Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Sergio Oliveira Costa Junior

Modelagem automática de sistemas *fuzzy* do tipo Mamdani baseada em otimização por enxame de partículas

Rio de Janeiro
2010

Sergio Oliveira Costa Junior

Modelagem automática de sistemas *fuzzy* do tipo Mamdani baseada em otimização por enxame de partículas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof^a. Dr^a. Nadia Nedjah
Co-orientadora: Prof^a. Dr^a. Luiza de Macedo Mourelle

Rio de Janeiro
2010

CATALOGAÇÃO NA FONTE
UERJ/REDE SIRIUS/CTC/B

C837 Costa Jr., Sergio Oliveira
Modelagem automática de sistemas *fuzzy* do tipo Mamdani baseada em otimização por enxame de partículas / Sergio Oliveira Costa Junior. – 2010.
115 f. : il.

Orientadora: Prof^a. Dr^a. Nadia Nedjah.
Co-orientadora: Prof^a. Dr^a. Luiza de Macedo Mourelle.
Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia.
Bibliografia: f. 105 – 109.

1. Sistemas fuzzy. 2. Otimização - Enxame de partículas. 3. Automatização. I. Nedjah, Nadia. II. Mourelle, Luiza de Macedo. III. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. IV. Título.

CDU 004.8

Assinatura _____ Data _____

Sergio Oliveira Costa Junior

Modelagem automática de sistemas *fuzzy* do tipo Mamdani baseada em otimização por enxame de partículas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em 15 de Julho de 2010.

Banca Examinadora:

Prof^a. Dr^a. Nadia Nedjah (Orientadora)
Faculdade de Engenharia, UERJ

Prof^a. Dr^a. Luiza de Macedo Mourelle (Co-orientadora)
Faculdade de Engenharia, UERJ

Prof. Dr. José Ernesto De Araujo Filho
Programa de Pós-graduação em Informática em Saúde, UNIFESP

Prof. Dr. Cláudio Márcio do Nascimento Abreu Pereira
Comissão Nacional de Energia Nuclear, CNEN

Rio de Janeiro
2010

AGRADECIMENTOS

A Deus, acima de tudo, por me ter dado a vida e me feito capaz de chegar até aqui;

À minha mãe, Maria Ondina, por dedicar sua vida aos filhos, nos tornar cidadãos de bem e capazes de lutar por nossos sonhos;

Às minhas irmãs e cunhados pelo incentivo a que eu fizesse curso de mestrado;

Aos meus sobrinhos por serem levados e atazanarem os pais;

À adorável gaúcha, Kamile, pelo constante incentivo durante estes anos;

À Universidade do Estado do Rio de Janeiro - UERJ por ter aberto a porta a este ex-aluno agora na condição de mestrando;

Aos funcionários e professores do Programa de Pós-graduação em Engenharia Eletrônica - PEL, por manterem este curso funcionando e nos dando o apoio necessário;

Às minhas orientadoras, professora Nadia Nedjah e professora Luiza de Macedo Mourelle, pelos puxões de orelha, os "e aís...", a dedicação ao trabalho realizado neste curso, e apoio nos momentos mais difíceis deste processo;

À Agência Nacional de Saúde Suplementar - ANS por ter me concedido horário especial quando participava das disciplinas do curso;

Ao professor Jorge Luís Machado do Amaral por ter percebido minhas dificuldades, devido a meu perfil profissional, e elaborado atividades neste perfil quando da realização de sua disciplina;

A todos os colegas do mestrado pelo compartilhamento de idéias e pela companhia agradável.

RESUMO

Junior, Sergio Oliveira Costa *Modelagem automática de sistemas fuzzy utilizando otimização por enxame de partículas*. 2010. 115 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2010.

