimação de parâmetros.

A tarefa de seleção de estrutura tem como objetivo definir as variáveis que serão consideradas pelo modelo *fuzzy* e o número de partições *fuzzy* no domínio de cada uma das variáveis. Tal tarefa é realizada por um algoritmo genético (AG), um método de otimização iterativo inspirado na teoria da evolução das espécies que é baseado na avaliação da qualidade de um conjunto de soluções candidatas.

Em modelos baseados em regras *fuzzy* interpretáveis, o conjunto de regras realiza uma transformação do espaço original (dados) ao espaço de características tal que os dados de treinamento disponíveis podem ser interpretados como um conjunto de regras simbólicas. A determinação do conjunto de regras simbólicas é feita na tarefa de identificação da base de regras. Nessa etapa lida-se com um dos problemas principais na extração de modelos *fuzzy*, que é a determinação de um número adequado de regras, o qual é relacionado à complexidade do problema e à capacidade do modelo resultante. A medida em que o problema se torna mais complexo, mais regras são necessárias para descrever adequadamente os dados e ainda alcançar uma boa precisão. Na FSM o número de regras *fuzzy* relevantes do modelo é computado através da análise espectral. Métodos espectrais são baseados na teoria dos grafos

e fornecem ferramentas para a análise de dados a partir da análise de autovalores. Um algoritmo de agrupamento calcula a posição das regras.

A tarefa de estimação de parâmetros tem como objetivo otimizar os pesos de cada uma das regras presentes no modelo *fuzzy*.

Conforme constatado em [5], a FSM apresenta como uma grande desvantagem o alto tempo de processamento necessário para a geração de modelos *fuzzy*, o que dificulta a geração de modelos para bases de dados com grandes números de registros.

O tratamento do alto tempo de processamento da FSM é o principal foco desse trabalho e para que esse problema fosse contornado foram tomadas algumas medidas.

Foram implementadas duas soluções paralelas para a FSM. Em uma delas, diversos processadores são usados para a execução simultânea do algoritmo genético onde cada processador fica responsável pela avaliação de uma parte das soluções candidatas. Na outra implementação, os processadores são divididos em grupos. Uma solução candidata é então avaliada por um grupo de processadores (e não apenas por um processador). Assim, a paralelização ocorre em dois níveis: em um nível mais alto, há a paralelização do algoritmo genético e em um nível mais baixo acontece a paralelização da análise espectral, realizada pelos processadores de um mesmo grupo.

Com o objetivo de reduzir o tempo de processamento da FSM, além da paralelização, foi implementado também um esquema para armazenar as soluções intermediárias do AG e utilizá-las de forma a reduzir o volume de processamento necessário para a solução da FSM.

Com a implementação desenvolvida nesse trabalho tem-se como objetivo que a FSM seja capaz de trabalhar com bases de dados com grandes números de registros, permitindo a geração de modelos *fuzzy* para problemas de dimensões compatíveis com vários problemas encontrados em aplicações reais.

Esse trabalho está dividido da seguinte forma. O Capítulo 2 apresenta uma visão geral sobre conceitos básicos relativos a sistemas para extração de conhecimento através de regras *fuzzy*. Apresenta também uma revisão sobre modelagem simbólica *fuzzy*, utilizada nesse trabalho para a construção da. O Capítulo 3 apresenta uma revisão sobre aspectos importantes no desenvolvimento de sistemas baseados em regras *fuzzy*, como seleção de estrutura e otimização de pesos de regras. Dá-se ênfase

ao algoritmo de indução de regras e aos métodos utilizados no desenvolvimento da FSM. No Capítulo 3 é também apresentada uma revisão sobre análise espectral e é explicado como a análise espectral foi utilizada na FSM para estimação do número de regras. O Capítulo 4 apresenta uma revisão sobre a utilização de vários processadores para diminuição do tempo de processamento global de uma aplicação. São também apresentados modelos para paralelização de algoritmos genéticos e soluções para a paralelização de problemas de autovalor. O Capítulo 5 apresenta os resultados de uma série de experimentos realizados com a FSM onde destaca-se a interpretabilidade dos modelos gerados, a eficiência de tais modelos em respeito a precisão e número de regras e a eficiência do paralelismo implementado. O Capítulo 6 apresenta algumas considerações finais e conclusões desse trabalho.

Capítulo 2

Sistemas de Inferência Baseados em Regras *Fuzzy*

2.1 Representação de Conhecimento Através de Regras *Fuzzy*

Métodos de classificação são frequentemente empregados para a extração de informações úteis presentes em bases de dados, que podem ser utilizadas para auxiliar a tomada de decisão em diversas áreas. A classificação permite a extração de modelos que permitem um melhor conhecimento dos dados através de sua análise [1]. Assim, a partir da observação de exemplos anteriores, os sistemas de classificação são capazes de predizer a classe de um novo registro, cuja classe ainda não é conhecida [7].

As redes neurais se tornaram muito populares como método de classificação a partir de dados e uma de suas principais vantagens é a boa precisão numérica, possibilitada por funções de ativação não lineares que permitem que uma rede neural funcione como um aproximador universal [8]. Entretanto, um dos principais pontos negativos desse tipo de método é o seu comportamento tipo "caixa preta" [9]. Apesar de proverem um modelo numérico, os coeficientes de uma rede neural não têm um significado direto para especialistas na área em que é aplicada.

Muitas vezes a boa taxa de precisão de um método de classificação é suficiente em um determinado problema. Entretanto, muitas vezes deseja-se que o modelo obtido

represente a base de dados de forma que um humano especialista seja capaz de compreender tal conhecimento. Para isso, métodos que representam o conhecimento através de regras são muito eficazes e têm sido objeto de vários estudos e diversos trabalhos têm sido publicados nesse sentido [5] [4] [10] [11] [12] [13] [14] [15].

Uma regra é uma implicação lógica composta de uma parte chamada antecedente - que descreve as condições da regra - e outra parte chamada consequente que realiza a conclusão da informação. A parte antecedente é formada por operadores lógicos agrupados (e, ou, não etc.), gerando uma expressão lógica. Se essa expressão for verdadeira, isto é, se as condições da regra forem satisfeitas, então a parte consequente determinará uma conclusão, a partir da qual pode-se tomar alguma decisão.

Um sistema que utiliza lógica *fuzzy* é capaz de expressar imprecisões e ambiguidades inerentes à fenômenos naturais e ao pensamento humano [16] através de regras.

A lógica *fuzzy* foi proposta na década de 60 com o objetivo de representar numericamente informações vagas e imprecisas. Segundo a lógica fuzzy, uma proposição pode ter graus de verdade em um intervalo $[0, 1]$, diferentemente da lógica booleana, onde uma proposição é apenas verdadeira ou falsa. Um determinado valor x tem seu grau de pertinência a um conjunto A dado por uma função de pertinência μ_A que pode assumir diferentes formas, e cujo valor indica o grau de compatibilidade entre x e o conjunto A. A combinação entre conjuntos fuzzy é feita através de operadores especiais como as t-normas e as t-conormas (que em um raciocínio aproximado definem as conjunções e as disjunções, respectivamente).

Em grande parte dos sistemas *fuzzy*, as regras do tipo se-então são derivadas de informações linguísticas, obtidas a partir do conhecimento de especialistas humanos [17]. Entretanto, esse tipo de abordagem possui dois pontos fracos, como observado por Wang & Mendel em [18]:

1. É completamente dependente do problema, isto é, um método que funciona muito bem para um problema pode não funcionar para outro;

2. Não existe um *framework* comum para a modelagem e representação dos diferentes aspectos das estratégias de solução desse tipo de sistemas, o que dificulta a análise teórica dessa abordagem.

Assim, Wang & Mendel concluem que em um cenário onde deseja-se simular um fenômeno complexo e não há um modelo matemático para ele, ou o modelo matemático é fortemente não-linear, é inviável a utilização de um controlador humano. Nesse caso há uma grande necessidade da criação de ferramentas para a extração automática de modelos a partir de dados.

O trabalho de Takagi & Sugeno [19] foi um dos primeiros a propor um sistema de inferência *fuzzy* para extração automática de regras, como as regras dos especialistas, a partir de dados.

Atualmente, sistemas *fuzzy* automáticos são conhecidos por além de serem capazes de manipular conceitos linguísticos reais, serem também aproximadores não-lineares universais de mapeamentos de vetores de entrada não-*fuzzy* em valores de saída não-*fuzzy* [18].

Duas das principais medidas para avaliação da qualidade de métodos de aprendizado de máquina são a precisão (capacidade do método predizer corretamente uma saída a partir de uma entrada) e a interpretabilidade (capacidade do método prover a compreensibilidade do fenômeno modelado) [1]. Apesar de muitas vezes a precisão dos modelos gerados ser muito mais enfatizada, o estudo da interpretabilidade tem ganho muita atenção atualmente [10] [14] [15].

Essencialmente, os sistemas *fuzzy* possuem duas características que definem a qualidade dos seus modelos *fuzzy* resultantes [6]:

- **Interpretabilidade**: Refere-se à capacidade do modelo *fuzzy* expressar o comportamento do sistema de uma maneira compreensível. Vários fatores interferem nessa propriedade, como a estrutura do modelo, o número de variáveis de entrada, o número de regras *fuzzy*, e o formato dos conjuntos *fuzzy*. Essa é uma propriedade subjetiva que engloba diferentes critérios como consistência e transparência.

- **Precisão**: Refere-se à capacidade do modelo *fuzzy* de representar fielmente o sistema modelado. Quanto mais similar é a resposta dada pelo modelo *fuzzy* da resposta dada pelo sistema real, maior é a sua precisão.

Entretanto, a medida que a complexidade do problema cresce, interpretabilidade e precisão se tornam características incompatíveis. Deve-se então encontrar um

limite onde essas duas características ficam balanceadas e esse é o principal ponto a ser levado em consideração no projeto de um sistema baseado em regras *fuzzy* interpretáveis, como será visto no Capítulo 3.

2.2 Modelagem Simbólica Fuzzy

Esta seção apresenta a modelagem simbólica *fuzzy*, utilizada nesse trabalho para o desenvolvimento do sistema de classificação *fuzzy*. Primeiramente será feita uma breve descrição das características do problema de modelagem simbólica *fuzzy*. Em seguida são apresentados alguns conceitos importantes no contexto de sistemas *fuzzy*. Após, são descritas as três etapas básicas que compõem o desenvolvimento do sistema desenvolvido nesse trabalho.

2.2.1 Descrição do Problema

Considera-se um conjunto de dados composto de N amostras de entrada e saída observadas $T = \{(x(t), y(t)), t = 1 \dots N\}$, onde $x \in \mathbf{R}^p$ é o vetor de variáveis de entrada e $y \in \mathbf{R}^q$ é o vetor de variáveis de saída. Considera-se também um conjunto de símbolos de entrada $A = \{A_i, i = 1 \dots n\}$ e um conjunto de símbolos de saída $B = \{B_j, j = 1 \dots m\}$, dos quais cada elemento pode ser associado a um conceito linguístico (ou a um conjunto de conceitos linguísticos conectados por operadores lógicos) que referem-se qualitativamente aos valores das variáveis de entrada e saída respectivamente.

Supõe-se a existência de uma relação $y = f(x)$ entre as variáveis de entrada e saída e o propósito principal da tarefa de aprendizado de máquina é computar uma aproximação $\widehat{y} = \widehat{f}(x)$, representada por um conjunto de regras $A_i \rightarrow B_j$ lidas como:

$$\text{se } \mathbf{x} \text{ é } A_i \text{ então } \mathbf{y} \text{ é } B_j. \tag{2.1}$$

A abordagem simbólica *fuzzy* é baseada em uma abstração do problema de aprendizado de máquina em três camadas como é ilustrado na Figura 2.1.

A camada inferior é a camada de observação, representada por um conjunto de amostras de dados observados de entrada e saída coletado da aplicação. A camada

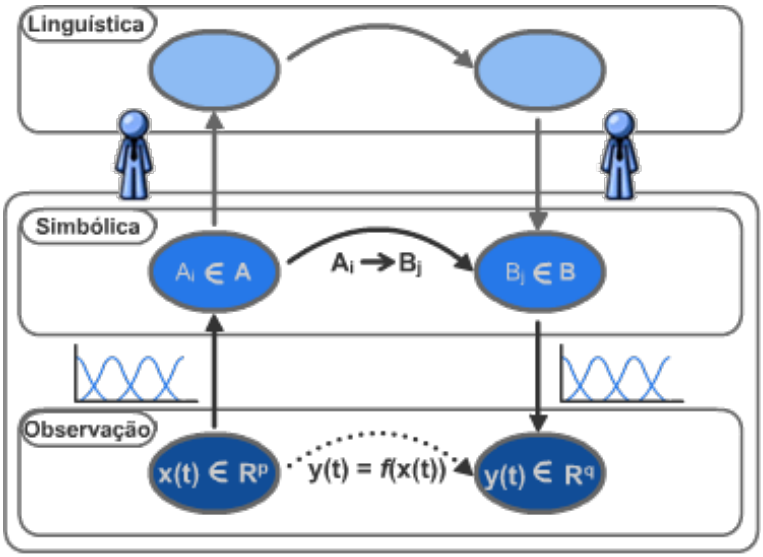

Figura 2.1: As três camadas de abstração na análise de dados.

simbólica é implementada por regras simbólicas interpretáveis $A_i \rightarrow B_j$, que relacionam variáveis de entrada e saída. A terceira camada é a camada linguística, onde assume-se que humanos especialistas sejam capazes de interpretar e compreender os símbolos e as relações definidas pelas regras simbólicas

Esse trabalho lida com a *Fuzzy Symbolic Machine* (FSM) proposta em [5] que realiza a modelagem dos dados de entrada e saída consistentemente de acordo com relações simbólicas de entrada e saída expressas por regras. Todavia, a busca pelo melhor modelo deve evitar a corrupção da interpretabilidade do modelo, que deve permanecer compreensível.

2.2.2 Conceitos Preliminares Sobre Sistemas *Fuzzy*

Em uma variável linguística, o significado de um símbolo é definido por um conjunto *fuzzy* através de sua função de pertinência. Um símbolo significativo pode então ser definido como qualquer símbolo A_i quando seu significado é definido pelo conjunto *fuzzy* $A_i = \{(x, \mu_{A_i}(x)), x \in X\}$, e pode ser atribuído a um conceito linguístico compreensível.

Outras restrições podem ser impostas à função de pertinência *fuzzy* para auxiliar a interpretabilidade dos conjuntos *fuzzy* [10] [20]. Isso é permitido pelo conceito de partição *fuzzy*. No escopo desse trabalho, um conjunto de símbolos significativos

$A = \{A_i, i = \ldots n\}$ é uma partição *fuzzy* se todas as funções de pertinência dos correspondentes conjuntos *fuzzy* somam uma unidade:

$$\sum_i \mu_{A_i}(x) = 1, \forall x \in X. \tag{2.2}$$

A partição *fuzzy* computa uma transformação não-linear do domínio da variável base, tal que uma amostra observada $x(t) \in X$ é mapeada em um vetor de características $u(t) \in [0,1]^n$, cujos componentes são calculados como:

$$u(t) = [u_i(t), \ldots, u_n(t)] = [\mu_{A_1}(x(t)), \ldots, \mu_{A_n}(x(t))]. \tag{2.3}$$

O vetor de características fornece uma descrição simbólica de um valor numérico que pode ser visto como um conjunto *fuzzy* definido em um universo simbólico. Um conjunto *fuzzy* $U_t = \{(A_i, \mu_{U_t}), A_i \in A\}$ é um símbolo *fuzzy* se todo símbolo $A_i \in A$ é um símbolo significativo e U_t é a descrição simbólica de um valor de uma variável observado $x(t) \in X$, tal que:

$$\mu_{U_t}(A_i) = \mu_{A_i}(x(t)). \tag{2.4}$$

Como grande parte dos sistemas *fuzzy*, a FSM computa uma aproximação $\widehat{y} = \widehat{f}(x)$ em três passos, implementados como os operadores de Fuzzificação, Inferência e Deffuzificação, que serão apresentados nas próximas seções.

2.2.3 Fuzzificação

Fuzzificação é o mapeamento do domínio de variáveis de entrada para um espaço de característica n-dimensional $F : \mathbf{R}^p \rightarrow [0,1]^n$, onde p é a dimensão do domínio de variáveis de entrada e n é a dimensão do espaço de características, que também é o número de regras na FSM. A fuzzificação é parametrizada pelo conjunto de vetores protótipos $\alpha = \{\alpha_k, k = 1 \ldots p\}$ e a matriz de regras *fuzzy* $\Xi = \{\xi_{ik}, i = 1 \ldots n, k = 1 \ldots p\}$, computada pelo algoritmo de indução de regras.

Cada variável do problema tem seu domínio dividido em K conjuntos *fuzzy*. Uma amostra observada $x(t) \in \mathbf{R}^p$ é mapeada em um vetor *fuzzy* $u(t) \in [0,1]^n$ através da Equação 2.3. Cada regra na FSM é expressada em termos do símbolo A_i que representa uma região no domínio multidimensional no domínio das variáveis de entrada.

A premissa de regra $\mathbf{x}(t)$ é A_i é computada como a conjunção de predicados *fuzzy* elementares $\mathbf{x}(t)$ é A_k^i, onde o termo A_k^i representa o conjunto *fuzzy* definido no domínio da variável x_k que deve ser considerada como um componente do símbolo multidimensional A_i. Os componentes do vetor de características são computados pela combinação das funções de pertinência elementares $\mu_{A_k^i}(x_k(t)), k = 1 \ldots p$ através do produto cartesiano:

$$u_i(t) = \mu_{A_i}(x(t)) = \mu_{A_1^i}(x_1(t)) \wedge ... \wedge \mu_{A_p^i}(x_p(t)), \tag{2.5}$$

onde $\wedge$ é um operador t-norma.

A saída do algoritmo de indução de regras é a matriz $\Xi = \{\xi_{ik}, i = 1 \ldots n, k = 1 \ldots p\}$, da qual um elemento $\xi_{ik} \in \Xi$ define qual símbolo na partição da variável x_k deve ser associado ao componente A_k^i na regra i. Um valor $\xi_{ik} = j$ indica que o conjunto *fuzzy* A_{kj} (isto é, o j-ésimo conjunto *fuzzy* na partição *fuzzy* da variável x_k) deve ser usada como o componente A_k^i na regra A_i.

Para ilustrar a etapa de fuzzificação, será apresentado um pequeno exemplo de um problema de duas classes (c_1 e c_2), com 400 registros e duas variáveis (x_1 e x_2). A Figura 2.2 apresenta a visualização dos registros da base de dados de exemplo no espaço. Os símbolos multidimensionais A_1, A_2, A_3 e A_4 representam conjunções dos símbolos definidos no domínio das variáveis x_1 e x_2. Foram utilizados dois conjuntos *fuzzy* de forma triangular na partição de cada variável.

A saída esperada do algoritmo de indução de regras para esse problema é a matriz Ξ conforme apresentado na Tabela 2.1:

Tabela 2.1: Representação da matriz Ξ para o exemplo.

	x_1	x_2
A_1	1	1
A_2	1	2
A_3	2	2
A_4	2	1

Dessa forma, os símbolos multidimensionais A_1, A_2, A_3 e A_4 representam as

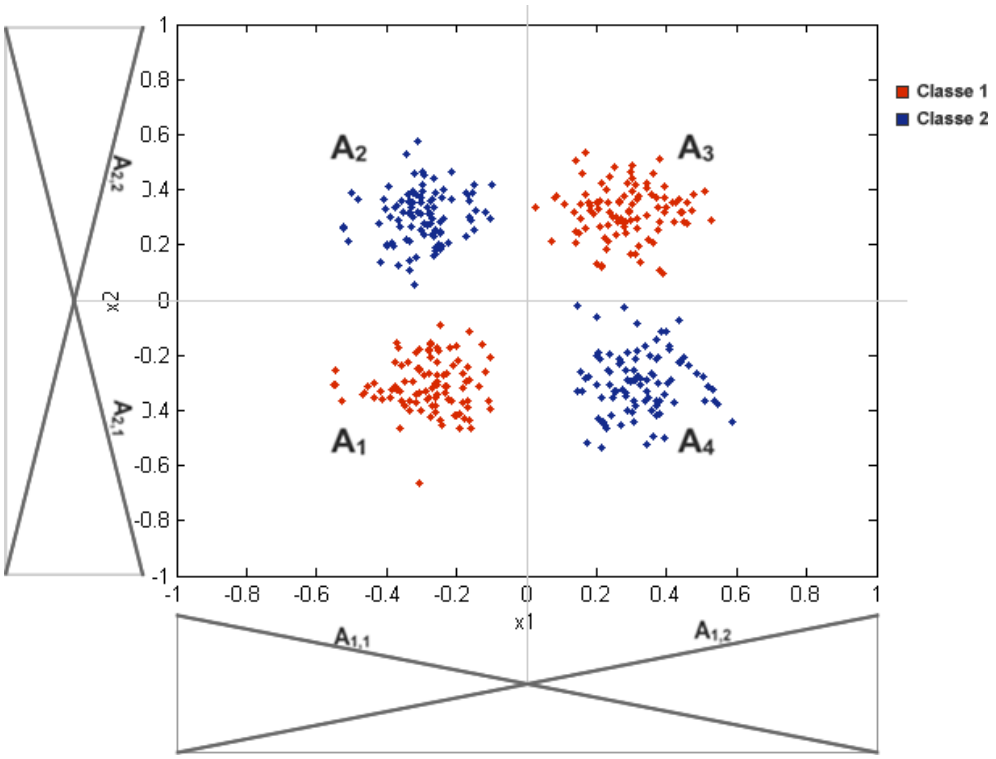

Figura 2.2: Exemplo de fuzzificação em uma base de dados sintética.

conjunções unidimensionais:

- A_1: $A_{1,1} \wedge A_{2,1}$
- A_2: $A_{1,1} \wedge A_{2,2}$
- A_3: $A_{1,2} \wedge A_{2,2}$
- A_4: $A_{1,2} \wedge A_{2,1}$

A Figura 2.3 utiliza o mesmo exemplo, porém nessa figura um registro $x(t)$ está destacado. Supõe-se que tal registro tenha como graus de pertinência: $\mu_{A_{1,1}}(x(t)) = 0,65$, $\mu_{A_{1,2}}(x(t)) = 0,35$, $\mu_{A_{2,1}}(x(t)) = 0,75$ e $\mu_{A_{2,2}}(x(t)) = 0,25$. Supõe-se também a utilização do operador t-norma produto. Dessa forma, a pertinência de $x(t)$ aos símbolos multidimensionais $A_i(x(t))$:

- $\mu A_1(x(t))$: $\mu A_{1,1}(x(t)) \wedge \mu A_{2,1}(x(t)) = 0.65 \wedge 0.75 = 0.65 \cdot 0.75 = 0.4875$
- $\mu A_2(x(t))$: $\mu A_{1,1}(x(t)) \wedge \mu A_{2,2}(x(t)) = 0.65 \wedge 0.25 = 0.65 \cdot 0.25 = 0.1625$
- $\mu A_3(x(t))$: $\mu A_{1,2}(x(t)) \wedge \mu A_{2,2}(x(t)) = 0.35 \wedge 0.25 = 0.35 \cdot 0.25 = 0.0875$

- $\mu A_4(x(t))$: $\mu A_{1,2}(x(t)) \wedge \mu A_{2,1}(x(t)) = 0.35 \wedge 0.75 = 0.35 \cdot 0.75 = 0.2625$

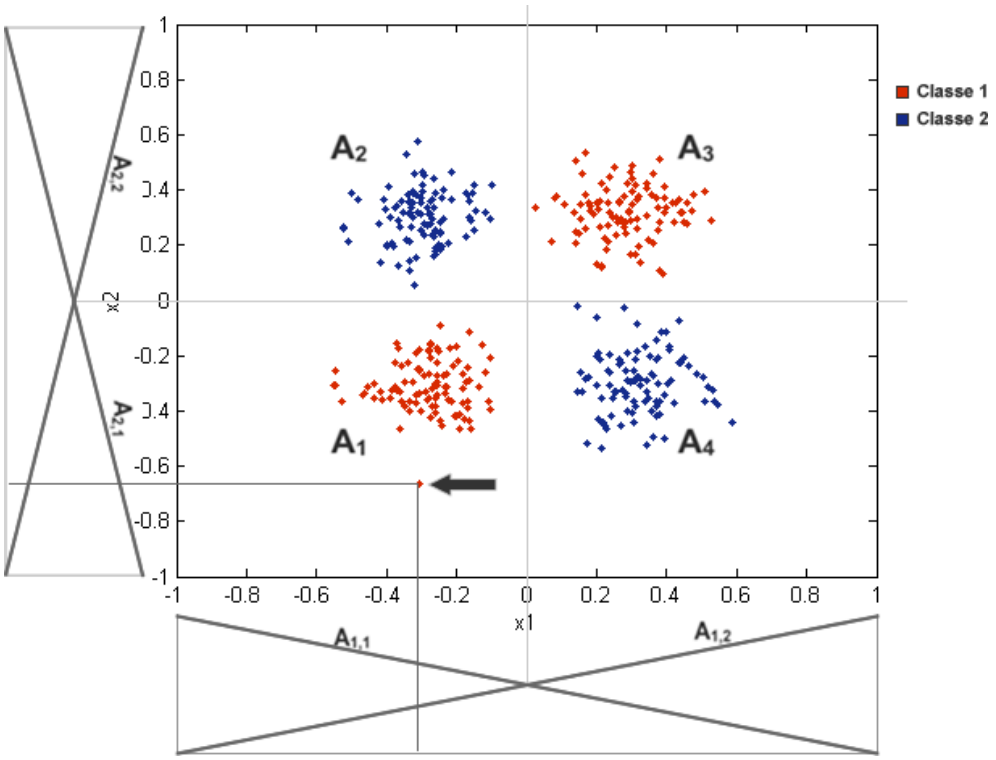

Figura 2.3: Exemplo de pertinência de um registro aos símbolos multidimensionais.

2.2.4 Inferência

A inferência é o mapeamento $I : [0,1]^n \rightarrow [0,1]^m$, onde n é o número de regras e m é o número de classes. Um modelo é construído pela matriz de relação *fuzzy* $\Phi \in [0,1]^{n \times m}$, da qual cada componente $\phi_{ij} = \mu_\varphi(A_i, B_j)$ representa a confiança da regra $A_i \rightarrow B_j$, tal que a regra ponderada é:

$$\text{se } \mathbf{x} \text{ é } A_i \text{ então } \mathbf{y} \text{ é } (B_1/\varphi_{i1}, \ldots, B_m/\varphi_{im}). \tag{2.6}$$

A inferência *fuzzy* é computada pelo operador de composição *fuzzy*.

$$\widehat{v}(t) = \widehat{u}(t) \circ \Phi. \tag{2.7}$$

Em problemas de classificação, os símbolos *fuzzy* de saída representam as classes e podem ser considerados independentemente, tal como $\Phi = [\varphi_1 | \ldots | \varphi_m]$. Nesse

caso, os componentes do vetor $\widehat{v} \in [0,1]^m$ são computados como o produto:

$$\widehat{v}_j(t) = \widehat{u}(t).\varphi_j, j = 1 \ldots m, \tag{2.8}$$

onde φ_j é a confiança da regra em relação à classe B_j e $\widehat{v}_j(t)$ é a pertinência da classe computada para a amostra $x(t)$, tal que $\widehat{v}_j(t) = \mu_{B_j}(x(t))$.

Para o mesmo exemplo apresentado pela Figura 2.2, a matriz Φ poderia ser representada pela Tabela 2.2:

Tabela 2.2: Representação da matriz Φ para o exemplo.

	c_1	c_2
A_1	1	0
A_2	0	1
A_3	1	0
A_4	0	1

Assim, o seguinte modelo de regras *fuzzy* poderia ser construído:

- Regra 1: Se x_1 é $A_{1,1}$ e x_2 é $A_{2,1}$ então c_1 (com $\varphi_1 = 1.00$) / c_2 (com $\varphi_2 = 0.00$)
- Regra 2: Se x_1 é $A_{1,1}$ e x_2 é $A_{2,2}$ então c_1 (com $\varphi_1 = 0.00$) / c_2 (com $\varphi_2 = 1.00$)
- Regra 3: Se x_1 é $A_{1,2}$ e x_2 é $A_{2,2}$ então c_1 (com $\varphi_1 = 1.00$) / c_2 (com $\varphi_2 = 0.00$)
- Regra 4: Se x_1 é $A_{1,2}$ e x_2 é $A_{2,1}$ então c_1 (com $\varphi_1 = 0.00$) / c_2 (com $\varphi_2 = 1.00$)

O uso de fatores de confiança para expressar a certeza de regras *fuzzy* é bastante investigado para problemas de classificação e será discutido na Seção 3.1.2.

2.2.5 Defuzzificação

Em problemas de regressão, defuzzificação é o mapeamento $D : [0,1]^m \rightarrow \mathbf{R}$, onde m é a dimensão do espaço de características de saída. Em problemas de classificação, defuzzificação é o mapeamento $D : [0,1]^m \rightarrow \mathbf{N}$ e computa o índice da classe de saída, baseado no símbolo *fuzzy* de saída. Geralmente a regra de máximo valor é

usada, tal que o índice da classe é computado como o componente com o maior valor de pertinência:

$$\widehat{y}(t) = j : v_j(t) = max(\widehat{v}(t)). \tag{2.9}$$

Capítulo 3

Identificação de Modelos *Fuzzy*

O desenvolvimento de sistemas baseados em regras *fuzzy* tem sido amplamente estudado e muitos algoritmos para a geração de modelos *fuzzy* interpretáveis têm sido propostos [9] [21] [10] [22] [14] [23].

De um ponto de vista dos dados, é desejável que a base de regras do modelo represente todo o domínio dos dados e, se possível, que informação adicional possa ser garantida sobre amostras que não estão contidas nos dados de treinamento. Entretanto, é importante também que se preserve a interpretabilidade do modelo, garantindo assim a aquisição de conhecimento a partir dos dados.

Dessa forma, um dos pontos principais no desenvolvimento de sistemas baseados em regras *fuzzy* interpretáveis é a definição do número de regras a ser considerado de forma a garantir um bom equilíbrio entre precisão e interpretabilidade.

O sistema *fuzzy* implementado nesse trabalho utiliza análise espectral para a estimação do número de regras e um algoritmo de agrupamento é utilizado para estimar os centros das regras. Um algoritmo para a solução de um problema de otimização quadrática limitado realiza a otimização dos parâmetros de confiança das regras.

Um algoritmo genético efetua a seleção da estrutura, buscando otimizar o subconjunto de variáveis a serem consideradas pelo modelo e também o número de conjuntos *fuzzy* no domínio de cada variável. O algoritmo genético utiliza uma função de aptidão que tem como objetivo balancear precisão e interpretabilidade do modelo gerado pela estrutura candidata.

3.1 Indução de Regras *Fuzzy*

O algoritmo de indução de regras tem como objetivo gerar uma base de regras que represente o máximo possível o conjunto de dados de treinamento e definir pesos para cada regra, garantindo assim alguma informação adicional sobre a relevância das regras com respeito ao problema.

A próxima subseção apresenta a análise espectral, utilizada no algoritmo de indução de regras para estimar o número de regras do modelo *fuzzy* gerado pelo algoritmo de indução de regras. A subseção seguinte apresenta o algoritmo de otimização de pesos de regras, que utiliza uma formulação de um algoritmo de otimização quadrática limitado.

3.1.1 Análise Espectral

Uma das técnicas mais utilizadas para análise exploratória de dados é o agrupamento, com aplicações nas mais diversas áreas do conhecimento como biologia, ciências sociais e psicologia [24]. Na maioria dos estudos envolvendo dados, é comum o estudo da similaridade entre os registros como forma de se ter uma idéia do comportamento do conjunto de dados como um todo. Recentemente, o agrupamento espectral tem se estabelecido como um importante método de agrupamento, em alternativa a métodos de agrupamento tradicionais, como é o caso do *k-means*, que geralmente precisam da definição de um conjunto de parâmetros, como o número de grupos.

Li et al. [25] apresentam um algoritmo para análise espectral de dados com o objetivo de estimar com eficiência o número de grupos em um conjunto de dados. Nesse trabalho, a análise espectral é utilizada para prover uma estimativa do número de regras.

Dado um conjunto de dados $x(t), T = 1 \ldots N$ e uma noção de similaridade a_{ij} entre todos os pares de registros $x(i)$ e $x(j)$. Tem-se então um conjunto de informações de similaridade que pode ser representado em forma de um grafo de similaridade $G = (V, E)$. O conjunto de vértices V no grafo representa as amostras de entrada e conjunto de arestas E é definido pela matriz de similaridade (ou matriz de afinidade) A, de dimensão $N \times N$.

O grafo é não-orientado e ponderado, tal que a matriz de afinidade é simétrica, de valores reais e seus elementos representam a similaridade entre as amostras de entrada. Nesse trabalho, a métrica de similaridade é calculada pela função Gaussiana, tal que cada elemento da matriz de afinidade é computado como:

$$a_{ij} = \begin{cases} exp\left(-\frac{\|x(i) - x(j)\|^2}{2\sigma^2}\right), \ se \ i \neq j \\ 0, \ caso \ contrário \end{cases} \tag{3.1}$$

onde σ é um parâmetro de dispersão que controla o espalhamento da função de similaridade. A matriz D de dimensão $N \times N$ é uma matriz diagonal cujos elementos são os graus dos nós de G, computados pela soma de similaridade dos vizinhos de cada nó:

$$d_{ii} = \sum_{j=1...N} a_{ij} \tag{3.2}$$

O problema de agrupamento espectral é uma adaptação do problema de agrupamento clássico reformulado para usar o grafo de similaridade G. É um problema de corte de grafo onde deve-se separar os nós em subconjuntos, tal que os arcos que ligam diferentes conjuntos tenham pesos muito baixos e os arcos dentro de um mesmo grupo tenham pesos altos. Isso significa que nós em um mesmo grupo são muito similares e nós em grupos diferentes não são similares.

Filippone et al. [26] provaram que a solução ótima desse problema é NP-difícil, tal que uma solução aproximada é obtida pelo cálculo de autovalores de uma transformação da matriz de adjacência, denominada matriz Laplaciana, computada como $L = D - A$. A matriz Laplaciana é geralmente normalizada e existem algumas definições para a Laplaciana normalizada, cada uma com suas propriedades peculiares [24].

Nesse trabalho, a Laplaciana normalizada analisada por Li et al. [25] é adotada, que é computada diretamente pela normalização da matriz de adjacência como:

$$L = D^{-1/2} A D^{-1/2}. \tag{3.3}$$

Ng et al. [27] utilizaram a mesma definição da Laplaciana normalizada para agrupamento espectral em um algoritmo que tem a mesma idéia básica da maioria

dos algoritmos de agrupamento espectral, isto é, encontrar uma nova representação dos dados baseada nos maiores autovalores da matriz normalizada Laplaciana e então realizar o agrupamento nessa nova representação. A principal questão, que é particularmente interessante para a indução de regras, é o significado da palavra "maior", que é diretamente relacionada ao número de grupos.

Segundo a teoria espectral dos grafos, o espectro do grafo G é definido como o conjunto de autovalores de sua matriz de adjacência e pode-se analisar a estrutura de G através da análise de seus autovalores. Muitas propriedades importantes de um grafo podem ser expressas por meio dos seus autovalores (seu espectro). Assim, pode-se realizar a análise da estrutura de um conjunto de dados pela observação dos autovalores da Laplaciana normalizada, computada pela autodecomposição:

$$L = Z\Lambda Z^T, \tag{3.4}$$

onde Z é a matriz ortogonal dos autovetorese Λ é a matriz diagonal dos autovalores, que são todos reais, já que a matriz Laplaciana normalizada é simétrica. As colunas de Λ (e de Z) são ordenadas tal que os autovalores $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_N$.

Quando a matriz Laplaciana normalizada é computada como na Equação 3.3, as seguintes propriedades de seus autovalores são derivadas da teoria espectral dos grafos [28] [25]:

1. $\lambda_1 = 1$ e $\sum_{i=1\ldots N} \lambda_i = 0$
2. $-1 \leq \lambda_i \leq 1, i = 1 \ldots N$
3. Se o grafo é conectado então $\lambda_2 < 1$

Pelas propriedades 1. e 2. acima pode-se concluir que haverá sempre um inteiro K, tal que $\lambda_i \geq 0, 1 < i \leq K$ e $\lambda_j < 0, K < j \leq N$, geralmente $K \ll N$.

Li et al. [25] também mostraram que, para uma base de dados contendo K grupos disjuntos e se a matriz de adjacência consegue um agrupamento perfeito, isto é:

$$\widehat{a}_{ij} = \begin{cases} 1, \ \textit{se } x(i) \textit{ e } x(j) \textit{ pertencem a um mesmo grupo} \\ 0, \ \textit{caso contrário} \end{cases} \tag{3.5}$$

então os autovalores da Laplaciana normalizada computados para a matriz de adjacência dada pela Equação 3.5 são:

$$\widehat{\lambda}_i = \begin{cases} 1, \ 1 < i \leq K \\ 0, \ K < i \leq N. \end{cases} \tag{3.6}$$

Se outra função de similaridade é utilizada para computar a matriz de adjacência, por exemplo a função baseada na função Gaussiana (Equação 3.1), então a solução da autodecomposição (Equação 3.4) é tal que $\lambda_i \rightarrow \widehat{\lambda}_i, i = 1 \ldots N$ [25]. O número de autovalores positivos da Laplaciana pode então ser usado para estimar o número de grupos.

Para ilustrar os conceitos descritos serão apresentado alguns exemplos de problemas de autovalores em diferentes tipos de bases de dados.

A Figura 3.1 apresenta a solução do problema de autovalor na mesma base de dados sintética utilizada como exemplo na Seção 2.2.3 (400 registros com valores aleatórios de distribuição Gaussiana em dois atributos). A Figura 3.2 apresenta a solução do problema de autovalor em uma base de dados semelhante à da Figura 3.2, porém com grupos mais espalhados e mais dificilmente separados.

Na Figura 3.1(A), nota-se visualmente quatro grupos distintos e o número correto de grupos é igual ao número de autovalores positivos encontrados (Figura 3.1(C)). No exemplo da Figura 3.2, seis grupos são estimados (Figura 3.2(C)), mas o quinto e o sexto autovalores positivos são muito pequenos.

As distâncias relativas entre os componentes das bases de dados sintéticas de exemplo podem ser observadas na Figura 3.1(B) e na Figura 3.2(B), através do *Data Image*, onde a escala de cor representa qualitativamente os valores da matriz de afinidade. Esse tipo de representação é muito útil na análise de dados e é utilizado para a avaliação da qualidade de métodos de agrupamento.

As Figuras 3.3 e 3.4 apresentam exemplos da análise espectral sobre base de dados Iris, que possui 150 amostras, quatro variáveis de entrada e três classes. No exemplo da Figura 3.3, a matriz de similaridade é computada considerando todas as quatro variáveis. Pela Figura 3.3(B) percebe-se que seis grupos são estimados pelos autovalores positivos da Laplaciana. No exemplo ilustrado pela Figura 3.4, apenas duas variáveis (comprimento da pétala e largura da pétala) são usados. O número de autovalores positivos da Laplaciana encontrados é igual ao número real

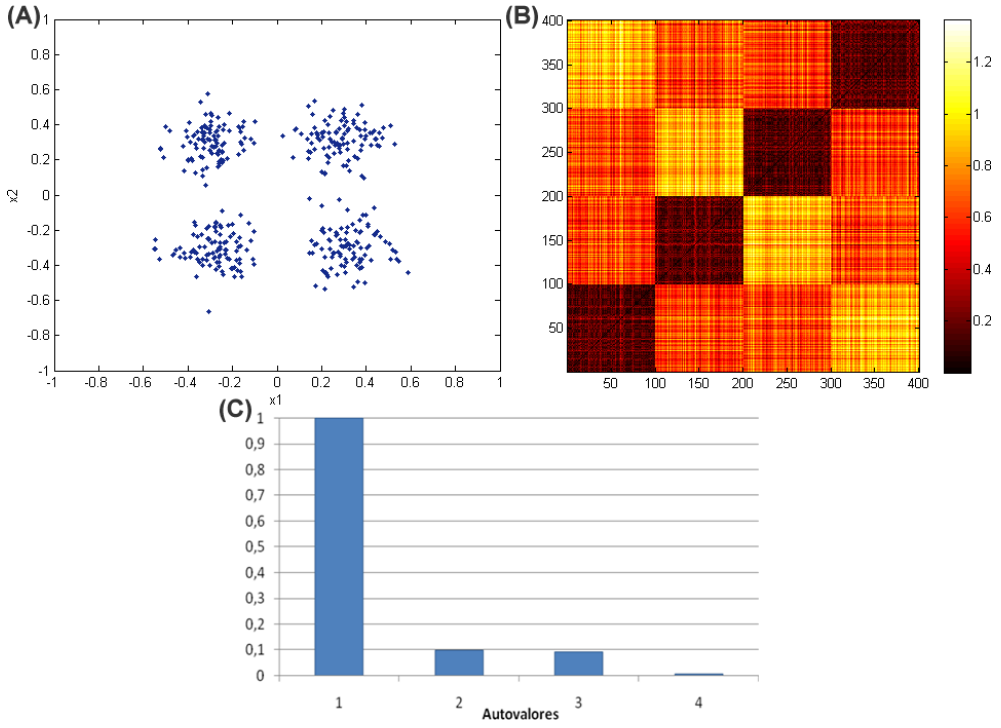

Figura 3.1: Base de Dados Sintética com Grupos Facilmente Separados: (A) Visualização dos Dados, (B) Distância Relativa Entre os Registros e (C) Autovalores da Laplaciana Encontrados.

de classes, três, como pode ser visto na Figura 3.4(B).

Novamente o *Data Image* é apresentado para as bases de dados (Figuras 3.3(A) e 3.4(A)).

Os exemplos apresentados mostram dois aspectos essenciais que são diretamente relacionados ao número de autovalores positivos: a separação geométrica das classes e o número de variáveis. Em outras palavras, o número de autovalores positivos da matriz Laplaciana está relacionado à complexidade do conjunto de dados e, então, poderia ser usado para estimar o número de regras no modelo *fuzzy* gerado.

Para qualquer par de registros no conjunto de dados, o correspondente componente de adjacência computado pela Equação 3.1 será maior se um menor número de variáveis é utilizado nesse cálculo. As diferenças relativas da matriz Laplaciana normalizada também serão maiores, resultando em grupos mais bem separáveis e, consequentemente, uma redução no número de autovalores. Por isso, a etapa de seleção de estrutura visa definir um subconjunto de variáveis que faça com que o problema seja menos complexo, como será apresentado na Seção 3.2.

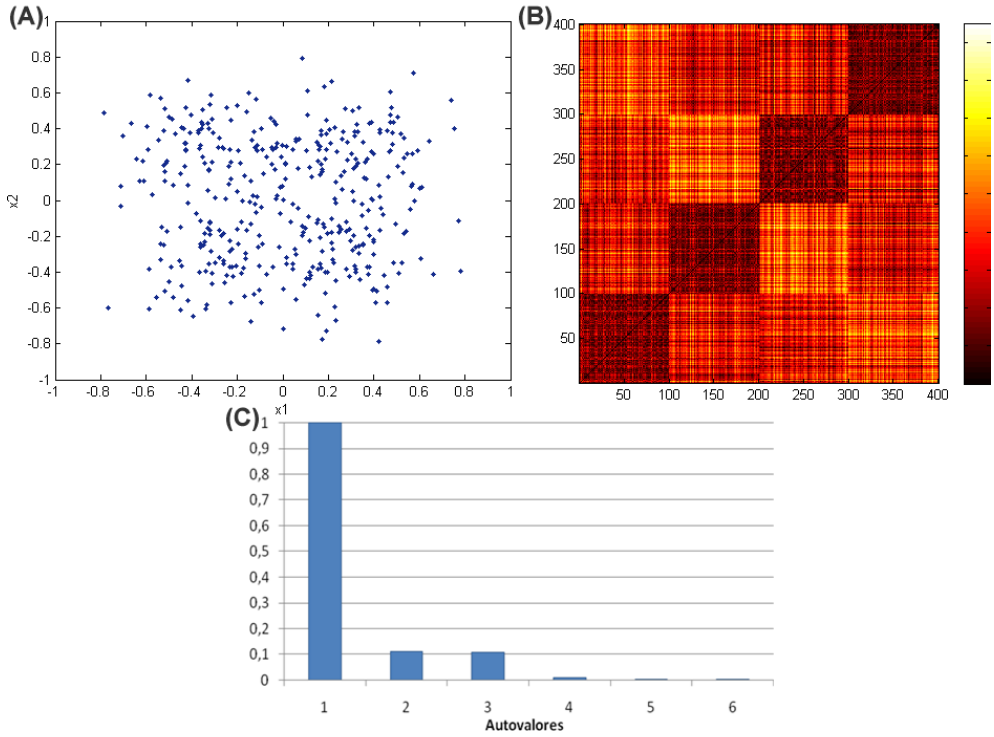

Figura 3.2: Base de Dados Sintética com Grupos Dificilmente Separados: (A) Visualização dos Dados, (B) Distância Relativa Entre os Registros e (C) Autovalores da Laplaciana Encontrados.