Esta dissertação investiga a utilização de *Particle Swarm Optimization* (PSO) para a obtenção automática de sistemas *fuzzy* do tipo Mamdani, tendo como insumo apenas as definições das variáveis do problema, seus domínios e a função objetivo. Neste trabalho utilizam-se algumas técnicas conhecidas na tentativa de minimizar a obtenção de sistemas *fuzzy* que não sejam coerentes. As principais técnicas usadas são o método de Wang e Mendell, chamado de WM, para auxiliar na obtenção de regras, e os conceitos de clusterização para obtenção das funções de pertinência. Na função de avaliação proposta, considera-se não somente a acurácia do sistema *fuzzy*, através da medida do erro, mas também a sua *interpretabilidade*, através da medida da compacidade, que consiste da quantidade de regras e funções membro, da *distinguibilidade*, que permite evitar que as funções membro não se confundam, e da *completude*, que permite avaliar que as funções membro abranjam o máximo do domínio. O propósito deste trabalho consiste no desenvolvimento de um algoritmo baseado em PSO, cuja função de avaliação congregue todos esses objetivos. Com parâmetros bem definidos, o algoritmo pode ser utilizado em diversos tipos de problemas sem qualquer alteração, tornando totalmente automática a obtenção de sistemas *fuzzy*. Com este intuito, o algoritmo proposto é testado utilizando alguns problemas pré-selecionados, que foram classificados em dois grupos, com base no tipo de função: contínua ou discreta. Nos testes com funções contínuas, são utilizados sistemas tridimensionais, com duas variáveis de entrada e uma de saída, enquanto nos testes com funções discretas são utilizados problemas de classificação, sendo um com quatro variáveis e outro com seis variáveis de entrada. Os resultados gerados pelo algoritmo proposto são comparados com aqueles obtidos em outros trabalhos.

Palavras-chave: Enxame. Partícula. Otimização. Sistema *fuzzy*.

ABSTRACT

This dissertation investigates the use of Particle Swarm Optimization (PSO) to allow automatic modeling of Mamdani fuzzy systems taking as input only the variable definitions, their respective domains and the objective function. This work uses several known techniques to avoid the consideration of invalid fuzzy systems. The main used techniques are the WM method, which is used to generate rules, and the clustering concept, which assists in the generation of the membership functions. The evaluation function proposed considers not only the accuracy of the generated fuzzy system, but also the properties of interpretability and distinguishability. The accuracy of the fuzzy system is measured using the underlaying error. The system interpretability is evaluated using a compactness measure, which consists mainly of the number of employed rules and membership functions, while its distinguishability is quantified using the completeness measure, which consists of measuring how the used membership functions are covering the corresponding domain. The main goal of this work is to develop a PSO-based algorithm that uses a fitness function which congregates all these objectives. With well-defined parameters, the algorithm can be used with different kinds of problems without any change, allowing for a fully automatic generation process of an adequate fuzzy system. In this purpose, the proposed algorithm is tested for some benchmark problems, which are classified in two groups, based on the type of function to be modeled by the yield fuzzy system: completely or partially defined function. In the cases for fully-defined functions, three-dimensional functions are used. These functions have two input variables and one output variable. In the cases for partially-defined functions, two classification problems are used, one having four variables and other six input variables. The results obtained by the proposed algorithm are compared to related work.

Keywords: Swarm. Particle. Optimization. Fuzzy system.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

INTRODUÇÃO

O MODO como os seres humanos tomam decisões sobre processos complexos, baseados em informações vagas ou aproximadas, serve como suporte para a teoria dos conjuntos *fuzzy*, definida por L.A. Zadeh em 1965 (ZADEH, 1965), e para os conceitos da lógica *fuzzy*. Esta lógica, contrariando a lógica de Aristóteles, na qual uma proposição ou é verdadeira, ou é falsa, prevê que uma proposição não precisa pertencer a apenas um grupo lógico, mas pode pertencer em diferentes graus a grupos lógicos distintos. Por exemplo, a temperatura morna da água de um chuveiro pode ser considerada um pouco fria e um pouco quente, com diferentes graus em cada uma destas possibilidades.