3.1.2 Utilização de Peso em Regras *Fuzzy*

A estimação dos parâmetros de peso das regras corresponde à definição dos pesos de regras φ_j que relacionam o símbolo de premissa de regra A_i ao consequente da regra (classe) B_j. Pode-se encontrar na literatura diversos estudos sobre a utilização de pesos em sistemas baseados em regras *fuzzy*.

Em [29] é discutida a utilização de pesos de regras *fuzzy* em problemas de aproximação de função e mostra-se que, ao invés de se atribuir pesos às regras, pode-se realizar modificações nas funções de pertinência dos conjuntos *fuzzy* de forma equivalente. Entretanto, Ishibuchi [20] faz um estudo sobre os efeitos da utilização de pesos de regras em sistemas *fuzzy* onde destaca que a modificação das funções de pertinência deteriora a compreensibilidade dos valores linguísticos nas regras *fuzzy* por especialistas humanos e que, mesmo que a modificação nos conjuntos *fuzzy* seja equivalente, o uso de pesos de regras não interfere na interpretabilidade do modelo e por isso é uma abordagem mais interessante.

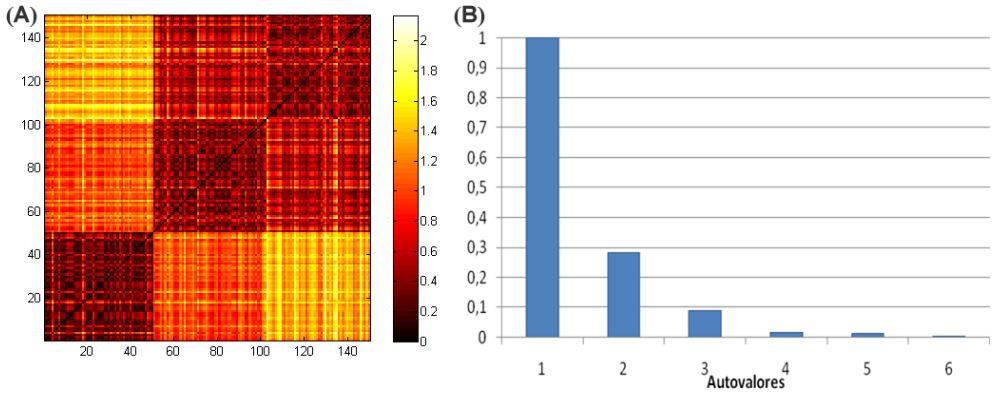

Figura 3.3: Base de Iris Utilizando os Quatro Atributos: (A) Distância Relativa Entre os Registros e (B) Autovalores da Laplaciana Encontrados.

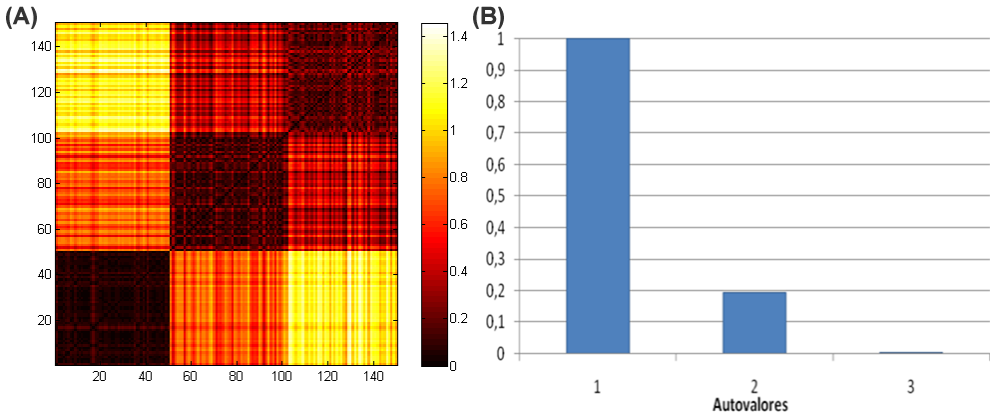

Figura 3.4: Base de Iris Utilizando Apenas Dois Atributos: (A) Distância Relativa Entre os Registros e (B) Autovalores da Laplaciana Encontrados.

O peso de uma regra pode ser interpretado como a sua força dentro do sistema. Quanto maior o peso da regra, maior é a área de decisão dessa regra [20]. Diversos autores têm proposto medidas de qualidade para as regras *fuzzy*. Observando-se regras com o mesmo antecedente mas diferentes consequentes pode-se definir o consequente mais adequado para esse antecedente como aquele que faz com que a regra tenha uma melhor medida de qualidade.

As medidas de qualidade de regras *fuzzy* mais típicas são o suporte e a confiança, definidos como [13]:

$$\text{supp}(A_i \rightarrow B_j) = \sum_{t=1...N} \mu_{A_i}(x(t)).v_j(t); \tag{3.7}$$

$$\text{conf}(A_i \rightarrow B_j) = \frac{\sum_{t=1\ldots N} \mu_{A_i}(x(t)).v_j(t)}{\sum_{t=1\ldots N} \mu_{A_i}(x(t))}, \tag{3.8}$$

onde $V_j = [v_j(1)|\ldots|v_j(N)]^T$ é o vetor coluna com as pertinências observadas correspondentes da classe B_j, tal que $v_j(t) = \mu_{B_j}(y(t))$.

Em [30], Ishibuchi e Yamamoto propõem algumas definições para o peso de regras baseadas no uso da confiança. Uma delas utiliza a diferença entre a maior e a segunda maior confiança para o mesmo antecedente da regra. Outra baseia-se na diferença entre a confiança de um antecedente para uma classe e a soma das confianças para o mesmo antecedente com as classes complementares.

Pode-se encontrar na literatura diversos métodos para otimização de pesos de regras *fuzzy* com o objetivo de otimizar os parâmetros de peso para priorizar as regras que melhor separam as classes. Jahromi et al. [31] propõem a otimização de peso de regras utilizando o método iterativo de subida em colina (*Hill-Climbing*), que é um método de busca local. Esse algoritmo é iniciado com peso 1 para toda a base de regras e, a cada iteração, o peso de apenas uma regra é ajustado, de forma que a próxima solução seja melhor que a atual ou igual a ela.

Nozaki et al. [12] utilizam um método para especificação de pesos de regras baseado em correção de erro de classificação através de recompensa e penalização nos parâmetros de peso. Se uma regra classifica corretamente um registro, seu peso é aumentado por uma função de recompensa. Caso contrário seu peso é reduzido por uma função de penalização.

Em [5], Evsukoff realiza a otimização do peso das regras através da solução de um problema de otimização quadrática. Essa foi a estratégia utilizada nesse trabalho e a próxima subseção tem o objetivo de detalhá-la.

Otimização de Pesos de Regras Através de Otimização Quadrática

Pesos de regras ótimos podem ser computados no sentido de mínimos quadrados, através da solução do seguinte problema de otimização quadrática limitado para cada classe B_j:

$$\begin{gathered} \underset{\varphi_j}{\text{minimizar}} \; \|U\varphi_j - V_j\|^2 \\ \text{sujeito a } 0 \leq \varphi_{ij} \leq 1, i = 1\ldots n \end{gathered} \tag{3.9}$$

onde $U = [u(1)|\ldots|u(N)]^T$, $U \in [0,1]^{N\times n}$ é a matriz da qual cada linha é o vetor *fuzzy* computado para cada amostra de dado como na Equação 2.3. Os limites tentam restringir os pesos dentro do intervalo $[0,1]$, tal que eles possam ser interpretados como a confiança da regra *fuzzy*. As restrições dos limites também evitam valores muito altos para os parâmetros de pesos de regras, permitindo a suavidade da solução.

Expandindo o termo quadrático na função objetivo e ignorando os termos constantes, a Equação 3.9 pode ser reformulada como:

$$\begin{aligned} &\underset{\varphi_j}{\text{minimizar}}\ \tfrac{1}{2}\varphi_j^T K\varphi_j - C^T\varphi_j \\ &\text{sujeito a } 0 \leq \varphi_{ij} \leq 1, i = 1\ldots n \end{aligned} \tag{3.10}$$

onde $K = U^T U$ é uma matriz positiva definida e $C^T = V_j^T U$.

O problema de otimização quadrática definido pela Equação 3.10 é amplamente conhecido e sua solução pode ser computada por eficientes algoritmos numéricos [32]. Esse trabalho não tem como objetivo realizar estudos sobre algoritmos para a solução de problemas de otimização quadrática.

Os pesos de regras resultantes podem ser utilizados para auxiliar a interpretação o modelo. Tipicamente, na solução de 3.10, três tipos de valores de pesos de regras podem ocorrer:

- $\varphi = 0$: significa que a regra $A_i \rightarrow \neg B_j$ é certa e a região definida pelo símbolo A_i não pode ser atribuída à classe B_j;
- $0 < \varphi_{ij} < 1$: significa que a regra é incerta e que o símbolo A_i representa uma região na borda da classe B_j;
- $\varphi_{ij} = 1$: significa que a regra $A_i \rightarrow B_j$ é certa e a região definida pelo símbolo A_i é totalmente relacionada à classe B_j.

Para uma estrutura de modelo fixada, quanto mais os valores de solução $\varphi_{ij} = 1$, mais certo é o classificador e também mais alta é a norma do vetor de pesos definido por:

$$\|\varphi_j\|^2 = \varphi_j^T.\varphi_j, j = 1\ldots m. \tag{3.11}$$

A soma $\sum_{j=1...m} \|\varphi_j\|^2$, quando comparada com o número de regras n é uma medida da quantidade de regras certas e pode então ser interpretada como uma medida de qualidade do classificador.

3.1.3 O Algoritmo de Indução de Regras

A combinação de conjuntos *fuzzy* unidimensionais é muito mais intuitiva que um único conjunto *fuzzy* multidimensional. Então, a idéia do algoritmo de indução de regras é executar um algoritmo de agrupamento sobre o conjunto de dados e então associar uma regra a cada grupo. Para cada grupo, a combinação mais próxima ao centro do grupo, entre todas as possíveis combinações de conjuntos *fuzzy*, é escolhida como a regra a ser incluída no modelo.

Essa abordagem para indução de regras é altamente flexível e um grande número de outras estratégias poderia ser utilizado [26]. Nesse trabalho é utilizado o agrupamento espectral de acordo com a abordagem de Ng et al. [27].

Na abordagem de agrupamento espectral, o algoritmo $k-means$ é aplicado sobre a matriz $\widehat{Z}$, computada como as primeiras K colunas da matriz de autovetores Z, lembrando que as colunas de Z são ordenadas de acordo com os correspondentes autovalores, tal que as primeiras colunas correspondem aos maiores autovalores. Através do $k-means$, K grupos são computados no espaço transformado e os registros pertencentes a cada um dos grupos são associados aos seus respectivos registros no espaço original de entrada. Então, K centros de grupos são computados como a média dos registros de cada grupo no espaço original.

Nesse trabalho, uma regra *fuzzy* é computada como a combinação de conjuntos *fuzzy* unidimensionais mais próximos de cada centro de grupo. Essa abordagem foi escolhida pelo fato de ser simples e eficiente, mas outra abordagem para derivar regras *fuzzy* interpretáveis de grupos multidimensionais poderia ser adotada. Por exemplo, Castellano et al. [15] propuseram uma abordagem que também ajusta os protótipos dos conjuntos *fuzzy* em respeito aos centros de grupos.

As funções de pertinência *fuzzy* unidimensionais, como definidas pela Equação 3.1, dependem apenas do número de conjuntos *fuzzy* na partição de cada variável de entrada. Supõe-se que o algoritmo de indução de regras seja executado com um *loop* externo que corresponde ao algoritmo de seleção de estrutura que de-

fine as variáveis que devem ser incluídas no modelo e o número de conjuntos *fuzzy* no domínio de cada variável.

A cada iteração do algoritmo de otimização de estrutura, estruturas candidatas representadas pelo vetor $\gamma = [\gamma_1, \ldots, \gamma_p]$ são geradas, como apresentado na Seção 3.2. Cada uma dessas estruturas é avaliada de acordo com o Algoritmo de Indução de Regras.

Uma versão em alto nível do Algoritmo de Indução de regras é exibida no Algoritmo 1.

Entrada: $\gamma = [\gamma_1, \ldots, \gamma_p]$;
$\alpha = \{\alpha_k, k = 1 \ldots p\}$;
$T = \{(x(t), y(t)), t = 1 \ldots N\}$

Saída: $\Xi = \{\xi_{ik}, i = 1 \ldots n, k = 1 \ldots p\}$

```
1  inicio
       /* Análise Espectral                                          */
2      Calcula a matriz A;
3      Calcula a matriz D;
4      Calcula L = D^{-1/2} A D^{-1/2};
5      Calcula L = Z Λ Z^T ;                          // λ_1 ≤ λ_2 ≤ ... λ_N
6      n ← K : λ_i ≥ δ, i = 1 ... K ;          // estima o número de regras
7      W ← agrupamento(n) ;                     // algoritmo de agrupamento
       /* Atribui uma regra a cada grupo encontrado                  */
8      para i = 1 ... n faça
9          para k = 1 ... p faça
10             se γ_k > 1 então
                   // calcula o protótipo mais próximo
11                 ξ_ik = j* : α_kj* = argmin(α_kj − w_ik);
12             fim
13         fim
14     fim
15 fim
```

Algoritmo 1: Algoritmo de Indução de Regras.

O número de regras é estimado pela análise espectral na linha 06. A matriz

$W \in \mathbf{R}^{p \times n}$ na linha 07 armazena as coordenadas de centros de grupos, computados pela abordagem discutida acima. As regras são de fato escolhidas na linha 11, pelo protótipo mais próximo às coordenadas de centro de grupo correspondente. O conjunto de regras encontrado pelo algoritmo de indução de regras define a premissa de 2.1 e representa uma descrição linguística do conjunto de aprendizado.

3.2 Seleção de Estrutura

O algoritmo de seleção de estrutura utilizado nesse trabalho provê uma estimação do vetor de estrutura $\gamma = [\gamma_1, \ldots, \gamma_p]$, que define as variáveis que devem ser incluídas no modelo e também o número de conjuntos *fuzzy* no domínio de cada variável.

A definição de atributos para um determinado problema tem grande influência nos algoritmos de classificação e a identificação dos atributos realmente relevantes para uma aplicação é um fator de extrema importância para o seu sucesso [11]. Alguns atributos podem prejudicar a tarefa de classificação de dados por esconderem informações importantes presentes em outros atributos.

Liu et al. [33] definem o procedimento de seleção de atributos como um processo iterativo em quatro passos: a seleção de subconjuntos de soluções candidatas, a avaliação do subconjunto candidato, a definição de um critério de parada e a validação do resultado.

No processo de seleção de atributos a seleção de sub-conjuntos de soluções candidatas pode ocorrer, de uma maneira mais simples, utilizando uma estratégia incremental, na qual cada variável é adicionada ao modelo de acordo com sua capacidade de atender a um critério de avaliação de desempenho. Basicamente, a seleção de atributos pode acontecer seguindo uma abordagem filtro, uma abordagem *wrapper* ou uma abordagem híbrida.

A abordagem filtro utiliza critérios de avaliação independentes do método empregado que visam avaliar a qualidade de um subconjunto de características através da exploração de características intrínsecas do conjunto de treinamento como medidas de distância, medidas de dependência e medidas de consistência [33].

A abordagem *wrapper* utiliza critérios de avaliação dependentes, que utilizam a precisão do próprio algoritmo de mineração de dados considerando o sub-conjunto

selecionado como medida de qualidade das características [33]. Geralmente esse tipo de abordagem encontra atributos mais adequados ao algoritmo de mineração, mas também tende a ser computacionalmente mais custoso.

A abordagem híbrida combina as duas abordagens anteriores utilizando medidas independentes para pré-selecionar os subconjuntos de variáveis candidatos mais promissores, e o algoritmo de classificação para selecionar o melhor subconjunto entre os pré-selecionados.

Mikut et al. utilizam um algoritmo de seleção de estruturas do tipo filtro que é iniciado com um conjunto vazio de variáveis selecionadas e, a cada passo, uma nova variável é adicionada, caso sua presença no modelo aumente a precisão esperada. A medida de relevância é baseada no ganho de informação do atributo e é independente do classificador.

Casillas et al. propõem um algoritmo de seleção de atributos híbrido. Tal algoritmo seleciona, na primeira etapa, o número de variáveis a ser considerado pelo modelo levando em consideração medidas de separabilidade de classes. O resultado da primeira etapa é utilizado como base para um algoritmo de seleção do tipo *wrapper* que leva em conta a precisão resultante de cada subconjunto de variáveis avaliado.

Kohavi e John [34] fazem uma revisão sobre diversos métodos de seleção de variáveis utilizando a abordagem filtro e a abordagem *wrapper*, fazendo um estudo comparativo entre esses métodos.

Além de definir os atributos a serem considerados pelo modelo, a estrutura pode definir o número de conjuntos *fuzzy* no domínio de cada variável.

Uma estratégia incremental foi utilizada para definir o número de conjuntos *fuzzy* para cada variável em problemas de regressão por Evsukoff em [35]. Em [5], Evsukoff propõe um algoritmo onde o problema de definição do número de conjuntos *fuzzy* para cada variável é unido ao problema de seleção de variáveis na seleção de estrutura. Segundo esse algoritmo, uma estrutura é representada pelo vetor $\gamma = [\gamma_1, \ldots, \gamma_p]$, onde $2 \leq \gamma_k \leq \gamma_{max}$ é o número de conjuntos *fuzzy* na partição de cada variável x_k e um valor $\gamma_k = 1$ indica que a variável x_k não deve ser incluída no modelo. Uma abordagem semelhante foi proposta por Cordon et al. [36], porém, esse algoritmo não leva em conta o problema de seleção de atributos.

O algoritmo de seleção de estrutura proposto por Evsukoff [5], por reunir o problema de seleção de variáveis e o problema de definição de conjuntos *fuzzy* por variável, faz com que o número de possíveis soluções se torne enorme e inviável para uma estratégia incremental. Algoritmos de otimização tradicionais também não são adequados nesse tipo de problema já que o espaço de busca é discreto, e não contínuo. Sendo assim, estratégias estocásticas, como os algoritmos genéticos são mais adequadas para a solução desse tipo de problema e são comumente utilizadas em problemas encontrados na literatura.

Nesse trabalho um algoritmo genético (AG) foi implementado para a seleção de atributos do modelo *fuzzy* e a seleção do número de conjuntos *fuzzy* no domínio de cada variável, de acordo com o algoritmo proposto em [5]. A próxima subseção apresenta uma visão geral sobre algoritmos genéticos e a subseção seguinte apresenta mais detalhadamente o algoritmo utilizado nesse trabalho para seleção de estrutura do modelo *fuzzy*.

3.2.1 Visão Geral Sobre Algoritmos Genéticos

O algoritmo genético (AG) é um método estocástico para otimização de problemas complexos e com vasto espaço de busca. O AG foi criado por John Holland [37] e é inspirado na Teoria da Evolução, de Charles Darwin.

Em sua forma básica, um algoritmo genético é implementado como uma analogia da evolução das espécies em que, a cada iteração (geração), uma população de representações abstratas de solução (indivíduos) é selecionada em um processo de busca de soluções melhores (evolução) por meio de operadores genéticos.

Os indivíduos de uma população são avaliados segundo uma função que retorna um valor que representa o quão apto é esse indivíduo. O operador de seleção define probabilisticamente os indivíduos na população que irão se recombinar e essa recombinação é feita, normalmente, através dos operadores de cruzamento (*crossover*) e mutação. Os indivíduos mais aptos tendem a gerar descendentes, que corresponderão à nova geração.

Basicamente, um algoritmo genético apresenta cinco aspectos fundamentais quando usado para resolver um problema:

1. Uma codificação genética de possíveis soluções para o problema;

2. Um procedimento para a criação de uma população inicial de possíveis soluções;

3. Uma função de avaliação que retorna a aptidão de cada indivíduo;

4. Operadores genéticos que manipulam a codificação dos indivíduos durante o processo de reprodução dando origem a novos indivíduos;

5. Parâmetros que controlam os processos de cruzamento e mutação, isto é, definem a probabilidade de ocorrência de tais operadores.

O Algoritmo 2 apresenta o ciclo básico de um algoritmo genético:

```
1  inicio
2      Inicializa a população;
3      Avalia os indivíduos da população;
4      enquanto critério de parada não satisfeito faça
5          Seleciona indivíduos para reprodução;
6          Aplica o operador de cruzamento;
7          Aplica o operador de mutação;
8          Avalia os indivíduos da população;
9          Seleciona os indivíduos para sobreviver (Elitismo);
10     fim enquanto
11 fim
```

Algoritmo 2: Ciclo Básico de um Algoritmo Genético

Em um algoritmo genético, um indivíduo é representado por um cromossomo, que corresponde a uma sequência de genes. Cada gene representa um atributo do problema a ser otimizado e pode ser codificado de várias formas. As mais usuais são as codificações binária e real.

A avaliação de um indivíduo (linhas 3 e 8) é feita pela função de aptidão e é determinada pelo cálculo da função objetivo, que depende das especificações do problema. Essa aptidão representa a sua tendência de sobrevivência durante o processo evolutivo.

O operador de seleção (linha 5) é um processo probabilístico que define os indivíduos selecionados para a geração de descendentes. Os métodos mais utilizados

são a seleção por roleta, seleção por *rank* e seleção por torneio.

O operador de cruzamento (linha 6) combina dois indivíduos pais escolhidos pelo operador de seleção, resultando em descendentes, que continuarão o processo de evolução do AG. Esse operador visa realizar a exploração de um local do espaço de busca próximo à região dos pais e deve ser aplicado sobre a população com uma alta probabilidade.

O operador de mutação (linha 7) realiza alterações aleatórias nos genes com o objetivo de perturbar o indivíduo. Tal operador visa realizar mudanças de região no espaço de busca, com o objetivo de melhor explorá-lo globalmente e deve ser aplicado aos indivíduos com uma baixa probabilidade.

O elitismo (linha 9) tem o objetivo de preservar uma parte da população que apresenta bons valores de aptidão. Esta técnica é aplicada como forma de auxiliar a convergência do AG, evitando que toda uma geração seja substituída por outra, possivelmente menos apta.

3.2.2 Seleção de Estrutura Utilizando Algoritmo Genético

No algoritmo de seleção de estrutura considerado nesse trabalho, cada indivíduo do algoritmo genético representa um vetor de estrutura candidata $\gamma = [\gamma_i, \ldots, \gamma_p]$, onde $2 \leq \gamma_k \leq \gamma_{max}$ é o número de conjuntos *fuzzy* na partição da variável x_k e um valor $\gamma_k = 1$ indica que a variável x_k não deve ser considerada pelo modelo. O valor γ_{max} deve ser fixado para cada problema para evitar que um grande número de conjuntos *fuzzy* prejudique a interpretabilidade do modelo.

Para cada solução candidata gerada pelo AG, uma base de regras é computada levando em conta as variáveis selecionadas e o número de conjuntos *fuzzy* para cada variável especificados pelo vetor de estrutura. A base de regras é gerada através do algoritmo de indução de regras (Algoritmo 1).

A base de regras gerada pelo Algoritmo 1 para cada indivíduo é avaliada de acordo com uma função de aptidão, que é baseada no balanceamento entre o desempenho do modelo e a complexidade da função de aproximação. A função objetivo a ser minimizada é o risco total $R(\gamma)$, composto pelos dois termos:

$$R(\gamma) = R_e(\gamma) + R_c(\gamma), \tag{3.12}$$

onde $R_e(\gamma)$ é o risco empírico, baseado no erro de classificação, e $R_c(\gamma)$ é o risco de complexidade que deve penalizar modelos com um grande número de regras.

Com o objetivo de ser consistente com a estimação dos parâmetros de regras, o risco empírico é calculado como a função objetivo do problema de otimização quadrático limitado definido pela Equação 3.9. O risco de complexidade é computado pela norma do vetor de pesos de regras (Equação 3.11). A função risco total é então computada pela soma dos termos correspondentes para todas as classes:

$$R(\gamma) = \frac{1}{2N} \sum_{j=1...m} \|U(\gamma)\varphi_j(\gamma) - V_j\|^2 + \frac{1}{2N} \sum_{j=1...m} \|\varphi_j(\gamma)\|^2 \tag{3.13}$$

onde o termo $\frac{1}{2N}$ é utilizado para reescalar a função risco no intervalo $[0, 1]$.

A função risco definida pela Equação 3.13 tem a forma de uma função regularizada frequentemente utilizada para a solução de problemas mal condicionados. Ela tem o efeito de balancear a precisão do modelo e a sua capacidade de acordo com a complexidade requerida pelo problema. Para um problema muito simples, o termo de risco empírico é pequeno e a seleção de estrutura irá encontrar um modelo com um pequeno número de regras para reduzir o termo de complexidade. Por outro lado, em um problema mais complexo, pode-se esperar que um modelo com um maior número de regras apresente melhor precisão mas também um maior número de regras certas, resultando em um maior valor no termo de complexidade, tal que complexidade e precisão devem "negociar"para encontrar o equilíbrio.

Uma formulação similar para a função de avaliação foi proposta por Cordón et al. [38], onde o termo de complexidade é simplesmente o número de regras e a importância relativa dos dois termos é ajustada por alguns parâmetros. Outras abordagens consideram os dois termos em um problema de otimização multi-objetivo [14] [39]. Ishibuchi e Yamamoto [39] também consideram um terceiro fator que penaliza o número de variáveis nas regras *fuzzy*.

A função de avaliação definida pela Equação 3.13 não é apenas uma função do número de regras; é principalmente uma função do número de regras certas, o qual é relacionado à complexidade do problema e, consequentemente, à precisão do modelo. A Equação 3.13 representa o termo de complexidade como uma penalidade à precisão em um único objetivo.

3.2.3 Operadores e Parâmetros Definidos para o AG para Seleção de Estrutura

O algoritmo para seleção de estrutura, que tem como saída o vetor $\gamma = [\gamma_1, \ldots, \gamma_{max}]$ foi implementado como um AG em sua forma mais simples, com operadores e codificação padrão. Durante a implementação do algoritmo genético, diferentes operadores e parâmetros comumente utilizados em algoritmos genéticos com propósito semelhante foram avaliados e, após alguns testes preliminares, os seguintes operadores e parâmetros foram adotados:

- **Tamanho da População:** Após a realização de alguns testes com variações no tamanho da população, percebeu-se que a parametrização do algoritmo genético com 32 indivíduos por população na maioria das vezes gera os melhores resultados e por isso esse valor foi adotado.

- **Abordagem para Evolução:** Foi adotada a abordagem geracional clássica para a evolução do algoritmo genético. Segundo essa abordagem, a cada geração do algoritmo genético todos os indivíduos devem ser avaliados e uma população inteiramente nova é formada através da aplicação dos operadores de cruzamento, mutação e seleção.

- **Critério de Parada:** O critério de parada utilizado foi o número de gerações, fixado em 150. Tal valor se mostrou, em todos os testes preliminares realizados, suficiente para a convergência do algoritmo genético.

- **População Inicial:** Para a formação da população inicial são gerados modelos univariados para cada um dos atributos do problema, variando o número de conjuntos *fuzzy* de 2 a $gamma_{max}$. Após, são selecionados os modelos com o número de conjuntos *fuzzy* ótimos considerando apenas cada uma das variáveis. Tal inicialização mostrou-se vantajosa em relação à inicialização tradicional pois garante um bom palpite inicial de indivíduos para a primeira geração e reduz a aleatoriedade do AG. Essa alternativa é viabilizada pelo seu baixo custo computacional, já que a geração de modelos univariados não envolve análise espectral.

- **Codificação dos Indivíduos:** Utilizou-se a codificação através de números inteiros. Tal codificação foi adotada pois, como o espaço de busca do problema de seleção de estrutura é discreto e não contínuo, a utilização de números inteiros adequa-se ao problema.

- **Seleção:** O algoritmo de roleta foi utilizado para a operação de seleção. Neste tipo de seleção, a probabilidade de um indivíduo ser escolhido como pai é diretamente proporcional à sua aptidão (quanto mais apto é o indivíduo, mais chance ele tem de gerar descendentes).

- **Cruzamento:** A operação de cruzamento acontece de forma uniforme. Assim, a carga genética de cada gene de um determinado indivíduo descendente pode ser herdada, aleatoriamente, de um dos indivíduos pais. A probabilidade de cruzamento foi fixada em 0.85.

- **Mutação:** Foi utilizado um operador de mutação com probabilidade de alteração em cada gene fixada em 0.05. Um gene que sofre mutação recebe um valor aleatório entre 1 e γ_{max}.

- **Valor Máximo de γ_i:** O valor máximo permitido para um gene, que representa o número de conjuntos *fuzzy* máximo para cada variável foi fixado em $\gamma_{max} = 8$.

Estratégia para Evitar Reavaliação de Indivíduos

Durante a execução do algoritmo genético para seleção de estrutura no FSM, em diversos momentos são gerados indivíduos repetidos, ou seja, que já foram gerados e avaliados em gerações anteriores. Por isso, a reavaliação desses indivíduos implica em um desperdício de recurso computacional.

Esse trabalho implementa um esquema para evitar a avaliação desnecessária de indivíduos. Sempre que um indivíduo é avaliado, seu valor de aptidão é armazenado. Dessa forma, a função de avaliação precisa ser executada apenas quando o valor de aptidão de um indivíduo não estiver armazenado, evitando assim a reavaliação de indivíduos e o consequente desperdício de recursos computacionais. A Figura 3.5 apresenta a representação esquemática dessa idéia para um exemplo com quatro variáveis.

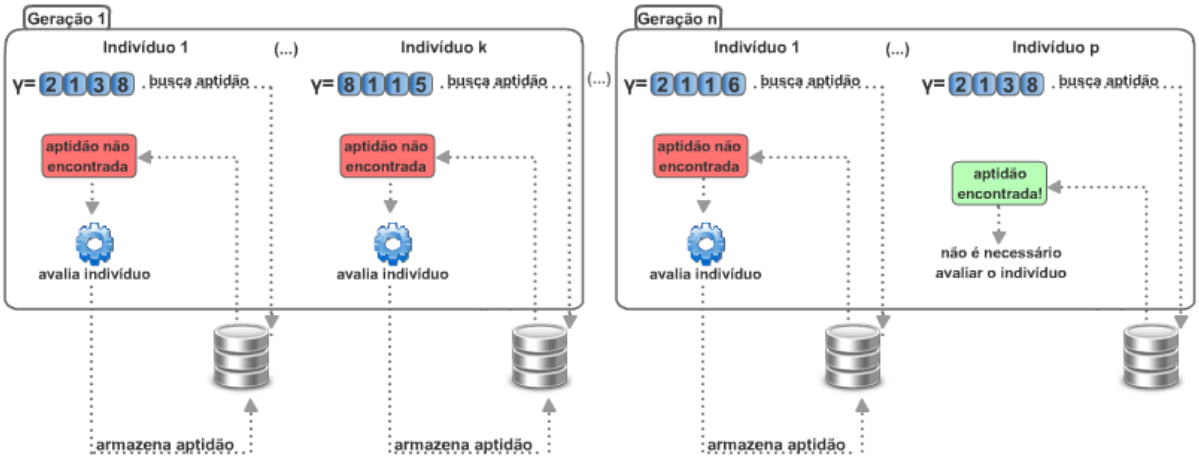

Figura 3.5: Representação esquemática da estratégia para evitar reavaliação de indivíduos.

A análise espectral realizada pelo algoritmo de indução de regras (Algoritmo 1) leva em consideração apenas a seleção de variáveis. Ou seja, na análise espectral o número de conjuntos *fuzzy* na partição de cada variável não é importante. É relevante apenas o fato de uma variável ser ou não considerada pelo modelo. Por isso, análises espectrais desnecessárias ocorrem com bastante frequência e sua reexecução é também evitada nesse trabalho. Assim, sempre que uma análise espectral é executada, a matriz $\widehat{Z}$ (com os K autovetores correspondentes aos maiores autovalores calculados pela Equação 3.4) é armazenada. Uma nova análise espectral só é executada quando sua respectiva matriz $\widehat{Z}$ não está armazenada. A Figura 3.6 apresenta a representação esquemática da estratégia para evitar reavaliação de indivíduos e da estratégia para evitar reexecução de análise espectral em um exemplo com quatro variáveis.

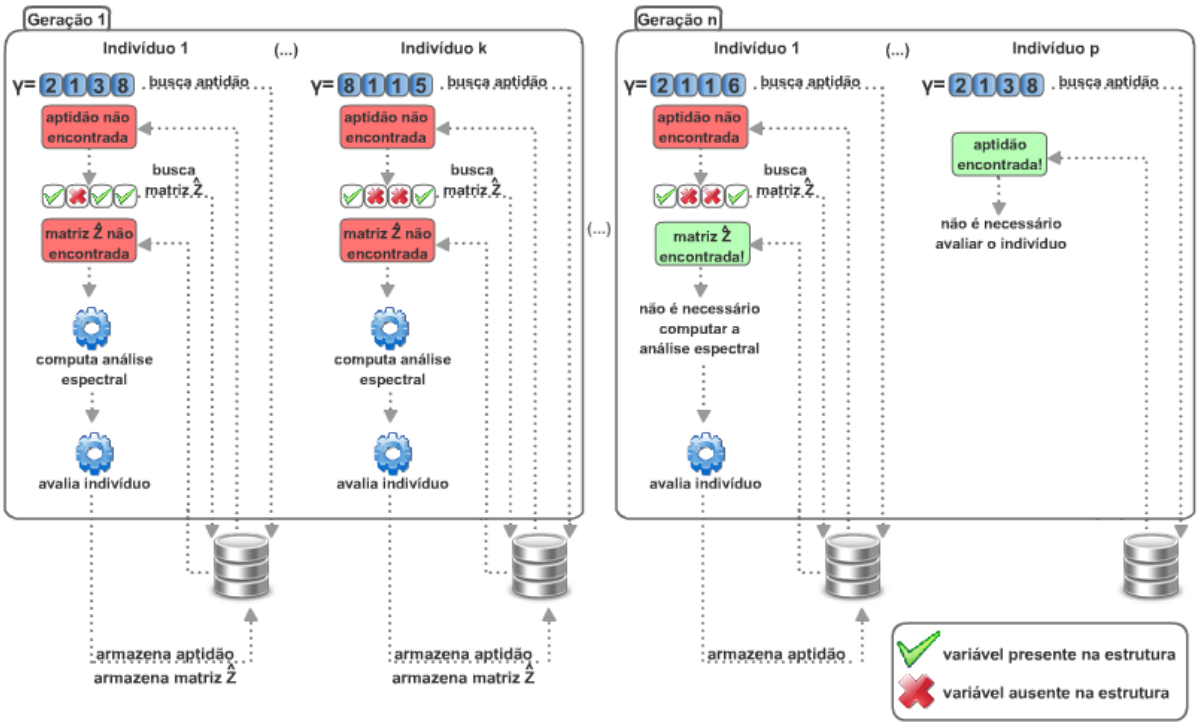

Figura 3.6: Representação esquemática da estratégia para evitar reavaliação de indivíduos e reexecução de análise espectral.

Capítulo 4

Implementações Distribuídas

O algoritmo genético para seleção de estrutura, apresentado anteriormente é uma tarefa que demanda muito poder de processamento, já que cada avaliação de candidato requer a solução da Equação 3.4, cuja complexidade é $O(N^3)$ para processamento e $O(N^2)$ para armazenamento, onde N é o número de registros da base de dados.

O tempo de execução do sistema para extração de modelos *fuzzy* apresentado nesse trabalho é uma de suas principais desvantagens, como foi constatado em [5] e [4]. Por isso, nesses trabalhos, propõe-se o estudo de medidas para diminuição do tempo de processamento global do método.

Em [40] são apresentadas algumas medidas para diminuição de tempo neste tipo de aplicação.

Uma delas é realizar uma melhoria do algoritmo que deseja-se processar, o que muitas vezes pode ser inviável ou até impossível.

Outra medida apresentada em [40] é a utilização de computadores com processadores mais rápidos. Durante muito tempo essa estratégia, apesar de bastante custosa, era viável devido a uma tendência observada por Gordon Moore, à qual se deu o nome de Lei de Moore [41]. Publicada em 1965, a Lei de Moore diz que, a cada período de 18 meses o poder de processamento dos *chips* aumentaria em 100%. Esse aumento se dá, basicamente, pelo aumento do número de transistores por *chip*, possibilitado pela diminuição do tamanho de cada transistor. Entretanto, o tamanho atual dos transistores já está muito próximo do tamanho de átomos, ou seja, o poder de processamento de processadores de silício está próximo do limite. A indústria de desenvolvimento de *hardware* atualmente trabalha para o desenvol-

vimento de tecnologias alternativas ao silício na a fabricação de processadores, mas essas tecnologias ainda estão longe do mercado de computadores pessoais.

Uma terceira estratégia apresentada em [40] é a divisão da execução entre múltiplas unidades de processamento, permitindo a execução simultânea de partes da aplicação [42]. Ou seja, de forma coerente e transparente, diversos processadores computam colaborativamente determinada tarefa, como se, em um nível mais alto da aplicação, apenas um único e centralizado processador estivesse executando a tarefa [43]. Apesar de estar sendo objeto de estudos há algum tempo a adoção de tal estratégia esteve bastante restrita ao ambiente acadêmico. Entretanto, a crescente disponibilidade de redes de alta velocidade e o baixo custo de computadores pessoais têm popularizado a utilização *clusters* de computadores fora do ambiente acadêmico.

Atualmente como tentativa de contornar o provável fim da Lei de Moore, as empresas de *hardware* têm difundido o uso de máquinas multi-processadas e diversas dessas empresas têm se unido com o objetivo de promover o desenvolvimento de *softwares* que tirem proveito de tal tecnologia. Uma dessas iniciativas consiste na criação dos Centros Universais de Pesquisas em Computação Paralela (*Universal Parallel Computing Research Center*) [44]. Essa é uma parceria entre a Intel, a Microsoft e diversas instituições de ensino que tem como objetivo acelerar o desenvolvimento de aplicações comerciais, tanto para consumidores quanto para empresas, utilizando computação paralela. Ou seja, pode-se observar no mercado uma forte tendência na disseminação de computadores com múltiplas unidades de processamento e a utilização do paralelismo como forma de diminuição do tempo de processamento em aplicações computacionais tem se mostrado cada vez mais vantajosa.

Esse trabalho utiliza o processamento distribuído como forma de diminuir o alto tempo de processamento global requerido para a solução do algoritmo de indução de regras (Algoritmo 1) para cada estrutura candidata avaliada pelo algoritmo genético.

4.1 Paralelismo em Métodos de Mineração de Dados

Pode-se encontrar na literatura outros trabalhos que utilizam a estratégia de paralelização em sistemas de mineração de dados. Em [45], Costa & Ebecken apresentam uma rede neural para classificação em paralelo utilizando uma estratégia que consiste na divisão dos dados entre os processadores. Cada processador guarda uma cópia de toda a arquitetura da rede.

Zaki et. al [46] apresentam a implementação em paralelo de uma árvore de decisão utilizando duas estratégias diferentes. Em uma delas, o particionamento das sub-árvores é dividido entre os processadores dinamicamente. Na outra estratégia, cada processador é responsável pela avaliação de um atributo (ou um conjunto de atributos).

Em [47], Araujo et. al apresentam um método para a descoberta de regras do tipo se-então em uma base de dados utilizando algoritmos genéticos. Tal AG é implementado de forma que cada indivíduo represente uma regra candidata. A população é dividida em diversas sub-populações e cada uma destas é atribuída a um processador. Assim, a evolução das diferentes sub-populações acontece simultaneamente. Essa estratégia de paralelização de algoritmos genéticos será melhor explorada mais adiante.

Modenesi et al. [48] realizam o estudo da paralelização de um método de agrupamento utilizando lógica *fuzzy*, *fuzzy c-means*. Cada processador efetua a computação do algoritmo *fuzzy c-means* sobre o mesmo conjunto de dados mas considerando diferentes números de grupos. Um índice de validação avalia o melhor particionamento entre os resultados obtidos pelos processadores e o melhor deles é escolhido como o particionamento final.

Nojima et al. [49] realizam um estudo sobre a paralelização de um sistema baseado em regras *fuzzy* induzidas por um algoritmo genético. Segundo essa abordagem, o AG é paralelizado e cada processador é responsável pela avaliação de uma parte dos indivíduos. Além disso, os dados também são divididos entre os processadores. Dessa forma, cada processador fica encarregado pela avaliação de uma parte dos indivíduos da população e cada um dos indivíduos classifica apenas uma parte do

conjunto de dados.

Nesse trabalho, duas estratégias de paralelismo para o FSM são avaliadas. Na primeira, apenas o algoritmo genético é implementado em paralelo e cada processador é responsável pela avaliação de uma parte da população segundo a Equação 3.13. Na segunda, além do AG, a análise espectral contida no Algoritmo 1 na avaliação de cada individuo também é implementada em paralelo e, assim, um grupo de processadores é responsável pela avaliação de uma parte da população.

4.2 Computação Paralela

4.2.1 Níveis de Paralelismo

O paralelismo nas aplicações pode ser explorado em diferentes níveis, dependendo do problema trabalhado e dos recursos de *hardware* disponíveis. De uma forma geral, a exploração de paralelismo em um algoritmo pode ser feita através do particionamento do problema em respeito a dados e tarefas [50].

- Paralelismo de dados: Uma mesma tarefa é alocada aos diversos processadores, sendo que cada um deles trabalha com uma porção de dados diferente do problema. Esta abordagem aplica-se a problemas cuja solução envolve grandes massas de dados. Alguns problemas clássicos desse tipo são a solução de sistemas de equações, a multiplicação de matrizes, a integração numérica e os problemas de autovalores.

- Paralelismo de tarefas: Tarefas distintas são alocadas aos diversos processadores (ou a grupos deles). Esta abordagem aplica-se a problemas onde pode haver uma independência entre blocos de execução, como é o caso dos algoritmos genéticos e do problema do produtor-consumidor.

4.2.2 Modelos de Programação Paralela

Para a construção de um programa paralelo, deve-se pensar no modo de comunicação entre os processadores envolvidos na sua execução. Existem, basicamente, dois paradigmas de comunicação: memória compartilhada e troca de mensagens.

Memória Compartilhada

O modelo de programação paralela utilizando memória compartilhada baseia-se no mesmo princípio utilizado na programação sequencial. Essencialmente, um único espaço de endereçamento é utilizado, e esse pertence ao fluxo de execução do programa. Por causa disso, o compartilhamento de memória é um paradigma mais natural e de mais fácil assimilação para programadores não habituados à programação paralela, já que a noção de compartilhamento de memória faz parte da cultura dos desenvolvedores de aplicações sequenciais [51].

A utilização de memória compartilhada adequa-se a computadores onde uma única memória é acessada por vários processadores. Dessa forma, a sincronização entre as tarefas é feita através de leitura/escrita na memória compartilhada. Normalmente uma mesma parte da memória não pode ser modificada por um processo enquanto outro a estiver acessando. Por isso faz-se necessária a utilização de rotinas de sincronização e de seção crítica. Mais detalhes sobre esses conceitos podem ser obtidos em [52].

Uma grande vantagem na utilização de memória compartilhada é a alta velocidade de comunicação entre as tarefas. Entretanto, a escalabilidade de aplicações desenvolvidas com memória compartilhada é limitada pelo número de barramentos disponíveis entre os processadores e a memória [51]. No caso de arquiteturas com memória fisicamente distribuída, a programação por compartilhamento de memória exige uma camada de *software* que forneça a abstração de memória compartilhada [40]. Essa abordagem é denominada memória compartilhada distribuída, ou seja, a memória é fisicamente distribuída, mas logicamente compartilhada.

Troca de Mensagens

Conceitualmente, a idéia de troca de mensagens é totalmente independente de *hardware*, sistema operacional, linguagens de programação e bibliotecas. Em um programa paralelizado via troca de mensagens deve-se distribuir os dados e/ou tarefas explicitamente entre os processadores . Como os espaços de endereçamento dos processos que compõem o programa paralelo são distintos, utiliza-se a abstração de mensagem, que pode ser enviada de um processo a outro através de um canal de comunicação [43].

O paradigma de troca de mensagens tem sido tradicionalmente empregado em sistemas fracamente acoplados, implementados em arquiteturas baseadas em memória distribuída, em que cada processador tem acesso somente à sua memória local. A comunicação em aplicações que utilizam troca de mensagens é feita através de primitivas que informam que um processo pretende enviar uma mensagem a outro, que espera recebê-la [53]. Na prática, existem bibliotecas que fornecem, entre outras funções mais complexas, esse tipo de primitivas de envio e recebimento de mensagens. Entre essas bibliotecas, uma das mais completas é a MPI.

O MPI (*Message Passing Interface*) [54] é um sistema de passagem de mensagens padronizado e portável, que funciona em uma grande variedade de computadores paralelos. A biblioteca MPI foi desenvolvida para ambientes de memória distribuída, máquinas paralelas, rede de estações de trabalho e redes heterogêneas.