A modelagem *fuzzy* tornou-se uma importante ferramenta a ser utilizada nas diversas áreas do conhecimento humano, por sua eficácia e por ser uma ferramenta que possibilita simular a forma de tomada de decisão do ser humano ao tratar com a vaguidade, como na engenharia, em especial na área de controle, na medicina, na economia etc. Existem diversos exemplos de soluções de problemas utilizando sistemas *fuzzy*, como: classificação multitemporal de imagens (CHANUSSOT; MAURIS; LAMBERT, 1999), avaliação de informações de usuários na internet (GUIMARãES, 2002), gerenciamento de filas em roteadores de alta velocidade (WANG, 2003), controle inteligente de robôs (FIGUEIREDO; VELLASCO; PACHECO, 2000), informações médicas para controle de baixa em campo de batalha (WENDELKEN; MCGRATH; BLIKE, 2003) etc.

Além disso, para aplicar esta técnica não há necessidade de complexas modelagens matemáticas do problema, o que torna o trabalho mais simples. Em contrapartida, para se chegar a um modelo *fuzzy* que seja adequado a um determinado problema, é necessário que um especialista consiga abstrair o problema em questão, elaborando um conjunto de regras e definindo de forma adequada funções de pertinência para cada variável, o que pode tornar-se complicado, principalmente para problemas com muitas variáveis e usuários que não são da área. A eficácia do sistema depende do entendimento do especialista e sua capacidade em definir aqueles componentes *fuzzy*.

A modelagem automática de sistemas *fuzzy* torna o processo menos dependente do es-

pecialista, sendo assim menos suscetível a seus vícios e dificuldades de modelar. Além de evitar erros de modelagem dos parâmetros de um sistema difuso, encurta-se o tempo de desenvolvimento do sistema. As maiores dificuldades encontradas nesta automatização são: *(i)* conseguir gerar um sistema *fuzzy* que seja de fácil interpretação por um ser humano, além de *(ii)* garantir sua acurácia, dois objetivos que muitas vezes acabam sendo conflitantes.

Vários trabalhos foram desenvolvidos nesta área, como o método para gerar regras automaticamente através de exemplos proposto por Wang e Mendell em 1992 (WANG; MENDEL, 1992)(WANG, 2003), chamado de método WM. Um outro trabalho foi o desenvolvimento do algoritmo *pre-shaped fuzzy c-means* (CHEN; CHEN, 2007) que utiliza a clusterização para gerar as funções membro do sistema *fuzzy* considerando a interpretabilidade do modelo.

Algoritmos evolucionários (AE) também são utilizados no intuito de automatizar a produção de sistemas *fuzzy*, tanto de regras, quanto de funções membro, como mostra o artigo (CINTRA; CAMARGO, 2007b), que utiliza algoritmos genéticos (AG) para gerar base de regras, com pré-seleção de regras candidatas, e o artigo (KIM; KIM; LEE, 2006) onde os autores utilizam AE para gerar sistemas *fuzzy* mais compactos e melhor interpretáveis ao ser humano. Além desses, em (SETNES; ROUBOS, 2000) os autores utilizam clusterização e AG para definir bons conjuntos de regras em problemas de classificação, enquanto nos artigos (CORDóN; HERRERA, 1996) e (XUE; CHONG; JAMSHIDI, 1994) os autores utilizam técnica evolucionária e AG para gerar sistemas *fuzzy* a partir de bases de conhecimento.

PSO (*particle swarm optimization*) ou, em português, otimização por enxame de partículas é um algoritmo de otimização, baseado em inteligência de enxame (*swarm inteligence*), empregado em problemas complexos, incluindo problemas a multiobjetivo. Inspirados nos padrões de vôo dos pássaros, em 1995, James Kennedy e Russel Eberhart (KENNEDY; EBERHART, 1995) desenvolveram um algoritmo que é capaz de predizer o comportamento social dos indivíduos do grupo de acordo com objetivos definidos.

O algoritmo PSO pode ser utilizado em diversas áreas do conhecimento para resolver diversos problemas, como o do caixeiro viajante (LOPES; COELHO, 2005), controle de veículo subaquático (GUO et al., 2007), particionamento de hardware e software (ABDELHALIM; SALAMA; HABIB, 2006), escalonamento (*schedule*) de tarefa (GROBLER; ENGELBRECHT; YADAVALLI, 2008), particionamento de *hardware* e *software* (ABDELHALIM; SALAMA; HABIB, 2006) etc. Além disso, por suas características de analisar mais de uma solução por vez, possuir uma busca direcionada às melhores soluções encontradas, simplicidade de implementação e baixo custo de processamento, observa-se que a técnica de PSO pode ser uma boa ferramenta para

auxiliar no desenvolvimento dos sistemas Fuzzy, na busca e definição de alguns ou todos os parâmetros necessários ao bom funcionamento destes (KHOSLA et al., 2006).