MPI define um conjunto de rotinas para facilitar a comunicação (troca de dados e sincronização) entre processos paralelos. É portável para qualquer arquitetura e possui diversas funções para programação e ferramentas para se analisar o desempenho da aplicação. Além disso, o MPI abrange um conjunto de características avançadas como estruturas de dados definidas pelo usuário, portas de comunicação persistentes, operações de comunicação coletiva poderosas e contextos de comunicação.

A biblioteca MPI possui rotinas para programas em Fortran 77 e ANSI C, portanto pode ser usada também para Fortran 90 e C++. Os algoritmos programados são compilados e ligados à biblioteca MPI. Várias linguagens de programação, como Java e Python oferecem aos programadores soluções para utilização de MPI através de bibliotecas de ligação indireta entre a aplicação em alto nível e a biblioteca MPI.

Em programas que utilizam MPI, todo paralelismo é explícito, ou seja, o programador é responsável por identificar o paralelismo e implementar os pontos de sincronização e troca de mensagens utilizando chamadas às funções da biblioteca MPI. Cada processo, identificado pelo seu IP ou pelo nome da máquina na rede e recebe uma réplica do código do programa paralelo como um todo. O trecho de código particular de cada processo (se houver) é identificado por um comando onde a condição envolve a sua identificação [53].

4.2.3 Limitações da Computação Distribuída

O ganho de desempenho em uma aplicação distribuída pode ser medido através do *speedup*:

$$speedud = \frac{T_{seq}}{T_{par}}, \tag{4.1}$$

onde T_{seq} é o tempo de execução sequencial, isto é, computada em apenas um processador, e T_{par} é o tempo de execução da aplicação em vários processadores.

Embora a computação distribuída tenha muito a oferecer, existe um limite no possível *speedup* de uma aplicação pela adição de novos computadores. Em uma situação ideal, espera-se que com n máquinas a computação aconteça n vezes mais rapidamente do que seria com apenas uma máquina[40]. Nessa situação diz-se que a aplicação paralela obteve um *speedup* linear.

Qualquer cálculo pode ser dividido em blocos de códigos ou instruções que podem ser classificados em uma das duas maneiras: ou um bloco de código pode ser paralelizado e dividido entre n máquinas, ou o código é essencialmente sequencial e as instruções devem ser executadas na ordem em que elas foram escritas. Qualquer código sequencial não será beneficiado pela adição de processadores [55].

Existem diversas razões para que um código não possa ser paralelizado e deva ser executado segundo uma ordem específica. Uma dessas razões vem da dependência entre os dados durante o código. Por exemplo, se é preciso utilizar o valor de x para o cálculo de y, então o cálculo de x deve ser feito antes do cálculo de y [40]. Basicamente, para que se possa paralelizar duas instruções, nenhuma das duas pode depender da outra, isto é, a ordem em que as duas instruções terminam de ser executadas não deve ser relevante.

Desta forma, qualquer programa pode ser visto como uma série alternada de seções que podem ser paralelizadas e efetivamente executada em diferentes máquinas intercaladas por seções que serão executadas em uma única máquina. Assim, a quantidade de código que deve ser executada sequencialmente limita o ganho de velocidade (*speedup*) que se pode esperar de uma aplicação paralela. Essa idéia corresponde à da lei de Amdahl [56].

Lei de Amdahl

No fim dos anos 60, Gene Amdahl em seus estudos formalizou a idéia do limite de uma porção sequencial de um código ao *speedup* em uma execução paralela através do que, hoje, é conhecido como lei de Amdahl. A lei de Amdahl afirma que a porção sequencial de um programa será o fator limitante a quanto pode-se aumentar a velocidade de uma execução de um programa utilizando múltiplos processadores [55].

Embora a lei de Amdahl seja a métrica mais conhecida e mais utilizada para descrever a limitação do desempenho paralelo, existem outras como as métricas de Gustafson-Barsus, de Sun, e de Ni [56].

Para ilustrar a lei de Amdahl pode-se imaginar uma situação hipotética em que se tenha tantos computadores que o código poderia ser executado instantaneamente. Ainda assim o programa deverá executar a porção sequencial do código, o que será feito em um único computador, e o tempo global da execução será o tempo de execução da porção sequencial [40].

O fator de limitação de *speedup* é então definido como:

$$\text{speedud}_{max} = \frac{1}{C_s + \dfrac{C_p}{N_{proc}}} \tag{4.2}$$

onde C_s é a fração de código sequencial, C_p é a fração de código que pode ser paralelizado e N_{proc} é o número de processadores disponíveis. Assim, se $s = 0,1$, por exemplo, pode-se esperar um fator de *speedup* máximo de 10, não importando a quantidade de processadores disponíveis.

Deve-se também lembrar que a lei de Amdahl é apenas um ideal. Na prática, existe um *overhead* introduzido pela paralelização do código. Por exemplo, a coordenação da comunicação entre os vários processos requer codificação adicional, que aumenta o tempo de execução global. Há também a contenção da rede ou de uma memória compartilhada, que podem suspender a execução de um processo, atrasando sua execução [40]. Assim, a lei de Amdahl define o melhor *speedup* que se pode esperar de uma aplicação paralela.

4.3 Algoritmos Genéticos Paralelos

Algumas características dos algoritmos genéticos permitem que a sua paralelização se dê de forma bastante natural [57]. Entre essas características pode-se destacar:

- O valor de aptidão de cada indivíduo numa população é independente de fatores externos a esse indivíduo;
- A aplicação dos operadores genéticos pode ser feita em qualquer ordem a qualquer indivíduo da população.

A estratégia de divisão de tarefas utilizada pode ser aplicada em algoritmos genéticos de várias maneiras. Alguns métodos de paralelização utilizam uma única população, enquanto outros dividem a população em diversas subpopulações relativamente isoladas [58] [59]. Nesse trabalho, adota-se a classificação apresentada por Cantú Paz em [60], que divide a população dos algoritmos genéticos em: (1) população única e global, (2) população única de granularidade fina, (3) população múltipla de granularidade grossa e (4) modelos híbridos. Esses modelos são melhor apresentados a seguir.

4.3.1 Algoritmo Genético Mestre-Escravo

Nos algoritmos genéticos paralelos do tipo mestre-escravo uma única população é utilizada e a avaliação de indivíduos e/ou aplicação dos operadores genéticos é feita em paralelo. Como em um AG sequencial, cada indivíduo deve competir e ser operado com qualquer outro indivíduo da população, e por isso algoritmos desse tipo são também conhecidos como algoritmos genéticos paralelos globais.

Como a aptidão de um indivíduo é independente do resto da população, essa é a operação mais comumente paralelizada em algoritmos genéticos mestre-escravo. A avaliação dos indivíduos é paralelizada pela atribuição de uma fração da população a cada um dos processadores disponíveis e a comunicação acontece apenas quando cada processador recebe seu subconjunto de indivíduos para avaliar e quando retorna seus valores de aptidão.

Assim, nesse tipo de algoritmo genético paralelo, há um processo central (mestre) que armazena todos os indivíduos da população e é responsável pela distribuição dos

indivíduos aos outros processos (escravos), que são responsáveis por calcular o valor de aptidão de cada indivíduo. Geralmente é tarefa do processo mestre a aplicação dos operadores de seleção, mutação e cruzamento à população. Esse esquema pode ser melhor visualizado na Figura 4.1.

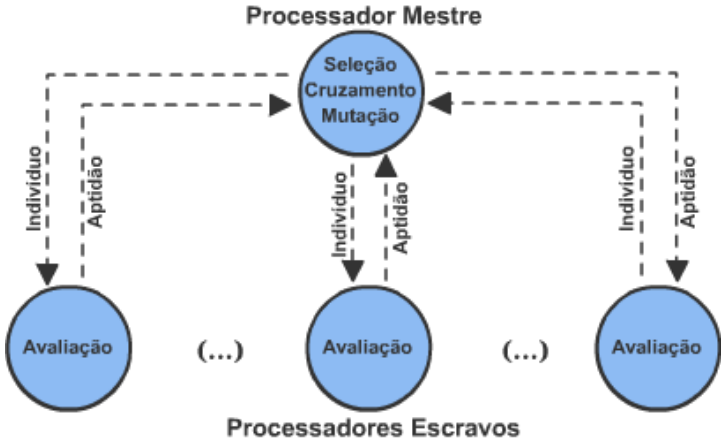

Figura 4.1: Algoritmo genético paralelo mestre-escravo.

Se a cada geração o AG aguarda o recebimento dos valores de aptidão de toda população antes de prosseguir, então o algoritmo é síncrono e tem exatamente as mesmas propriedades de um AG sequencial, diferenciado apenas pelo tempo de execução. Entretanto, é possível que se implemente algoritmos genéticos mestre-escravo de maneira assíncrona, onde o algoritmo não aguarda o valor de aptidão de todos os processo, mas esse tipo de abordagem não funciona exatamente como um AG sequencial.

O modelo de paralelização global não assume nada sobre a arquitetura do computador paralelo em que é executado, e pode ser implementado eficientemente tanto em computadores com memória compartilhada quanto em computadores com memória distribuída. Em computadores com memória distribuída, o processo mestre deve ser responsável por explicitamente enviar os indivíduos aos processos escravos para avaliação e coletar os resultados.

Nesse modelo de paralelização deve-se ter como preocupação a manutenção do balanceamento de carga computacional entre os processos através da utilização de esquemas de escalonamento dinâmico.

Outro aspecto do algoritmos genéticos mestre-escravo que pode ser paralelizado

é a aplicação dos operadores genéticos. Cruzamento e mutação podem ser paralelizados utilizando a mesma idéia do particionamento da população e da distribuição do trabalho entre os vários processos. Entretanto, esses operadores são tão simples que geralmente o tempo requerido para o envio dos indivíduos entre os processos pode eliminar qualquer ganho de desempenho.

4.3.2 Granularidade Fina

Algoritmos genéticos paralelos de granularidade fina utilizam apenas uma população, mas há uma estrutura espacial que limita a interação entre os indivíduos. Um indivíduo pode apenas competir e ser operado com seus vizinhos, mas a sobreposição das vizinhanças permite que boas soluções possam ser disseminadas por toda a população. A Figura 4.2 apresenta uma visão esquemática de um modelo de granularidade fina definido por uma topologia de grade, onde cada nó representa um processo. O caso ideal nesse modelo é que cada processo corresponda a um indivíduo.

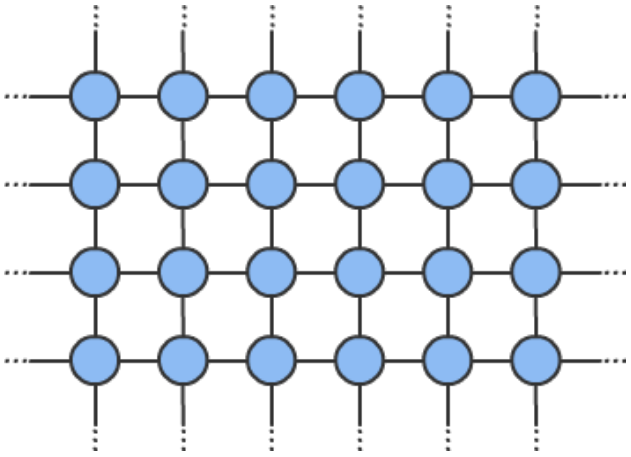

Figura 4.2: Algoritmo genético paralelo de granularidade fina.

É comum distribuir os indivíduos de um algoritmo genético paralelo em uma grade bidimensional, já que em muitos computadores massivamente paralelos, os elementos de processamento são conectados com essa topologia.

4.3.3 Granularidade Grossa

Modelos de algoritmos genéticos paralelos com múltiplas populações de granularidade grossa consistem na divisão da população em diversas subpopulações, que

evoluem independentemente e trocam indivíduos ocasionalmente. Essa troca de indivíduos é denominada migração e é controlada por diversos parâmetros. Algoritmos genéticos paralelos com múltiplas populações são conhecidos na literatura por diferentes nomes como algoritmos genéticos distribuídos ou modelo de ilha. A Figura 4.3 mostra uma representação esquemática desse tipo de modelo.

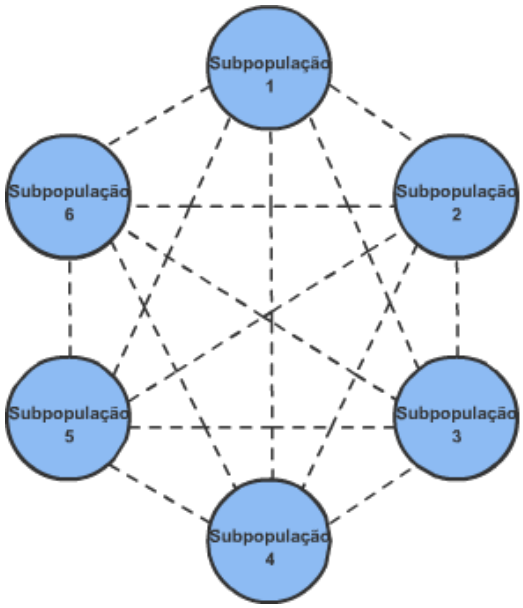

Figura 4.3: Algoritmo genético paralelo de granularidade grossa.

Como o tamanho das populações nesse modelo são menores do que em algoritmos genéticos sequenciais, espera-se que convirjam mais rapidamente. Entretanto, quando compara-se o desempenho do modelo de granularidade grossa com algoritmos sequenciais deve-se também levar em conta que a qualidade da solução em cada ilha tende a ser mais pobre.

O modelo de ilhas tem como inspiração principal a evolução das espécies na natureza. A observação das espécies que se desenvolvem em ambientes isolados mostra que muitas delas desenvolvem uma adaptação muito grande a particularidades de seus ambientes.

Os algoritmos genéticos de granularidade grossa têm uma sequência de execução muito parecida com algoritmos genéticos sequenciais. Cada subpopulação evolui da maneira tradicional, diferenciando-se apenas pelo operador de migração dos elementos entre as diferentes ilhas.

A operação de migração ocorre numa frequência não muito grande, fazendo com

que a sobrecarga causada pela comunicação entre os processos seja aceitável. Assim, esse tipo de modelo é muito bem aplicada em sistemas paralelos onde a rede não é muito rápida.

4.3.4 Modelos Híbridos

Uma outra alternativa para modelagem de algoritmos genéticos paralelos é a combinação de dois modelos de paralelização, produzindo algoritmos genéticos paralelos hierárquicos. Quando dois métodos de paralelização de algoritmos genéticos são combinados eles formam uma hierarquia, na qual, no nível mais alto estão os algoritmos de múltiplas populações.

O nível inferior pode ter um AG de granularidade fina, aproveitando-se o poder de computadores massivamente paralelos, como é esquematizado na Figura 4.4.

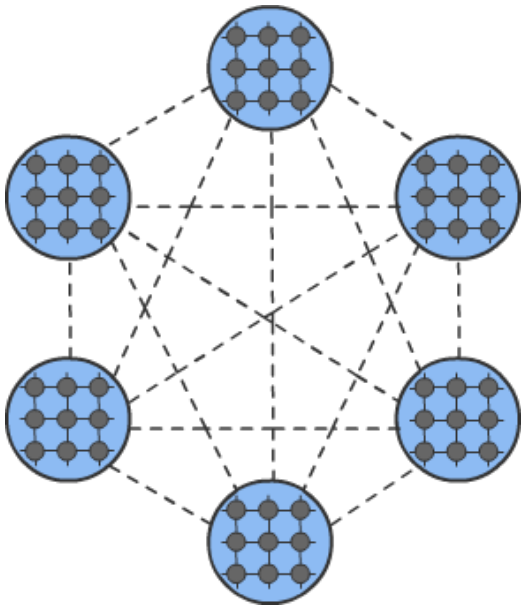

Figura 4.4: Algoritmo genético paralelo híbrido com granularidade fina.

Uma outra abordagem é utilizar o modelo mestre-escravo no nível inferior, como é esquematizado na Figura 4.5. Assim, cada ilha corresponde a um conjunto de processadores.

Pode-se também utilizar modelos com múltiplas populações em ambas as camadas, como é esquematizado na Figura 4.6. Assim, pode-se forçar a mistura dos indivíduos no nível inferior com uma taxa de migração alta, enquanto uma taxa de

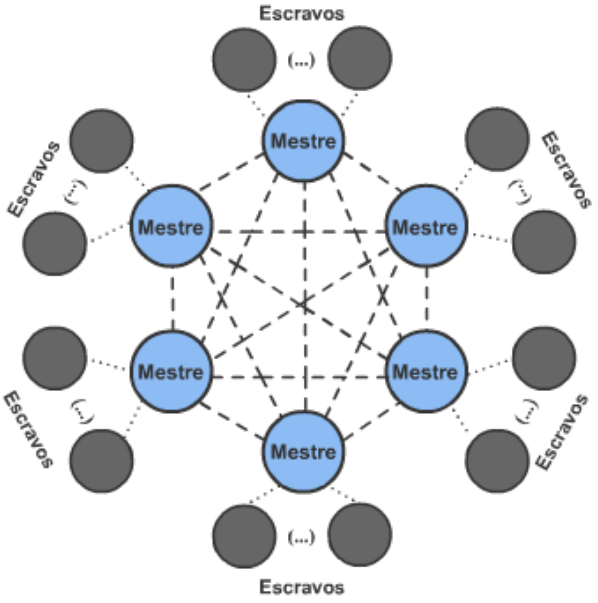

Figura 4.5: Algoritmo genético paralelo híbrido com mestre-escravo.

migração baixa é utilizada no nível mais baixo.

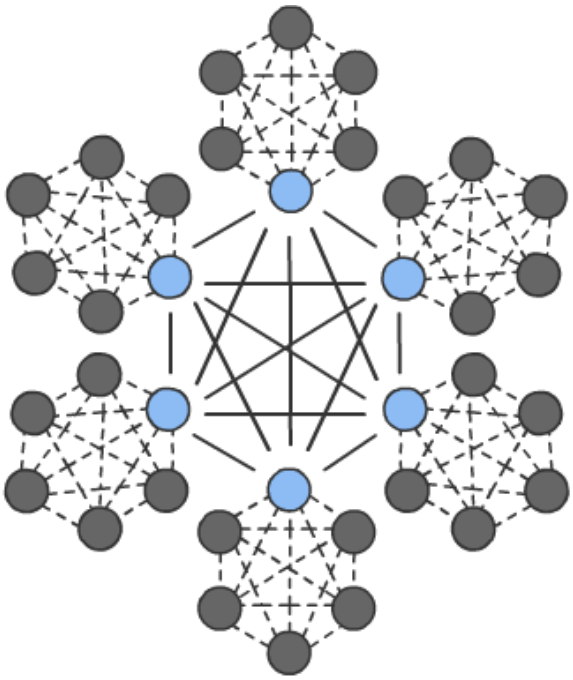

Figura 4.6: Algoritmo genético paralelo híbrido com granularidade grossa.

4.3.5 Implementação Paralela do Algoritmo Genético para Seleção de Estrutura

Nesse trabalho, a implementação do algoritmo genético para seleção de estrutura em paralelo para foi feita segundo a abordagem mestre escravo. Tal modelo foi adotado, pois sua implementação é independente da arquitetura da máquina e do modelo de comunicação utilizado.

Assim, inicialmente, um processador mestre realiza os procedimentos iniciais do FSM, como o carregamento da base de dados, a inicialização das estruturas de dados e a preparação para a solução do problema. Em seguida o processador mestre inicializa a população do AG e realiza a avaliação da população inicial. É também tarefa do processador mestre realizar as operações de seleção, cruzamento e mutação e enviar os indivíduos para os processadores escravos. Os escravos calculam o valor de aptidão de uma parte da população. O processador mestre também realiza a avaliação de uma parte da população. Dessa forma o processador mestre não fica ocioso durante a fase de avaliação dos indivíduos e todo o recurso computacional disponível é aproveitado. Cada processador realiza a avaliação dos indivíduos sequencialmente, ou seja, um indivíduo é avaliado por vez.

Como a implementação do algoritmo genético foi feita com a linguagem de programação Python (como será descrito na Seção 5.1), a implementação do paralelismo no AG foi possível graças ao pacote MPI4Py [61], construído no topo da biblioteca MPI e que provê uma interface entre o padrão MPI à linguagem de programação Python.

4.4 Análise Espectral Distribuída

Durante a execução do algoritmo de seleção de estrutura, a avaliação de cada indivíduo candidato envolve uma execução do Algoritmo de Indução de Regras (Algoritmo 1) onde a solução da Equação 3.4 requer a solução de um problema de autovalor.

Segundo [62], a computação de autovalores em aplicações reais é sempre um problema de larga escala, considerando-se o processamento e o uso de memória requeridos por este tipo de problema. Por essa razão deve-se focar a atenção no

aumento de velocidade e melhor utilização de memória nesse tipo de problema. e, por isso, a abordagem paralela é uma maneira muito interessante para essa situação.

Nesse trabalho, um modelo de paralelização em dois níveis é avaliado. Em um nível mais alto está o paralelismo do algoritmo genético, onde o mestre realiza as operações sobre a população e envia uma certa quantidade de indivíduos para cada escravo. Em um outro nível está o paralelismo da análise espectral, onde cada escravo divide o cálculo da análise espectral com outros processadores.

4.4.1 Principais Bibliotecas para a Solução Paralela de Problemas de Autovalor

Devido ao alto custo computacional associado à solução de problemas de autovalor, a paralelização desses métodos tem recebido atenção. Uma medida bastante adotada para a solução paralela de problemas de autovalor é a utilização de bibliotecas de *software*, que fornecem aos usuários implementações robustas desenvolvidas por especialistas [62].

Apesar disso, no caso de problemas de autovalor, a utilização de bibliotecas de *software* não é uma tarefa simples e requer um conhecimento do problema para a escolha da biblioteca e do algoritmo adequados. Além disso, até que o problema seja resolvido com sucesso, deve-se realizar vários ciclos de testes e ajustes de parâmetros.

As próximas subseções apresentam algumas das bibliotecas de *software* mais utilizadas para a solução de problemas de autovalor em paralelo e suas principais características.

ScaLAPACK [63]

ScaLAPACK (*Scalable LAPACK*) é uma biblioteca de álgebra linear de alto desempenho com rotinas para computadores com memória distribuída. É uma continuação do projeto LAPACK [64], uma biblioteca escrita em Fortran77 que contém rotinas para solução de sistemas lineares, problemas de mínimos quadrados, e problemas de autovalor principalmente com matrizes densas.

O projeto ScaLAPACK é um esforço colaborativo entre várias instituições (Rice University, University of California, University of Illinois, entre outras) e tem como objetivos eficiência, escalabilidade, portabilidade e flexibilidade. Esses objetivos

foram alcançados com sucesso, limitando a dependência das máquinas às bibliotecas BLAS [65] e BLACS [66]. O *software* pode ser baixado gratuitamente na *web* (`http://www.netlib.org/scalapack/`).

A biblioteca de *software* ScaLAPACK faz uso de passagem de mensagem explícita e foi desenvolvida para computadores heterogêneos que suportam MPI ou PVM.

Assim como no LAPACK, as rotinas do ScaLAPACK são baseadas em algoritmos particionados em blocos, com o objetivo de se minimizar a movimentação de dados entre os diversos níveis de hierarquia de memória.

P_ARPACK [67]

P_ARPACK (*Parallel Arpack*) é uma versão paralela do *software* ARPACK (*Arnoldi Package*) [68], que é uma coleção de rotinas escritas em Fortran77 para a solução de problemas de autovalor em larga escala. O *software* é capaz de calcular apenas alguns autovalores da matriz com características específicas tais como aqueles de maior parte real ou maior magnitude. ARPACK implementa o método de Arnoldi, usado para solução de problema de autovalor em matrizes grandes e esparsas.

A biblioteca P_ARPACK é fornecida como uma extensão da biblioteca ARPACK e é focada em sistemas com memória distribuída e passagem de mensagem. O P_ARPACK está disponível para livre distribuição na *web* no endereço ftp://ftp.caam.rice.edu/pub/software/ARPACK.

Algumas das características importantes que são apresentadas em [67] são:

- Habilidade de retornar k autovalores que satisfaçam um critério especificado pelo usuário.
- Um requisito de armazenamento pré-determinado satisfaz o processamento. Usualmente esse requisito é $n.\theta(k) + \theta(k^2)$ onde k é o número de autovalores a ser computado e n é a ordem da matriz.
- Autovetores e/ou vetores de Schur são também computados quando requisitados.
- A precisão numérica dos autovalores e autovetores computados é especificada pelo usuário.

- Permite o uso de técnicas de deflação sem grandes dificuldades computacionais.

SLEPc [69]

O SLEPc (*Scalable Library for Eigenvalue Problem Computation*) é um pacote de *softwares* para a solução de grandes problemas de autovalor em computadores paralelos. Trabalha bem tanto com matrizes densas quanto esparsas e pode ser usado também para a solução de problemas de decomposição em valores singulares (SVD) e trabalha tanto com aritmética real ou complexa. É desenvolvido pelo *High Performance Networking and Computing Group* da Universidade Politécnica de Valencia. Uma descrição introdutória sobre o SLEPc pode ser encontrada em [70].

O pacote disponibiliza diversos métodos, tais como Arnoldi, Lanczos e iteração de subespaço. Em adição aos seus próprios *solvers*, o SLEPc provê acesso transparente a outros pacotes de *software* externos, como ARPACK, BLZPACK e TRLAN.

O SLEPc é distribuído livremente na *web* e foi construído no topo do PETSc (*Portable, Extensible Toolkit for Scientific Computation*) [71], que é um conjunto de bibliotecas de estruturas de dados e rotinas para soluções paralelas de aplicações científicas. Sendo assim, o SLEPc pode ser considerado uma extensão do PETSc que provê as funcionalidades necessárias para a solução de problemas de autovalor. Isso significa que o PETSc deve estar previamente instalado para que se use o SLEPc.

A biblioteca de *software* SLEPc, assim como o PETSc, lida com a passagem de mensagem entre os processadores, fazendo com que o usuário tenha que se preocupar com o paralelismo apenas em um nível mais alto. Além disso, o programador tem acesso às estruturas de dados em alto nível, não tendo que se preocupar com níveis mais baixos destas estruturas.

Ambos SLEPc e PETSc implementam vários conceitos de orientação a objetos, como encapsulamento, herança e polimorfismo. Por exemplo, o usuário chama apenas uma rotina de interface genérica, que então manipula corretamente a estrutura de dados em particular. Pode-se também, por exemplo definir uma classe de matrizes com alguns operadores e esquemas de armazenamento específicos, mas que herdam características do objeto matriz base.

O SLEPc oferece ao usuário controle total sobre o processo de solução e pode ser utilizado com as linguagens de programação C, C++ e Fortran.

4.4.2 Implementação da Análise Espectral no FSM

Nesse trabalho, a solução do problema de autovalor 3.4 contido no algoritmo de indução de regras (Algoritmo 1) foi realizada com o auxílio da biblioteca SLEPc, descrita na Seção 4.4.1. O SLEPc foi adotado devido à sua livre disponibilidade na *web* e ao grande número de materiais de documentação que auxiliam a sua utilização.

Foi utilizado o *solver* Krylov-Schur, que implementa o método de Arnoldi para solução de problemas de autovalor e, segundo os próprios autores da biblioteca, é provavelmente o *solver* para problemas de autovalor mais robusto do SLEPc [72]. Esse *solver* implementa um método de projeção de Krylov-Schur para problemas de autovalor. Métodos de projeção têm como objetivo computar apenas uma solução parcial do problema de autovalor 3.4. Ou seja, dada a matriz L (de ordem N), o objetivo é computar um pequeno número de autovalores selecionados K. Dessa forma, pode-se computar apenas os autovalores positivos, como descrito na Seção 3.1.1. Entretanto, para evitar pequenas flutuações em torno de zero, fixa-se um valor δ e calcula-se os autovalores $\lambda > \delta$. Nesse trabalho, δ foi fixado em $\delta = 0.01$.

4.4.3 Implementação da Abordagem Paralela em Dois Níveis: Algoritmo Genético Paralelo com Análise Espectral Paralela

A abordagem em dois níveis foi construída nesse trabalho de modo que em um nível mais alto fique o paralelismo da seleção de estrutura, implementada pelo AG mestre-escrevo e no nível mais baixo fique o paralelismo da solução do problema de autovalor presente na análise espectral. Dessa forma, é possível aproveitar o paralelismo naturalmente explorado nos algoritmos genéticos enquanto se explora o paralelismo na solução de problemas de autovalor, discutido na Seção 4.4.

Para uma melhor ilustração da idéia envolvendo o paralelismo em dois níveis, a Figura 4.7 apresenta uma representação esquemática da arquitetura implementada.

Assim como na implementação sequencial, a implementação do paralelismo na análise espectral foi realizada utilizando a linguagem de programação C e com auxílio do pacote SLEPc. Para a implementação do paralelismo neste nível, foi utilizada a biblioteca MPI.

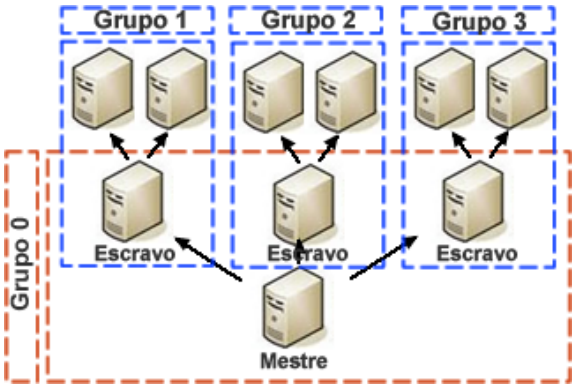

Figura 4.7: Implementação da arquitetura em dois níveis.

A implementação da arquiteura em dois níveis foi possível devido aos conceitos de grupos e comunicadores providos pelo MPI. Um comunicador permite a definição de módulos que encapsulam operações de comunicação dentro de um grupo. O comunicador identifica o grupo de processos e o contexto ao qual uma determinada operação deve ser aplicada.

O MPI fornece um comunicador predefinido, denominado MPI_COMM_WORLD, que identifica todos os processos envolvidos na computação através de identificadores globais.

No modelo em dois níveis implementado neste trabalho, o processador mestre, pertencente ao grupo 0 envia os indivíduos aos processadores escravos que pertencem também ao grupo 0 (escravos nível 1). Cada um dos escravos nível 1 pertence também a um outro grupo e é responsável pela comunicação com os escravos nível 2 de seu grupo. Cada grupo é responsável pela análise espectral uma parte da população. A comunicação dentro de cada grupo é possível devido à criação de novos subcomunicadores.

Capítulo 5

Experimentos Computacionais

Esse capítulo apresenta os experimentos computacionais realizados para avaliação do método implementado. Inicialmente é apresentado o ambiente computacional onde os experimentos foram realizados e as tecnologias utilizadas para a implementação. São apresentadas também as bases de dados utilizadas para avaliação dos resultados e é feita uma breve discussão sobre os resultados obtidos.

5.1 Linguagens de Programação e Ambiente Computacional Utilizados

Toda a implementação do FSM foi feita através de *softwares* livres disponíveis na *web*. O sistema operacional utilizado para a implementação foi o Linux Ubuntu 7.10.

A base do sistema foi desenvolvida utilizando a linguagem de programação Python [73]. Python é uma linguagem interpretada, orientada a objetos que recentemente vem tendo um grande crescimento em aplicações científicas nas mais diversas áreas. As etapas menos computacionalmente custosas do FSM, como preparação de dados, pré-processamento, apresentação de resultados e AG para seleção de estrutura foram implementados na linguagem Python.

Como o Python integra-se muito bem com outras linguagens, as etapas mais computacionalmente custosas do FSM foram programadas com a linguagem de programação C. Assim, foram implementadas em C, basicamente a análise espectral e o algoritmo de otimização de pesos de regras. Dessa forma, pode-se tirar proveito da versatilidade da linguagem Python aliada à eficiência da linguagem C.

Todos os experimentos foram executados no Laboratório de Fisiologia Computacional e Computação de Alto Desempenho da Universidade Federal de Juiz de Fora. Foi utilizado um *cluster* Linux de oito núcleos composto de duas máquinas, cada uma equipada com dois processadores Intel Xeon de *clock* $1.6GHz$ e memória RAM com capacidade de $4GB$.

5.2 Bases de Dados *Benchmark* Utilizadas

Um conjunto de bases de dados *benchmark* foi extraído do *UCI Machine Learning Repository* [74] e utilizado para a avaliação do desempenho do FSM em relação à precisão e à interpretabilidade dos modelos obtidos. As características básicas das bases de dados *benchmark* selecionadas podem ser encontradas na Tabela 5.1.

Tabela 5.1: Bases de dados *benchmark* utilizadas.

Base de Dados	Atributos	Classes	Registros
Iris	4	3	150
Wine	13	3	178
Diabetes	8	2	768
Glass	9	6	214
Heart	13	5	291
Ionosphere	32	2	244
Balance	4	3	625
Cancer	9	2	683
Sonar	60	2	208
Image	18	7	210

A Tabela 5.2 apresenta alguns resultados de precisão apresentados por outros modelos encontrados na literatura utilizando as mesmas bases de dados da Tabela 5.1, entretanto esses valores devem ser utilizados apenas como referência, já que os experimentos não foram realizados sob as mesmas condições.

Em [75] é feita uma combinação de classificadores fracos com o objetivo de se construir um classificador com melhores taxas de precisão. Em [31] é apresentado

Tabela 5.2: Resultados encontrados na literatura.

Base de Dados	[75]	[31]	[39]
Iris	94.67	-	96.4
Wine	-	99.44	97.2
Diabetes	74.93	-	-
Glass	69.60	53.27	-
Heart	-	-	-
Ionosphere	90.89	-	-
Balance	86.19	-	-
Cancer	71.93	-	-
Sonar	75.65	81.25	-
Image	-	82.86	-

um sistema *fuzzy* onde as regras têm seus pesos ajustados por funções de punição e recompensa.

5.3 Resultados de Desempenho do Classificador *Fuzzy*

Com o objetivo de avaliar o FSM em relação a precisão e interpretabilidade, cada execução do método foi realizada em validação cruzada de 10 dobras (10-*fold cross-validation*). Ou seja, a base de dados é dividida em 10 partes e o método é executado em 10 etapas. Em cada etapa, nove partes são utilizadas para a geração do modelo e a parte restante é utilizada para testar o modelo gerado. Assim, o método é executado 10 vezes e a precisão do modelo é a média da precisão resultante em cada uma das etapas. Esse tipo de análise evita que o modelo seja sobre-ajustado a um determinado conjunto de registros, permitindo que seja generalizado para outros dados.

Cada execução do método foi repetida 10 vezes, garantindo assim a generalidade do algoritmo genético para seleção de estrutura. Assim, para cada base de dados, o

algoritmo genético para seleção de estrutura foi executado 100 vezes (10 execuções com validação cruzada em 10 dobras).

A média da precisão de classificação assim como a média do número de regras para cada base de dados *benchmark* são apresentados na Tabela 5.3.

Tabela 5.3: Resultado de desempenho médio em relação a precisão e número de regras para as bases de dados *benchmark*.

Base de Dados	Precisão	Número de Regras
Iris	0.9826	3.2
Wine	0.9781	8.7
Diabetes	0.7867	12.5
Glass	0.7394	11.7
Heart	0.6089	14.3
Ionosphere	0.9147	26.7
Balance	0.8141	8.3
Cancer	0.9813	8.4
Sonar	0.8084	32.8
Image	0.8712	12.8

Como pode-se observar, o classificador FSM obteve melhor resultado de precisão do que os métodos similares encontrados na literatura. Entretanto, uma comparação direta entre tais valores deve ser feita com cuidado uma vez que a igualdade nas condições de execução dos experimentos não é garantida. Da Tabela 5.3 pode-se perceber também que o número de regras é mais baixo em problemas menos complexos. Em problemas mais complexos, onde as classes não são facilmente separáveis e há um grande número de atributos, há uma tendência de elevação no número de regras no modelo na tentativa de elevar a sua precisão.

5.4 Interpretação do Modelo

Um modelo *fuzzy* foi gerado para cada base de dados *benchmark* utilizando todos os registros na base de dados. Os resultados são apresentados na Tabela 5.4, onde p é

o número original de variáveis, $\widehat{p}$ é o número de variáveis selecionadas, n é o número de regras, e γ é o vetor de estrutura que define o número de variáveis e também o número de conjuntos *fuzzy* na partição de cada variável, utilizado como entrada para o Algoritmo 1.

Tabela 5.4: Estruturas encontradas para as bases de dados *benchmark*.

Base de Dados	p	$\widehat{p}$	n	Precisão	γ
Iris	4	3	3	0.9667	[3,1,7,5]
Wine	13	9	12	0.9888	[4,2,1,1,2,1,6,1,4,6,6,7,5]
Diabetes	8	5	10	0.7826	[1,6,1,3,1,4,2,4]
Glass	9	9	10	0.7389	[2,7,4,8,3,6,4,5,7]
Heart	13	10	17	0.6426	[2,6,5,1,2,2,1,6,2,6,1,7,8]
Ionosphere	34	22	31	0.9331	[6,6,7,8,2,4,6,1,1,1,3,3,1,2,2,1,...
					5,7,6,6,1,1,4,5,4,7,1,1,7,3,1,6]
Balance	4	4	5	0.8043	[2,4,4,5]
Cancer	9	5	8	0.9722	[4,6,1,3,1,4,1,2,1]
Sonar	60	44	49	0.9038	[1,4,5,2,4,4,1,4,5,7,3,6,1,2,6,4,...
					1,5,3,6,2,1,6,1,7,7,7,3,5,2,1,1,...
					5,1,6,4,5,6,7,1,2,1,4,6,3,5,1,4,...
					8,1,4,5,3,4,1,1,4,1,5,4]
Image	18	17	15	0.9000	[4,7,5,2,7,4,2,8,6,6,2,8,7,4,7,3,3,1]

A Tabela 5.5 apresenta a tabela de regras (matriz Ξ) e os pesos de confiança das regras (matriz Φ) para o modelo obtido com a base de dados Iris. Cada linha da matriz Ξ corresponde à premissa de uma regra. Cada coluna representa uma variável do problema. Cada valor da matriz Ξ é então o índice do conjunto *fuzzy* na partição *fuzzy* de cada variável que deve ser incluído na regra. O símbolos c_1, c_2 e c_3 referem-se, respectivamente, à classe 1, classe 2 e classe 3 no problema Iris.

No exemplo da Tabela 5.5, a primeira regra é definida pela premissa $A_{1,2} \wedge A_{3,4} \wedge A_{4,3}$, onde $A_{i,j}$ é o j^{esimo} conjunto *fuzzy* na partição *fuzzy* da variável x_i e cada conjunto *fuzzy* corresponde a um símbolo que pode ser associado a um conceito linguístico no contexto da aplicação. Para um melhor entendimento, os dados da

Tabela 5.5: Modelo gerado para a base de dados Iris.

	Matriz Ξ				Matriz Φ		
	x_1	x_2	x_3	x_4	c_1	c_2	c_3
A_1	2	-	4	3	0.00	1.00	0.00
A_2	2	-	3	2	1.00	0.00	0.00
A_3	2	-	5	4	0.00	0.00	1.00

Tabela 5.5 podem ser lidos como:

- Regra 1: Se x_1 é $A_{1,2}$ e x_3 é $A_{3,4}$ e x_4 é $A_{4,3}$ então c_1 (com $\varphi_1 = 0.00$) / c_2 (com $\varphi_2 = 1.00$) / c_3 (com $\varphi_3 = 0.00$)
- Regra 2: Se x_1 é $A_{1,2}$ e x_3 é $A_{3,3}$ e x_4 é $A_{4,2}$ então c_1 (com $\varphi_1 = 1.00$) / c_2 (com $\varphi_2 = 0.00$) / c_3 (com $\varphi_3 = 0.00$)
- Regra 3: Se x_1 é $A_{1,2}$ e x_3 é $A_{3,5}$ e x_4 é $A_{4,4}$ então c_1 (com $\varphi_1 = 0.00$) / c_2 (com $\varphi_2 = 0.00$) / c_3 (com $\varphi_3 = 1.00$)

Pode-se perceber que, apesar de o algoritmo de seleção de estrutura definir um grande número de conjuntos *fuzzy* para a partição *fuzzy* de algumas variáveis, as regras não utilizam todos os conjuntos *fuzzy*. Por exemplo, sete conjuntos *fuzzy* foram definidos para a variável x_4 mas apenas os conjuntos *fuzzy* 2, 3 e 4 foram efetivamente utilizados. Isso certamente ocorre devido à busca pela melhor localização dos protótipos dos conjuntos *fuzzy*, que não podem ser livremente ajustados para preservar a interpretabilidade do modelo. Assim o algoritmo de seleção de estrutura aumenta o número de conjuntos *fuzzy* na partição de algumas variáveis.

Apenas para ilustração, será apresentado também o modelo gerado para a base de dados Diabetes, que é uma base mais complexa em termos de número de variáveis e sobreposição das classes:

- Regra 1: Se x_2 é $A_{2,3}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,3}$ e x_7 é $A_{7,1}$ e x_8 é $A_{8,2}$ então c_1 (com $\varphi_1 = 1.00$) / c_2 (com $\varphi_2 = 0.00$)
- Regra 2: Se x_2 é $A_{2,3}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,2}$ e x_7 é $A_{7,1}$ e x_8 é $A_{8,2}$ então c_1 (com $\varphi_1 = 1.00$) / c_2 (com $\varphi_2 = 0.00$)

- Regra 3: Se x_2 é $A_{2,3}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,3}$ e x_7 é $A_{7,1}$ e x_8 é $A_{8,3}$ então c_1 (com $\varphi_1 = 0.68$) / c_2 (com $\varphi_2 = 0.32$)

- Regra 4: Se x_2 é $A_{2,3}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,2}$ e x_7 é $A_{7,2}$ e x_8 é $A_{8,2}$ então c_1 (com $\varphi_1 = 0.92$) / c_2 (com $\varphi_2 = 0.08$)

- Regra 5: Se x_2 é $A_{2,3}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,3}$ e x_7 é $A_{7,2}$ e x_8 é $A_{8,2}$ então c_1 (com $\varphi_1 = 0.55$) / c_2 (com $\varphi_2 = 0.45$)

- Regra 6: Se x_2 é $A_{2,4}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,2}$ e x_7 é $A_{7,1}$ e x_8 é $A_{8,2}$ então c_1 (com $\varphi_1 = 0.77$) / c_2 (com $\varphi_2 = 0.23$)

- Regra 7: Se x_2 é $A_{2,4}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,3}$ e x_7 é $A_{7,1}$ e x_8 é $A_{8,2}$ então c_1 (com $\varphi_1 = 0.48$) / c_2 (com $\varphi_2 = 0.52$)

- Regra 8: Se x_2 é $A_{2,5}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,3}$ e x_7 é $A_{7,2}$ e x_8 é $A_{8,3}$ então c_1 (com $\varphi_1 = 0.10$) / c_2 (com $\varphi_2 = 0.90$)

- Regra 9: Se x_2 é $A_{2,4}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,3}$ e x_7 é $A_{7,2}$ e x_8 é $A_{8,3}$ então c_1 (com $\varphi_1 = 0.08$) / c_2 (com $\varphi_2 = 0.92$)

- Regra10:Se x_2 é $A_{2,4}$ e x_4 é $A_{4,2}$ e x_6 é $A_{6,2}$ e x_7 é $A_{7,1}$ e x_8 é $A_{8,4}$ então c_1 (com $\varphi_1 = 0.82$) / c_2 (com $\varphi_2 = 0.18$)

A relação entre a complexidade dos dados, o número de variáveis e o número de regras resultantes geradas pelo modelo pode ser avaliada através da comparação entre os modelos gerados para a base Iris e a base Diabetes. O problema Diabetes é mais dificilmente separado que o problema Iris, como pode ser visto pelas diferenças relativas na matriz de afinidade, apresentada na Figura 5.1. Consequentemente, o número de variáveis selecionadas e o número de regras para a base de dados Diabetes é maior que o número de regras encontrado para a base de dados Iris. Isso é refletido também no número de regras certas. Enquanto o modelo encontrado para a base de dados Iris apresenta regras certas, várias regras do modelo para a base de dados Diabetes apresentam um alto grau de incerteza, como pode ser visto nos modelos apresentados anteriormente.

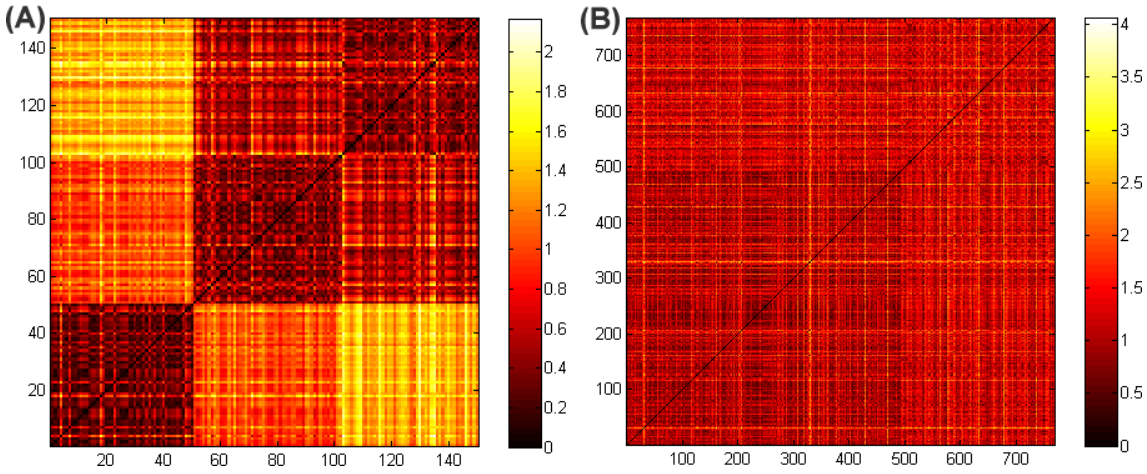

Figura 5.1: Comparação entre o *data image* das bases: (A) Iris e (B) Diabetes.