O Capítulo 1 deste trabalho apresenta a modelagem *fuzzy*, mostrando o que são os conjuntos *fuzzy*, algumas operações que podem ser feitas sobre os mesmos, o que são as funções de pertinência, e discorrendo sobre os princípios da lógica *fuzzy* e seu sistema de inferência.

O Capítulo 2 mostra como é o algoritmo PSO, a representação do problema, como é feita a atualização das partículas, seu posicionamento no espaço de busca, e os parâmetros de controle da relação de exploração deste espaço de busca.

O Capítulo 3 traz referências de trabalhos que possuem alguma relação com o conteúdo desta dissertação e que serviram de base para sua elaboração. Os artigos utilizados para comparação são descritos neste capítulo.

O Capítulo 4 mostra como foi definido o modelo para a solução do problema em questão. Neste capítulo é feita uma apresentação do problema e são definidos os detalhes do algoritmo desenvolvido, como a forma da representação utilizada nas partículas do PSO, a definição do posicionamento das mesmas, a inicialização do algoritmo, a descrição da função de avaliação e a forma de atualização das soluções em cada iteração.

O Capítulo 5 explica os testes executados para cada problema selecionado, dividido em funções contínuas e discretas, apresentando as comparações com outros trabalhos, e mostrando os resultados encontrados.

O Capítulo 6 apresenta as conclusões obtidas a partir das informações dos testes, e traz algumas propostas de mudança do modelo que poderiam implicar em uma melhoria.

Capítulo 1

SISTEMAS *FUZZY*

ARISTÓTELES , filósofo grego (384 - 322 a.C.) (BLANCHE; DUBUCS, 1899), foi o precursor da ciência da lógica, e estabeleceu regras rígidas para que conclusões pudessem ser aceitas como logicamente válidas. A lógica de Aristóteles, refinada pelos pensadores George Boole, Georg Cantor, David Hilbert e Bertrand Russell, é uma lógica binária (*crisp*), na qual uma proposição é verdadeira ou falsa, não podendo ser, simultaneamente, verdadeira e falsa (COX, 1994).

O modo como os seres humanos tomam decisões sobre processos complexos, baseados em informações vagas ou aproximadas, serve como suporte para a teoria dos conjuntos *fuzzy*, definida por L.A. Zadeh em 1965 (ZADEH, 1965), e para os conceitos da lógica *fuzzy*.

Esta lógica, contrariando a lógica de Aristóteles, prevê que uma proposição não precisa pertencer a apenas um grupo lógico, mas pode pertencer em diferentes graus a grupos lógicos distintos. Por exemplo, a temperatura da água de um chuveiro pode ser considerada um pouco fria e um pouco quente, com diferentes graus em cada uma destas possibilidades.

Através da lógica e da teoria dos conjuntos *fuzzy* pode-se traduzir em termos matemáticos a informação vaga expressa por um conjunto de regras (COX, 1994).

A modelagem *fuzzy* (TANSCHEIT, 2003) é uma importante ferramenta para solução de problemas complexos baseados em informações vagas e de modelagem matemática complexa. Se especialistas forem capazes de formular suas estratégias para a solução de seus problemas como um conjunto de regras, então um algoritmo passível de ser implementado em computador pode ser construído com base na teoria de conjuntos *fuzzy* e na lógica *fuzzy*.

Em 1974, o professor Mamdani (MAMDANI, 1974), do Queen Mary College, Universidade de Londres, obteve sucesso utilizando o raciocínio *fuzzy* para controlar uma máquina a vapor, o que serviu de alavanca para o desenvolvimento de diversas outras aplicações com sistemas *fuzzy*.

Os sistemas *fuzzy* têm sido utilizados com sucesso na solução de diversos problemas, como controle de sistemas não-lineares (ZHAO; BOSE, 2002), previsão de séries temporais (WANG; MENDEL, 1992), classificação e clusterização (CHEN; PENG; ABRAHAM, 2006). Alguns exemplos são, os controladores *fuzzy* de plantas nucleares (ATKIN; ALTIN, 1991), sistema de operação automática de trens (SANKAR; KUMAR, 2006), máquina diesel (PENA et al., 2002).

Modelos *fuzzy* vêm sendo empregados com sucesso em um crescente número de campos de aplicação pelo fato de sua implementação ser barata, sua habilidade de resolver problemas não-lineares de solução complexa, e possuir um comportamento robusto. Além disso, os modelos *fuzzy* provêm uma atrativa alternativa à "caixa preta" característica das redes neurais, pois seu comportamento pode ser explicado em termos linguísticos compreensíveis (KIM; KIM; LEE, 2006).

Neste capítulo serão apresentados os princípios da lógica *fuzzy* e os elementos básicos que compõem um sistema *fuzzy*. Na Seção 1.1.1 serão definidos termos e variáveis linguísticas. A Seção 1.1.2 apresenta os conjuntos *fuzzy*, e as funções de pertinência. Na Seção 1.1.3 será apresentada a lógica *fuzzy*.

1.1 Lógica *Fuzzy*

1.1.1 Variáveis Linguísticas e Termos Linguísticos

Os conceitos referentes a um problema são quantificados, em um sistema *fuzzy*, nas chamadas variáveis linguísticas. As características correspondentes a estas variáveis são chamadas de termos linguísticos (BABUSKA, 1998).

Os termos linguísticos possuem relação qualitativa com o problema a ser solucionado, e facilitam, para um ser humano, o entendimento do significado do que cada característica representa. Por exemplo, no caso da temperatura da água de um chuveiro poderiam ser utilizados os termos linguísticos "quente", "morna" e "fria", para representar três diferentes características.

As variáveis linguísticas são variáveis cujos valores são termos linguísticos. Voltando ao exemplo da água do chuveiro, a variável linguística para representar esta situação poderia ser a "temperatura".

Os valores das variáveis linguísticas são expressões que contemplam termos linguísticos relacionados (frio, quente, alto, baixo, forte), conectivos lógicos (não, e, ou), além de modificadores tais como muito, pouco e delimitadores tais como parênteses e colchetes (TANSCHEIT, 2003).

1.1.2 Conjuntos *Fuzzy*

A teoria clássica dos conjuntos define o conceito de pertinência de forma binária, ou seja, um elemento pertence, ou não, a um determinado conjunto. Assim, pode-se expressar a pertinência de um determinado elemento x, a um conjunto A, conforme a seguinte função (TANSCHEIT, 2003):

$$f(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases} \tag{1}$$

1.1.2.1 Definições sobre os conjuntos *fuzzy*

A teoria dos conjuntos *fuzzy*, proposta por Zadeh, generaliza a função acima possibilitando valores no intervalo [0, 1], desta forma a definição de elementos que são membros e não-membros de um dado conjunto não se dá mais de forma abrupta (COX, 1994). Esta definição passa a ser gradual, desde o valor zero, que expressa o fato de um elemento ser totalmente não membro de um conjunto, até o valor um que, ao contrário, expressa o fato de ser totalmente membro. Assim, pode-se definir um conjunto *fuzzy* A em um universo $\mathbb{X}$ por uma função de pertinência (COX, 1994) $\mu_A(x) : X \rightarrow [0, 1]$, e sua representação por um conjunto de pares ordenados $A = \{(\mu_A(x), x), x \in \mathbb{X}\}$. Então $\mu_A(x)$ indica o quanto x é pertinente ao conjunto A.

Assim como ocorre com os conjuntos ordinários, existem algumas definições envolvendo conjuntos *fuzzy* (ZADEH, 1965). Considerando A e B como sendo dois conjuntos *fuzzy* quaisquer, serão apresentadas, nesta seção, algumas definições relativas a estes conjuntos.

Definição 1 (Conjunto Vazio). *Um conjunto fuzzy A em um universo $\mathbb{X}$ é vazio se e somente se sua função de pertinência é igual a zero em todo $\mathbb{X}$:*

$$A = \emptyset \quad se \quad \mu_A(x) = 0, \quad \forall x \in \mathbb{X} \tag{2}$$

Definição 2 (Conjuntos Iguais). *Dois conjuntos fuzzy A e B em um universo $\mathbb{X}$ são iguais se suas funções de pertinência tiverem o mesmo valor em todo domínio $\mathbb{X}$:*

$$A = B \quad se \quad \mu_A(x) = \mu_B(x), \quad \forall x \in \mathbb{X} \tag{3}$$

Definição 3 (Subconjunto). *Um conjunto fuzzy A é um subconjunto de B se sua função de pertinência for menor ou igual à de B em todo domínio $\mathbb{X}$:*

$$A \subset B \quad se \quad \mu_A(x) \leq \mu_B(x), \quad \forall x \in \mathbb{X} \tag{4}$$

1.1.2.2 Operações sobre os conjuntos *fuzzy*

Nesta seção serão apresentadas algumas definições importantes de operações que se aplicam a conjuntos *fuzzy* (ZADEH, 1965).

Definição 4 (Complemento de um Conjunto). *O complemento A' de um conjunto fuzzy A é definido por:*

$$\mu_{A'}(x) = 1 - \mu_A(x), \quad \forall x \in \mathbb{X} \tag{5}$$

Definição 5 (União de Conjuntos). *A função de pertinência para a união de dois conjuntos A e B, sobre o domínio $\mathbb{X}$, é definida como:*

$$\mu_{A \cup B}(x) = max(\mu_A(x), \mu_B(x)), \quad \forall x \in \mathbb{X} \tag{6}$$

Definição 6 (Interseção de Conjuntos). *A função de pertinência para a interseção de dois conjuntos A e B, sobre o domínio $\mathbb{X}$, é definida como:*

$$\mu_{A \cap B}(x) = min(\mu_A(x), \mu_B(x)), \quad \forall x \in \mathbb{X} \tag{7}$$

Tanto a união, como a interseção podem ser representadas por outros operadores, além de max e min, respectivamente. Um outro operador, sugerido por Zadeh (ZADEH, 1965), para estas duas operações foi, a soma algébrica para a união,

$$\mu_{A \cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x), \quad \forall x \in \mathbb{X} \tag{8}$$

e o produto algébrico para a interseção,

$$\mu_{A \cap B}(x) = \mu_A(x)\mu_B(x), \quad \forall x \in \mathbb{X}. \tag{9}$$

A diferença entre estas representações, encontra-se na participação dos conjuntos no resultado final. Quando os operadores utilizados são max e min, apenas um dos conjuntos influencia no resultado da interseção ou união, mas quando são utilizados os operadores de soma ou produto algébrico, o resultado será uma combinação dos graus de pertinência da variável nesses conjuntos.

Com o objetivo de generalização da teoria dos conjuntos *fuzzy*, foram definidos operadores de base axiomática, baseados no conceito de *norma triangular* (*norma-t*) e *co-norma triangular* (*co-norma-t* ou *norma-s*)(TANSCHEIT, 2003)(NEDJAH et al., 2005).

Definição 7 (Norma-t). *Uma norma-t é uma operação binária $* : [0,1] \times [0,1] \rightarrow [0,1]$ tal que, $\forall x, y, z, w \in [0,1]$, são satisfeitas as propriedades abaixo:*

- *Comutatividade:* $x * y = y * x$
- *Associatividade:* $(x * y) * z = x * (y * z)$
- *Monotonicidade: se* $x \leq y$, $w \leq z$, *então* $x * w \leq y * z$
- *Condições de contorno:* $x * 0 = 0$ *e* $x * 1 = x$

Deste modo, qualquer operação que satisfaz as propriedades acima, pode ser chamada de *norma-t*.

Definição 8 (Co-norma-t ou Norma-s). *Uma co-norma-t, ou uma norma-s é uma operação binária* $\oplus : [0,1] \times [0,1] \rightarrow [0,1]$, *que satisfaz as propriedades abaixo:*

- *Comutatividade:* $x \oplus y = y \oplus x$
- *Associatividade:* $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- *Monotonicidade: se* $x \leq y$, $w \leq z$, *então* $x \oplus w \leq y \oplus z$
- *Condições de contorno:* $x \oplus 0 = x$ *e* $x \oplus 1 = 1$

Assim, qualquer operação que satisfaz as propriedades acima, pode ser chamada de *co-norma-t*.

Existem inúmeras *normas-t* e *co-normas-t*, como informa o artigo (TANSCHEIT, 2003) e os exemplos do artigo (DETYNIECKI; YAGER; BOUCHON-MEUNIER, 2000). Além disso, (TANSCHEIT, 2003) informa que em aplicações da engenharia têm sido utilizados preponderantemente os operadores min e produto algébrico para interseção e max para união.

1.1.2.3 Funções de pertinência

A função de pertinência $\mu_A(x) : X \rightarrow [0,1]$ mapeia cada elemento, valor ou ponto x do domínio X, a um número (grau) no intervalo entre entre os reais $[0,1]$. A função de pertinência $\mu_A(x)$ pode ser entendida como sendo o grau de compatibilidade entre o elemento x e o conceito expresso por A (ARAUJO, 2009).

Existem diversas formas possíveis para representar funções de pertinência (ZHAO; BOSE, 2002), e a escolha por uma forma ou outra depende do problema em questão (do conceito que se deseja representar) e do conhecimento e experiência do especialista. Duas pessoas podem descrever funções de pertinência diferentes para a mesma variável de um mesmo problema

(TANSCHEIT, 2003), pois isso depende totalmente do entendimento, e ponto de vista, de cada um sobre o problema em questão.

É comum que se faça uso de funções de pertinência padrão (TANSCHEIT, 2003) (COX, 1994) , como, por exemplo, funções triangulares, trapezoidais, Gaussianas, sigmóides etc., mas um especialista poderia definir funções diferentes a partir de sua própria experiência. Não há regras que definam em que tipo de problemas utilizar este ou aquele tipo de função de pertinência.

Abaixo, nas Definições 9 a 13, apresentam-se exemplos (ZHAO; BOSE, 2002) de algumas destas funções de pertinência.

Definição 9 (Função triangular). *Uma função triangular pode ser descrita por três parâmetros a, b e c (ZHAO; BOSE, 2002), e definida conforme a Equação 10*

$$f(x; a, b, c) = max\left\{min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right\}, \tag{10}$$

onde a e c formam a base e b é o pico, como mostra a Figura 1.

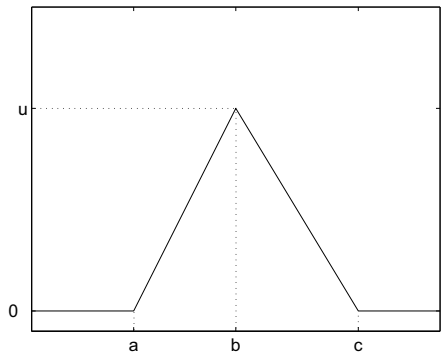


Figura 1: Função triangular

Definição 10. *Função trapezoidal Uma função trapezoidal pode ser descrita por quatro parâmetros a, b, c e d (ZHAO; BOSE, 2002), e definida conforme a Equação 11*

$$f(x; a, b, c, d) = max\left\{min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right\}, \tag{11}$$

onde a e d formam a base, e b e c o topo, como mostra a Figura 2.

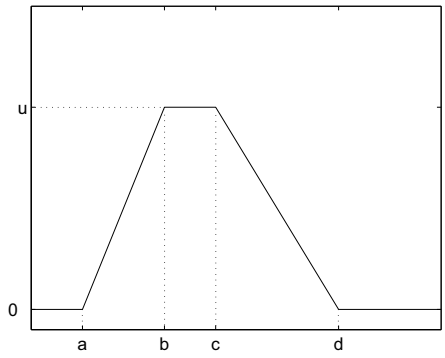


Figura 2: Função trapezoidal

Definição 11. *Função Gaussiana Uma função Gaussiana simétrica (ZHAO; BOSE, 2002), como apresentada na Figura 3, pode ser definida conforme a Equação 12*

$$f(x;\sigma,c) = e^{\frac{-(x-c)^2}{2\sigma^2}}, \tag{12}$$

onde o parâmetro c indica a distância da função à origem e σ indica a largura da curva.

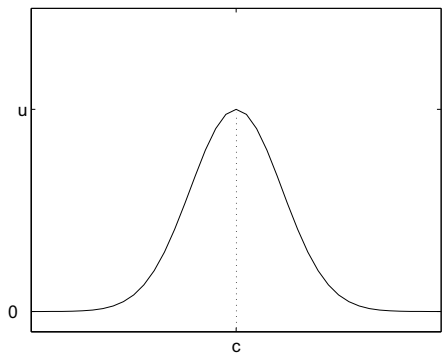
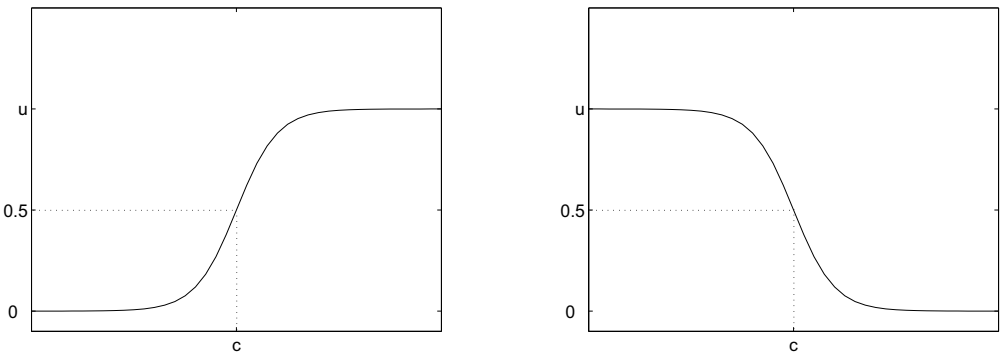


Figura 3: Função Gaussiana

Definição 12. *Função sigmóide Uma função sigmóide pode ter sua inclinação direcionada para a direita ou para a esquerda (ZHAO; BOSE, 2002), como mostram as Figuras 1.4(a) e 1.4(b), respectivamente, e pode ser definida conforme a Equação 13*

$$f(x;a,c) = \frac{1}{1+e^{-a(x-c)}}, \tag{13}$$

onde o parâmetro c indica a distância da função à origem e a determina a inclinação da função.



(a) Função sigmóide direita (b) Função sigmóide esquerda

Figura 4: Funções sigmóide

Definição 13. *Função sino Uma função sino possui forma simétrica (ZHAO; BOSE, 2002), como mostra a Figura 5, e pode ser definida conforme a Equação 14*

$$f(x; a, b, c) = \frac{1}{1 + |\frac{x-c}{a}|^{2b}}, \tag{14}$$

onde o parâmetro b em geral é positivo, c indica a distância da função à origem e a determina a largura da curva.

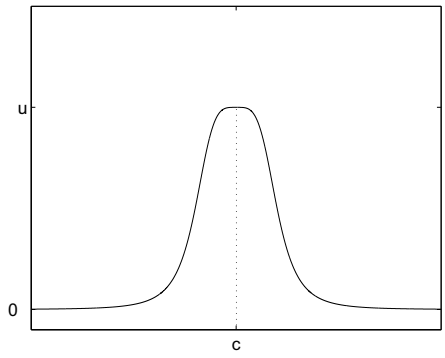


Figura 5: Função sino

Para o exemplo do chuveiro, Seção 1.1.1, pode-se representar as funções de pertinência, referentes à variável "temperatura", da forma mostrada na Figura 6 onde aos conjuntos *fuzzy*, que chamaremos de A, B e C, correspondem os termos linguísticos "fria", "morna"e "quente".

No exemplo da Figura 6, temperaturas de até 10°C apresentam grau de pertinência igual a 1 no conjunto A (fria), e o grau de pertinência neste conjunto decresce à medida que a temperatura aumenta, até que seja zero. De maneira oposta o grau de pertinência no conjunto