5.5 Resultados de Desempenho no Paralelismo

Como já discutido nesse trabalho, o alto tempo requerido para o processamento do FSM é um ponto negativo no método. A Tabela 5.6 apresenta o tempo (em segundos) de processamento de uma execução do algoritmo genético para seleção de estrutura nas mesmas bases de dados apresentadas na Tabela 5.1 utilizando apenas um processador sob as condições descritas na Seção 5.1.

Para uma melhor avaliação das implementações paralelas, foram utilizadas, além das bases de dados *benchmark*, algumas bases de dados sintéticas, sem significado real, mas com um número maior de registros. As bases de dados sintéticas são apresentadas na Tabela 5.7, assim como o tempo de processamento sequencial (em segundos) de cada uma delas. Todas as bases de dados sintéticas utilizadas nessa análise possuem 18 atributos. Como as bases sintéticas apresentam um grande número de registros, para facilitar a realização deste trabalho, a análise destas bases foi feita com apenas uma geração do o algoritmo genético para seleção de estrutura, e não com 150 gerações, como foi feito com as bases de dados *benchmark*.

A utilização de bases de dados com um número maior de registros possibilita uma melhor análise do comportamento das implementações paralelas em problemas maiores. Os dados apresentados na Tabela 5.7 são também apresentados na Figura **??**, possibilitando uma melhor visualização do crescimento na necessidade de redução do tempo de execução da FSM a medida em que são utilizadas bases de dados com maiores números de registros.

Tabela 5.6: Tempo de processamento sequencial da FSM para as bases de dados *benchmark*.

Base de Dados	Tempo Sequencial(s)
Iris	31.56
Wine	633.14
Diabetes	1795.72
Glass	487.39
Heart	1260.93
Ionosphere	4752.24
Balance	145.69
Cancer	1665.62
Sonar	8472.36
Image	963.83

Tabela 5.7: Tempo de processamento sequencial da FSM para as bases de dados sintéticas.

Base de Dados	Número de Registros	Tempo Sequencial(s)
Sintética 1000	1000	99.68
Sintética 1500	1500	291.44
Sintética 2000	2000	580.18
Sintética 2500	2500	1202.09
Sintética 3000	3000	1952.7
Sintética 3500	3500	2925.37
Sintética 4000	4000	4339.75

5.5.1 Algoritmo Genético Paralelo com Análise Espectral Sequencial

A Tabela 5.8 apresenta o tempo de processamento (em segundos) da execução da FSM com AG paralelo (e análise espectral sequencial) para as bases de dados *ben-*

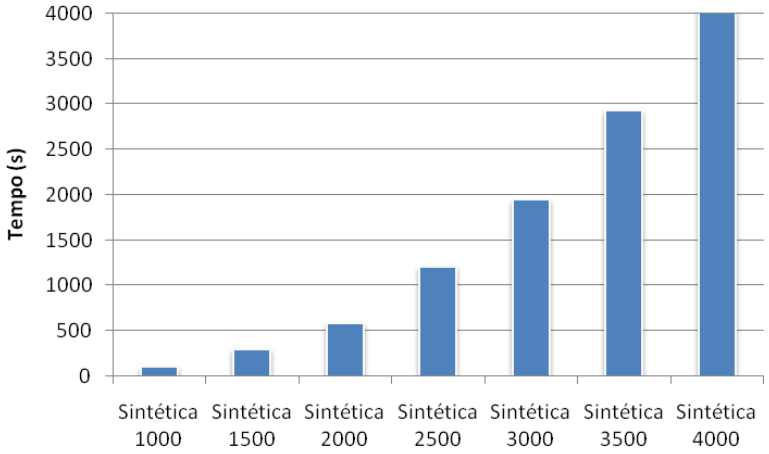

Figura 5.2: Tempo de processamento da FSM para as bases de dados sintéticas.

chmark para 2, 4 e 8 processadores.

Tabela 5.8: Tempo de execução da FSM com algoritmo genético paralelo e análise espectral sequencial para as bases de dados *benchmark*.

Base de Dados	Tempo 2 Proc.(s)	Tempo 4 Proc.(s)	Tempo 8 Proc.(s)
Iris	20.26	14.86	15.47
Wine	383.40	240.50	159.37
Diabetes	1194.80	738.28	550.75
Glass	311.88	194.60	133.64
Heart	709.78	428.30	288.53
Ionosphere	2517.57	1354.10	853.56
Balance	107.29	88.35	69.51
Cancer	1070.35	696.54	476.76
Sonar	4573.21	2480.80	1365.99
Image	537.29	331.01	221.60

A eficiência paralela da FSM com algoritmo genético paralelo e análise espectral sequencial para as bases de dados *benchmark* é apresentada na Figura 5.3. A eficiência paralela foi computada da seguinte forma:

$$\varepsilon_{nprocs} = \frac{1}{p}\frac{T_S}{T_p} \tag{5.1}$$

onde *nprocs* é o número de processadores, T_S o tempo de processamento sequencial (em segundos) e T_p o tempo de processamento paralelo utilizando p processadores.

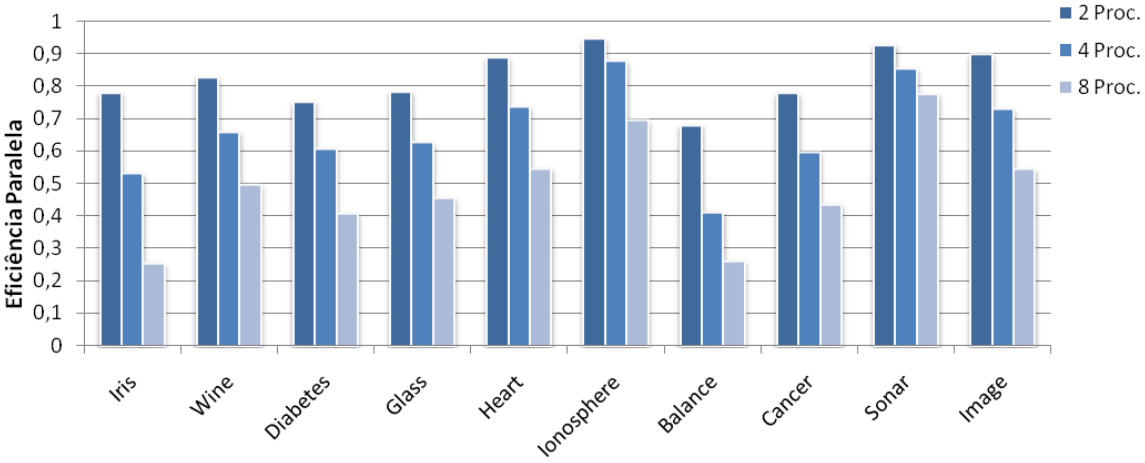

Figura 5.3: Eficiência paralela da FSM com algoritmo genético paralelo e análise espectral sequencial para as bases de dados *benchmark*.

A Tabela 5.9 apresenta o tempo de processamento (em segundos) da execução da FSM com AG paralelo (e análise espectral sequencial) para as bases de dados sintéticas utilizando 2, 4 e 8 processadores.

Tabela 5.9: Tempo de execução da FSM com algoritmo genético paralelo e análise espectral sequencial para as bases de dados sintéticas.

Base de Dados	Tempo 2 Proc.(s)	Tempo 4 Proc.(s)	Tempo 8 Proc.(s)
Sintética 1000	54.79	47.66	54.16
Sintética 1500	159.68	114.97	127.02
Sintética 2000	317.14	210.73	230.89
Sintética 2500	651.65	433.32	447.95
Sintética 3000	1050.2	700.89	620.05
Sintética 3500	1601.02	1170.52	1028.74
Sintética 4000	2380.84	1765.83	1486.28

A eficiência paralela do algoritmo genético paralelo com análise espectral sequencial para as bases de dados sintéticas é apresentada na Figura 5.4.

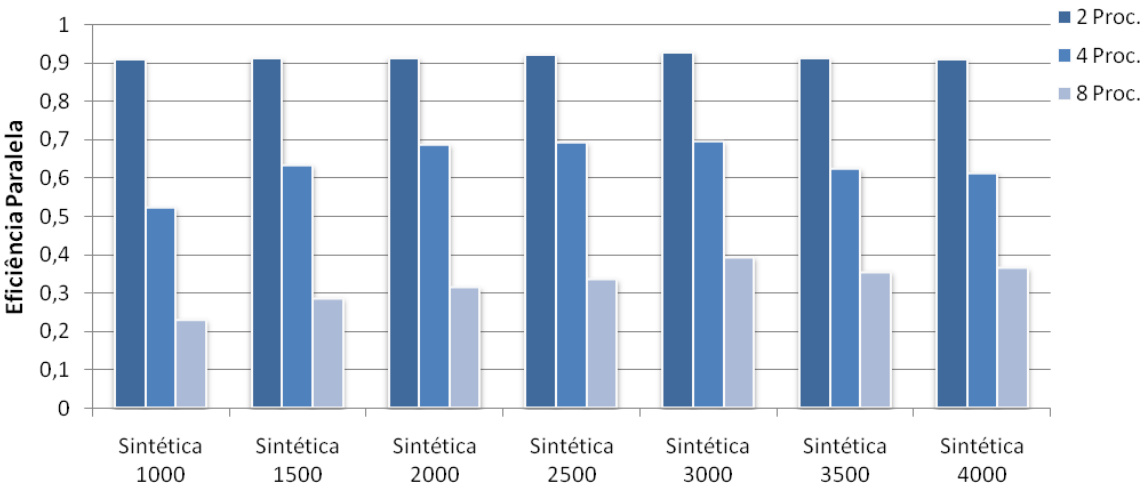

Figura 5.4: Eficiência paralela da FSM com algoritmo genético paralelo e análise espectral sequencial para as bases de dados sintéticas.

Através da observação dos resultados anteriores, pode-se notar que os valores de eficiência paralela para as bases de dados sintéticas são consideravelmente melhores quando comparados aos valores nas bases de dados *benchmark*. Isso acontece devido ao baixo número de registros apresentado nas bases de dados *benchmark*, que faz com o tempo demandado pela avaliação dos indivíduos, que é de fato o trecho da implementação paralelizado, seja muito pequeno quando comparado ao restante do algoritmo, tornando a paralelização menos eficiente.

Pode-se observar também que a eficiência paralela diminui consideravelmente a medida que o número de processadores cresce. Isso acontece devido a uma série de razões. Uma delas é a sobrecarga de comunicação durante a execução, que é ainda mais prejudicial em problemas com poucos números de registros.

Outra razão é o desbalanceamento de carga introduzido pelo esquema para evitar reavaliação de indivíduos, apresentado na Seção 3.2.3. O desbalanceamento de carga ocorre porque a estratégia para evitar reavaliação de indivíduos pode não ser uniforme através dos processadores escravos, ou seja, um número diferente de indivíduos pode ser enviado para a avaliação nos processos escravos. Assim, se o número de indivíduos a serem avaliados em uma determinada geração do AG não for divisível pelo número de processadores disponíveis, um ou mais processadores ficarão ociosos naquela geração, aguardando que os outros processadores finalizem o trabalho para sincronizar em uma próxima geração. Por isso, no cenário estudado,

quanto maior o número de processadores, maior é a chance de haver desbalanceamento de carga. Esse efeito de de desbalanceamento de carga é minimizado em problemas mais complexos e que apresentam um maior número de variáveis, porque nesse tipo de problemas o algoritmo genético para seleção de estrutura tende a avaliar mais indivíduos distintos, fazendo com que pequenas diferenças na carga dos processadores não seja tão prejudicial.

Outro fato a se notar da Figura 5.4 é que quando o número de registros chega em 3500, há uma redução na eficiência paralela com 4 e 8 processadores. Um dos motivos para isso acontecer é que, como o ambiente em que os experimentos foram executados possui duas máquinas com 4 processadores compartilhando uma única memória, utilizando-se 4 ou 8 processadores cada máquina executa, simultaneamente, a avaliação de quatro indivíduos, o que faz com quatro processos tenham que utilizar a mesma memória, causando competição entre esses processos pelo acesso às estruturas de dados necessárias para a solução do algoritmo.

5.5.2 Algoritmo Genético Paralelo com Análise Espectral Paralela

A implementação em dois níveis (algoritmo genético paralelo com análise espectral paralela) foi também avaliada no mesmo ambiente computacional descrito anteriormente. A Tabela 5.10 apresenta o tempo de processamento (em segundos) da execução da FSM com AG paralelo e análise espectral paralela para as mesmas bases de dados *benchmark* utilizando 2, 4 e 8 processadores e atribuindo dois processadores por grupo (isto é, cada análise espectral é realizada por dois processadores em paralelo).

A eficiência paralela do algoritmo genético paralelo com análise espectral paralela para as bases de dados *benchmark* é apresentada na Figura 5.5.

A Tabela 5.11 apresenta o tempo de processamento (em segundos) da execução da FSM com AG paralelo e análise espectral paralela para as bases de dados sintéticas utilizando 2, 4 e 8 processadores.

A Figura 5.6 apresenta a eficiência paralela da FSM com AG paralelo e análise espectral paralela para as bases de dados sintéticas.

Novamente nota-se que a eficiência paralela é reduzida a medida que o número

Tabela 5.10: Tempo de execução da FSM com algoritmo genético paralelo e análise espectral paralela para as bases de dados *benchmark*.

Base de Dados	Tempo 2 Proc.(s)	Tempo 4 Proc.(s)	Tempo 8 Proc.(s)
Iris	27.97	20.52	19.29
Wine	798.65	456.81	328.30
Diabetes	2185.32	1303.46	833.62
Glass	569.87	346.09	252.54
Heart	1409.67	826.30	579.43
Ionosphere	5704.34	3401.24	2949.77
Balance	185.67	128.28	116.71
Cancer	1628.91	994.00	745.43
Sonar	15981.88	8619.51	10239.66
Image	1111.12	631.11	471.26

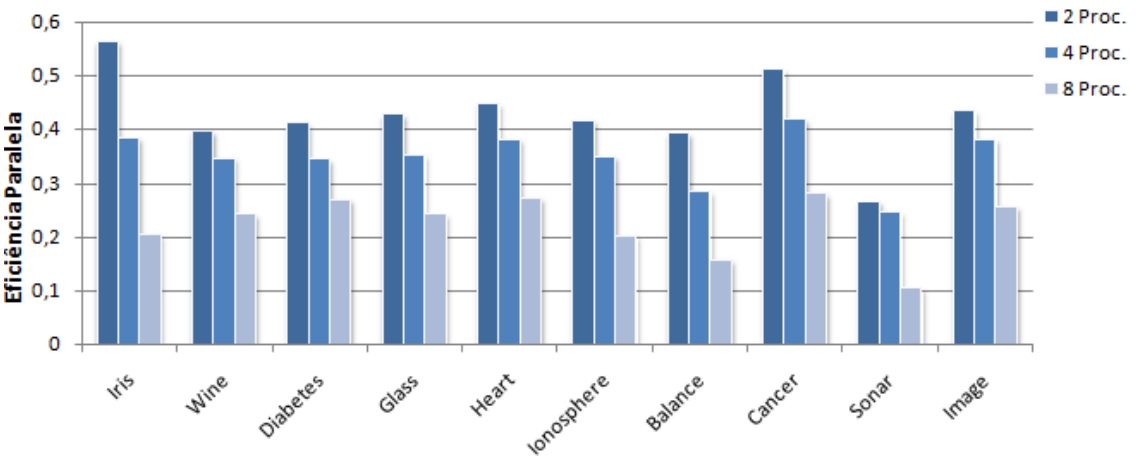

Figura 5.5: Eficiência paralela da FSM com algoritmo genético paralelo e análise espectral paralela para as bases de dados *benchmark*.

de processadores aumenta. Além disso, pode-se perceber que o paralelismo em dois níveis se mostra uma estratégia ruim quando se trata de bases de dados com poucos registros. Isso acontece porque para bases de dados com número reduzido de registros, a solução do problema de autovalor é computada rapidamente pelo SLEPc, fazendo com que o *overhead* introduzido pela paralelização seja relativamente grande

Tabela 5.11: Tempo de execução da FSM com algoritmo genético paralelo e análise espectral paralela para as bases de dados sintéticas.

Base de Dados	Tempo 2 Proc.(s)	Tempo 4 Proc.(s)	Tempo 8 Proc.(s)
Sintética 1000	76.01	40.03	35.41
Sintética 1500	194.03	101.03	90.17
Sintética 2000	368.81	195.59	160.38
Sintética 2500	742.67	392.49	290.63
Sintética 3000	1182.08	615.06	416.85
Sintética 3500	1743.35	904.13	571.02
Sintética 4000	2537.81	1300.7	805.05

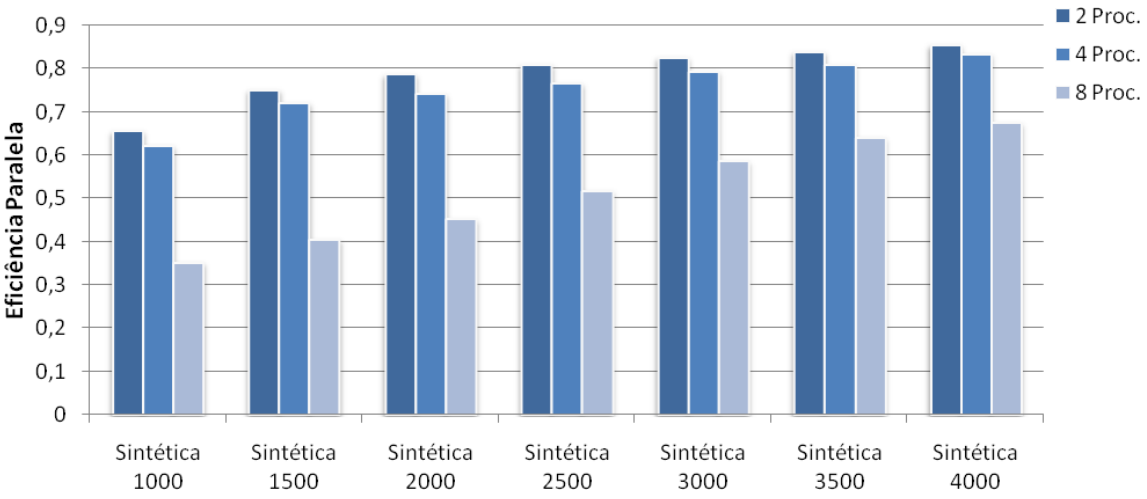

Figura 5.6: Eficiência paralela da FSM com algoritmo genético paralelo e análise espectral paralela para as bases de dados sintéticas.

quando comparado ao tempo de processamento, prejudicando consideravelmente a eficiência da paralelização.

Em problemas com maiores números de registros entretanto, a estratégia em dois níveis, mostra resultados mais interessantes. Quando o número de indivíduos a serem avaliados em uma geração é pequeno em relação ao número de processadores, o desbalanceamento de carga introduzido pelo esquema para evitar reavaliação pode prejudicar muito o paralelismo. Entretanto esse efeito pode ser reduzido quando os

processadores são agrupados para a avaliação do indivíduos.

Outra grande vantagem da paralelização em dois níveis é a possibilidade de se trabalhar com bases de dados com números de registros muito grandes. A matriz Laplaciana, sobre a qual o problema de autovalor é calculado, é de ordem $O(N^2)$, onde N é o número de registros do problema. Quando executa-se o AG paralelo com análise espectral sequencial, *nprocs* matrizes Laplaciana são armazenadas simultaneamente em memória, onde *nprocs* é o número de processadores envolvidos na execução, fazendo com que a memória seja um gargalo na execução. Dividindo-se a solução do problema de autovalor entre os processadores, apenas uma matriz Laplaciana é armazenada em memória em um determinado instante, permitindo a execução do método com bases de dados maiores.

Variação no Número de Processadores para a Paralelização da Análise Espectral

Com o objetivo de explorar a vantagem oferecida pelo paralelismo de dados no problema de autovalor, foram realizados testes através da variação do número de processadores para a solução da análise espectral.

As Tabelas 5.12 e 5.13 exibem o tempo de processamento (em segundos) de apenas uma análise espectral para diferentes bases de dados, considerando 1, 2, 4 e 8 processadores para as bases de dados *benchmark* e sintéticas, respectivamente.

As Figuras 5.7 e 5.8 apresentam a eficiência paralela de apenas uma análise espectral para diferentes bases de dados, considerando 1, 2, 4 e 8 processadores.

Pode-se observar pelas Tabelas 5.12 e 5.13, pelas Figuras 5.7 e 5.8 pela novamente que para bases de dados com poucos registros a paralelização da análise espectral não é uma boa alternativa. Entretanto, a medida em que o número de registros aumenta, a eficiência paralela torna-se uma boa estratégia por oferecer uma série de vantagens, como discutido anteriormente, prejudicando pouco a eficiência do paralelismo.

Tabela 5.12: Tempo de processamento da análise espectral utilizando diferentes números de processadores para as bases de dados *benchmark*.

Base de Dados	Tempo seq.(s)	Tempo 2 Proc.(s)	Tempo 4 Proc.(s)	Tempo 8 Proc.(s)
Iris	0.0296	0.0579	0.0665	0.1870
Wine	0.0487	0.0917	0.1169	0.3046
Diabetes	1.7994	1.4715	1.0809	1.3009
Glass	0.0806	0.1457	0.1698	0.4801
Heart	0.1481	0.1959	0.1960	0.4892
Ionosphere	0.3214	0.4618	0.5178	1.3675
Balance	0.9439	0.8040	0.5798	0.6590
Cancer	1.3674	1.1930	0.9432	1.3669
Sonar	0.1708	0.4006	0.5293	1.7186
Image	0.0735	0.1210	0.1464	0.4019

Tabela 5.13: Tempo de processamento da análise espectral utilizando diferentes números de processadores para as bases de dados sintéticas.

Base de Dados	Tempo seq.(s)	Tempo 2 Procs.(s)	Tempo 4 Procs.(s)	Tempo 8 Procs.(s)
Sintética 1000	3.5805	2.6324	1.8028	1.8055
Sintética 1500	11.3881	7.6000	4.7763	4.0203
Sintética 2000	25.4861	15.8534	9.6380	6.9047
Sintética 2500	49.3627	29.7753	17.5701	12.4055
Sintética 3000	83.5211	48.7718	28.3070	18.9678
Sintética 3500	131.6441	75.4448	42.7243	27.5723
Sintética 4000	198.7666	113.3115	64.5201	41.8887

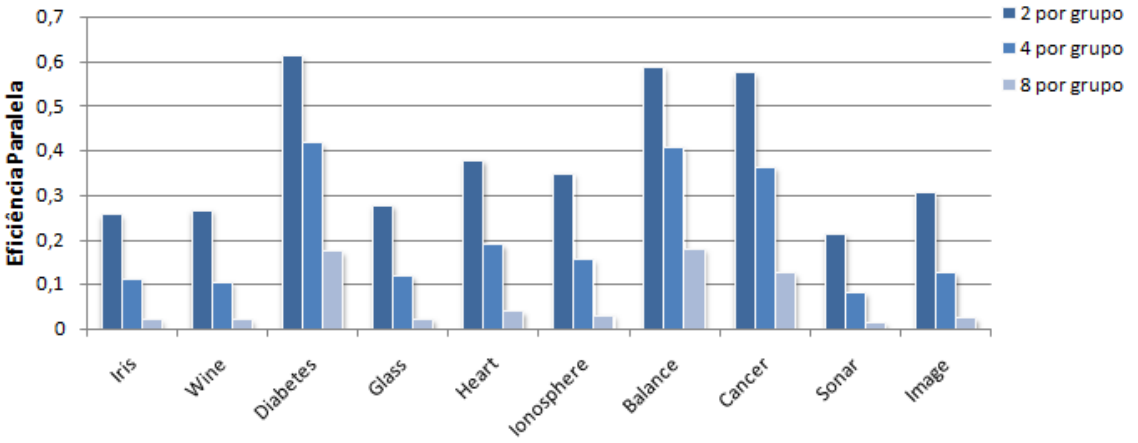

Figura 5.7: Eficiência paralela da análise espectral utilizando diferentes números de processadores para as bases de dados *benchmark*.

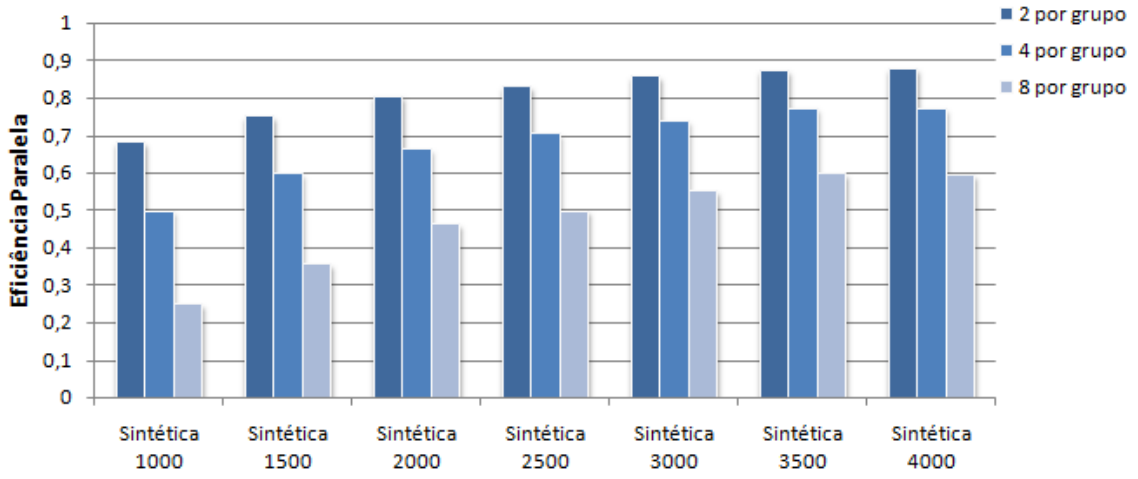

Figura 5.8: Eficiência paralela da análise espectral utilizando diferentes números de processadores para as bases de dados sintéticas.

Capítulo 6

Conclusões

Nesse trabalho foi desenvolvida a implementação paralela da *Fuzzy Symbolic Machine* (FSM), um sistema para extração de regras *fuzzy* interpretáveis. A FSM efetua a geração de modelos *fuzzy* através de três passos principais: seleção de estrutura, identificação da base de regras e estimação de parâmetros. O classificador tem como objetivo gerar regras que descrevam com precisão o conjunto de dados e sejam também interpretáveis por especialistas humanos.

O número de regras no modelo *fuzzy* resultante é definido através da análise espectral, uma técnica derivada da teoria dos grafos que tem como objetivo estimar o número de grupos em um conjunto de dados através da observação do espectro dos dados, definido por seus autovalores. Os parâmetros de confiança das regras são otimizados através da solução de um problema de otimização quadrática limitado.

As funções de pertinência *fuzzy* foram arbitrariamente escolhidas para serem igualmente espaçadas no domínio de cada variável de entrada, com o objetivo de melhorar a interpretabilidade. Entretanto, qualquer outra escolha para posicionamento das funções de pertinência poderia ser utilizada no algoritmo de indução de regras.

A estrutura do modelo é otimizada por um algoritmo genético *wrapper*, que simultaneamente define as variáveis que devem entrar no modelo e o número de conjuntos fuzzy na partição *fuzzy* de cada variável.

O método foi implementado em um ambiente de alto desempenho seguindo duas abordagens. Em uma delas, o algoritmo genético para otimização da estrutura foi paralelizado, de modo que, ao mesmo tempo, várias estruturas candidatas pudes-

sem ser avaliadas. Outra implementação foi feita paralelizando, além do algoritmo genético para seleção de estrutura, a solução do problema de autovalor contido na função de avaliação.

O uso de computadores multi-processados e *clusters* de computadores, que antes era bastante restrito a ambientes acadêmicos, hoje em dia vem ganhando espaço em ambientes comerciais. Por isso, implementações paralelas vêm possibilitando que métodos computacionalmente custosos, como é o caso da FSM, possam ser aplicados a problemas reais, vivenciados no dia a dia de empresas e organizações não acadêmicas.

Foi implementado também um esquema para armazenar as soluções intermediárias do AG e utilizá-las de forma a reduzir o volume de processamento necessário para a solução da FSM.

Os experimentos realizados mostram que a FSM possui bons resultados de precisão quando comparado a modelos similares encontrados na literatura. Os resultados dos testes para avaliação da eficiência do paralelismo indicam que, mesmo com o desbalanceamento de carga introduzido pela estratégia para evitar reavaliação, paralelizando-se apenas o algoritmo genético pode-se reduzir o tempo de processamento da FSM para bases de dados menores. A abordagem em dois níveis mostrou-se uma estratégia mais adequada para a solução de problemas com um grande número de registros. Com a utilização dessas duas implementações cria-se a possibilidade de se trabalhar eficientemente com bases de dados de diferentes tamanhos, tornando o sistema apto à geração de modelos em problemas reais de variados tamanhos.

Como trabalhos futuros propõe-se a exploração do paralelismo em outras etapas do algoritmo de indução de regras, fazendo com que o uso de vários processadores seja melhor aproveitado. Propõe-se também a avaliação do método em um maior número de processadores e com bases de dados com mais registros.

Uma outra direção possível de ser seguida em trabalhos futuros é a extensão do procedimento para problemas de regressão, garantindo a adequação do método a uma gama ainda maior de problemas. Além disso, o uso de interfaces gráficas poderia deixar o sistema mais acessível a usuários comuns.

Referências Bibliográficas

[1] HAN, J., KAMBER, M., *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

[2] NETO, D. O. G., JUNIOR, W. M., FERREIRA, R. A. C., "Anteater: A Service Oriented Architecture for High-Performance Data Mining", *IEEE Internet Computing*, v. 10, n. 4, pp. 36–43, 2006.

[3] SHAWA, M. J., SUBRAMANIAMA, C., TANA, G. W., et al., "Knowledge management and data mining for marketing", *Decision Support Systems*, v. 31, n. 1, 2001.

[4] EVSUKOFF, A., GALICHET, S., LIMA, B. S. L. P., et al., "Design of Interpretable Fuzzy Rule-Based Classifiers Using Spectral Analysis With Structure and Parameters Optimization", *Fuzzy Sets and Systems*, 2008.

[5] EVSUKOFF, A. G., "Learning Fuzzy Rule Based Classifier with Rule Weights Optimization and Structure Selection by a Genetic Algorithm". In: *IEEE International Conference on Fuzzy Systems, London*, 2007.

[6] CASILLAS, J., CORDÓN, O., HERRERA, F., et al., "Accuracy Improvements to Find the Balance Interpretability-Accuracy in Linguistic Fuzzy Modeling: An Overview", *Springer - Studies in Fuzziness and Soft Computing*, 2003.

[7] MITCHELL, T., BUCHANAN, B., DEJONG, G., et al., "Machine Learning", *Annual Review of Computer Science*, v. 4, pp. 417–433, Junho 1990.

[8] KECMAN, V., *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. The MIT Press, Cambridge, MA, 2001.

[9] GUILLAUME, S., "Designing Fuzzy Inference Systems from Data: An Interpretability-Oriented Review", *IEEE Transactions on Fuzzy Systems*, v. 9, n. 3, Junho 2001.

[10] DUCH, W., HAYASHI, Y., "Computational intelligence methods and data understanding". In: *International Symposium on Computational Intelligence, Kosice*, 2000.

[11] ESPÍNDOLA, R. P., "Sistema Inteligente para Classificação de Dados", 2004.

[12] NOZAKI, K., ISHIBUCHI, H., TANAKA, H., "Adaptive Fuzzy Rule-Based Classification Systems", *IEEE Transactions on Fuzzy Systems*, v. 4, n. 3, Agosto 1996.

[13] DUBOIS, D., HULLERMEIER, E., PRADE, H., "A Systematic Approach to the Assessment of Fuzzy Association Rules", *Data Mining and Knowledge Discovery*, v. 13, 2006.

[14] GONZÁLEZ, J., ROJAS, I., POMARES, H., et al., "Improving the Accuracy While Preserving the Interpretability of Fuzzy Function Approximators by Means of Multi-Objective Evolutionary Algorithms", *International Journal of Approximate Reasoning*, v. 44, n. 1, Janeiro 2007.

[15] CASTELLANO, G., FANELLI, A. M., MENCAR, C., "A Neuro-Fuzzy Network to Generate Human-Understandable Knowledge from Data", *Cognitive Systems Research*, v. 3, pp. 125–144, 2002.

[16] QUINLAN, J. R., "Probabilistic decision trees", , pp. 140 –1521990.

[17] ISHIBUCHI, H., NOZAKI, K., YAMAMOTO, N., et al., "Selecting Fuzzy If-Then Rules for Classification Problems Using Genetic Algorithms", *IEEE Transactions on Fuzzy Systems*, v. 3, n. 3, Agosto 1995.

[18] WANG, L., MENDEL, J. M., "Generating Fuzzy Rules by Learning from Examples", *IEEE Transactions on Systems, Man, and Cybernetics*, v. 22, n. 6, Novembro 1992.

[19] TAKAGI, T., SUGENO, M., "Fuzzy Identification of Systems and Its Applications to Modeling and Control", *IEEE Transactions on Systems, Man, and Cybernetics*, v. 15, n. 1, Janeiro 1985.

[20] ISHIBUCHI, H., NAKASHIMA, T., "Effect of Rule Weights in Fuzzy Rule-Based Classification Systems", *IEEE Transactions on Fuzzy Systems*, v. 9, n. 4, Agosto 2001.

[21] WANG, L., "The WM Method Completed: A Flexible Fuzzy System Approach to Data Mining", *IEEE Transactions on Fuzzy Systems*, v. 11, n. 6, Dezembro 2003.

[22] HULLERMEIER, E., "Fuzzy Methods in Machine Learning and Data Mining: Status and Prospects", *Fuzzy Sets and Systems*, v. 156, n. 3, 2005.

[23] CORDÓN, O., JESUS, M. J., HERRERA, F., "A Proposal on Reasoning Methods in Fuzzy Rule-Based Classification Systems", *International Journal of Approximate Reasoning*, v. 20, n. 1, 1999.

[24] LUXBURG, U., "A Tutorial on Spectral Clustering", *Statistics and Computing*, v. 17, n. 4, Novembro 2007.

[25] LI, W., ONG, K. L., NG, W. K., et al., "Enhancing the Effectiveness of Clustering with Spectra Analysis", *IEEE Transactions on Knowledge and Data Engineering*, v. 19, n. 7, 2007.

[26] FILIPPONE, M., CAMASTRA, F., MASULLI, F., et al., "A Survey of Kernel and Spectral Methods for Clustering", *Pattern Recognition*, v. 41, 2008.

[27] NG, A. Y., JORDAN, M. I., WEISS, Y., "On spectral clustering: Analysis and an algorithm". In: *Advances in Neural Information Processing Systems 14*, pp. 849–856, MIT Press, 2001.

[28] CHUNG, F. R. K., "Lectures on Spectral Graph Theory". In: *CBMS Regional Conf. Series in Mathematics*, n. 92, American Mathematic Society, 1997.

[29] NAUCK, D., KRUSE, R., "How the Learning of Rule Weights Affects the Interpretability of Fuzzy Systems", *em Fuzzy Systems Proceedings, IEEE World Congress on Computational Intelligence*, v. 2, Maio 1998.

[30] ISHIBUCHI, H., YAMAMOTO, T., "Rule Weight Specification in Fuzzy Rule-Based Classification Systems", *IEEE Transactions on Fuzzy Systems*, v. 13, n. 4, Agosto 2005.

[31] JAHROMI, M. Z., TAHERI, M., "A Proposed Method for Learning Rule Weights in Fuzzy Rule-Based Classification Systems", *Fuzzy Sets and Systems*, v. 159, 2008.

[32] COLEMAN, T. F., LI, Y., "A Reflective Newton Method for Minimizing a Quadratic Function Subject to Bounds on Some of the Variables", *SIAM Journal on Optimization*, v. 6, n. 4, 1996.

[33] LIU, H., "Toward Integrating Feature Selection Algorithms for Classification and Clustering", *IEEE Transactions on Knowledge and Data Engineering*, v. 17, n. 4, Abril 2005.

[34] KOHAVI, R., JOHN, G. H., "Wrappers for Feature Subset Selection", *Artificial Intelligence*, v. 97, n. 1-2, pp. 273–324, 1997.

[35] EVSUKOFF, A. G., BRANCO, A. C. S., GALICHET, S., "Structure Identification and Parameter Optimization for Non-Linear Fuzzy Modeling", *Fuzzy Sets and Systems*, v. 132, n. 2, Dezembro 2002.

[36] CORDÓN, O., HERRERA, F., VILLAR, P., "Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base", *IEEE Transactions on Fuzzy Systems*, v. 9, 2001.

[37] HOLLAND, J., "Genetic algorithms", *Scientific American*, pp. 66–72, 1992.

[38] CORDÓN, O., JESUS, M. J., HERRERA, F., "A proposal on reasoning methods in fuzzy rule-based classification systems", *International Journal on Approximate Reasoning*, v. 20, n. 1, 1999.

[39] ISHIBUCHI, H., YAMAMOTO, T., "Fuzzy Rule Selection by Multi-Objective Genetic Local Search Algorithms and Rule Evaluation Measures in Data Mining", *Fuzzy Sets and Systems*, v. 141, n. 1, Janeiro 2004.

[40] SLOAN, J. D., *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI*. O'Reilly, 2004.

[41] MOORE, G. E., "Cramming more Components onto Integrated Circuits", *Electronics*, v. 38, n. 8, 1965.

[42] KUCK, D. J., "A Survey of Parallel Machine Organization and Programming", *ACM Computing Surveys*, v. 9, n. 1, pp. 29–59, 1977.

[43] LOQUES, O., LEITE, J., CARRERA, E. V., "P-RIO: a Modular Parallel-Programming Environment", *IEEE Concurrency*, v. 6, n. 1, pp. 47–57, Janeiro-Março 1998.

[44] UPCRC, "The Universal Parallel Computing Research Center", `http://www.upcrc.illinois.edu/`, Último acesso em: 01 de Abril de 2009.

[45] COSTA, M. C. A., EBECKEN, N. F. F., "Data Mining High Performance Computing Using Neural Networks". In: *International Conference on Applications of High-Performance Computing in Engineering*, 2000.

[46] ZAKI, M. J., TIEN HO, C., AGRAWAL, R., "Parallel Classification for Data Mining on Shared-Memory Multiprocessors". In: *IEEE International Conference on Data Engineering*, pp. 198–205, 1999.

[47] DE ARAUJO, D. L. A., LOPES, H. S., FREITAS, A. A., "A Parallel Genetic Algorithm for Rule Discovery in Large Databases". In: *IEEE SMC '99 Conference Proceedings*, v. 3, pp. 940–945, 1999.

[48] MODENESI, M. V., COSTA, M. C. A., EVSUKOFF, A. G., et al., "Parallel Fuzzy c-Means Cluster Analysis", *Lecture Notes in Computer Science*, v. 4395, n. 1, pp. 52–65, 2007.

[49] NOJIMA, Y., AND, H. I., "Parallel Distributed Genetic Fuzzy Rule Selection and Isao Kuwajima", *Genetic Fuzzy Systems: Recent Developments and Future Directions*, v. 13, n. 5, pp. 511–519, 2008.

[50] DUNCAN, R., "A Survey of Parallel Computer Architectures", *Computer*, v. 23, n. 2, pp. 5–16, Fevereiro 1990.

[51] LOBOSCO, M., LOQUES, O., AMORIM, C. L., "Reducing Memory Sharing Overheads in Distributed JVMs", *Lecture Notes in Computer Science*, , n. 3726, pp. 629–639, 2005.

[52] TANENBAUM, A. S., *Modern Operating Systems*. Prentice Hall, 2001.

[53] GROPP, W., LUSK, E., DOSS, N., et al., "A High Performance, Portable Implementation of the MPI Message Passing Interface Standard", *Parallel Computing*, v. 22, n. 6, pp. 789–828, Setembro 1996.

[54] MPI-FORUM, "MPI: A Message-Passing Interface Standard", `http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html`, Último acesso em: 18 de Fevereiro de 2009.

[55] GUSTAFSON, J. L., "Reevaluating Amdahl's Law", *Communications of the ACM*, v. 31, n. 5, pp. 532–533, 1988.

[56] KLEINROCK, L., "On Parallel Processing Systems: Amdahl's Law Generalized and Some Results on Optimal Design", *IEEE Transactions on Software Engineering*, v. 18, n. 5, 1992.

[57] TOMASSINI, M., "Parallel and Distributed Evolutionary Algorithms: A Review", *Evolutionary Algorithms in Engineering and Computer Science*, pp. 113–133, 1999.

[58] NOWOSTAWSKI, M., POLI, R., "Parallel Genetic Algorithm Taxonomy", *Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference*, pp. 88–92, Dezembro 1999.

[59] GOLUB, M., JAKOBOVIC, D., "A New Model of Global Parallel Genetic Algorithm", *Information Technology Interfaces, 2000. ITI 2000. Proceedings of the 22nd International Conference on*, pp. 363–368, Junho 2000.

[60] CANTÚ-PAZ, E., "A Survey of Parallel Genetic Algorithms", *Calculateurs Paralleles*, v. 10, 1998.

[61] MPI4PY, "MPI for Python", http://mpi4py.scipy.org/, Último acesso em: 17 de Março de 2009.

[62] KATAGIRI, T., "A Study on Large Scale Eigensolvers for Distributed Memory Parallel Machines", 2000.

[63] CHOI, J., DEMMEL, J., DHILLON, I., et al., *ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance*, Tech. rep., Knoxville, TN, USA, 1995, (Relatório Técnico).

[64] LAPACK, "LAPACK: Linear Algebra PACKage", http://www.netlib.org/lapack/, Último acesso em: 18 de Fevereiro de 2009.

[65] BLAS, "BLAS: Basic Linear Algebra Subprograms", http://www.netlib.org/blas/, Último acesso em: 18 de Fevereiro de 2009.

[66] BLACS, "BLACS: Basic Linear Algebra Communication Subprograms", http://www.netlib.org/blacs/, Último acesso em: 18 de Fevereiro de 2009.

[67] MASCHHOFF, K. J., SORENSEN, D. C., "P ARPACK: An Efficient Portable Large Scale Eigenvalue Package for Distributed Memory Parallel Architectures". In: *Applied Parallel Computing in Industrial Problems and Optimization*, Springer-Verlag, 1996.

[68] ARPACK, "ARPACK: Arnoldi Package", http://www.caam.rice.edu/software/ARPACK/, Último acesso em: 18 de Fevereiro de 2009.

[69] SLEPC, "SLEPc: Scalable Library for Eigenvalue Problem Computations", http://www.grycap.upv.es/slepc/, Último acesso em: 18 de Fevereiro de 2009.

[70] HERNANDEZ, V., ROMAN, J. E., VIDAL, V., "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems", *ACM Transactions on Mathematical Software*, v. 31, n. 3, pp. 351–362, 2005.

[71] PETSC, "PETSc: Portable, Extensible Toolkit for Scientific Computation", http://www.mcs.anl.gov/petsc/petsc-as/, Último acesso em: 18 de Fevereiro de 2009.

[72] HERNANDEZ, V., ROMAN, J. E., VIDAL, V., *Krylov-Schur Methods in SLEPc*, Tech. rep., 2007, (Relatório Técnico).

[73] PYTHON, "Python Programming Language - Official Website", `http://www.python.org/`, Último acesso em: 17 de Março de 2009.

[74] UCI, "UCI Machine Learning Repository", `http://archive.ics.uci.edu/ml/`, Último acesso em: 01 de Abril de 2009.

[75] RODRÍGUEZ, J. J., MAUDES, J., "Boosting recombined weak classifiers", *Pattern Recognition Letters*, v. 28, n. 10, pp. 1049–1059, 2008.

[76] WAN, E. A., "Neural Network Classification: A Bayesian Interpretation", *IEEE Transactions on Neural Networks*, v. 1, n. 4, Dezembro 1990.

[77] CHEN, Y., WANG, J. Z., "Support Vector Learning for Fuzzy RUle-Based Classification Systems", *IEEE Transactions on Fuzzy Systems*, v. 11, n. 6, Janeiro 2003.

[78] QUINLAN, J. R., "Fuzzy Sets", *Information and Control*, v. 8, pp. 338 –353, 1965.

[79] STANKOVIC, N., ZHANG, K., "A Distributed Parallel Programming Framework", *IEEE Transactions on Software Engineering*, v. 28, n. 5, pp. 478–493, Maio 2002.

[80] VAPNIK, V. N., *The Nature of Statistical Learning Theory*. Springer: New York, NY, USA, 2000.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo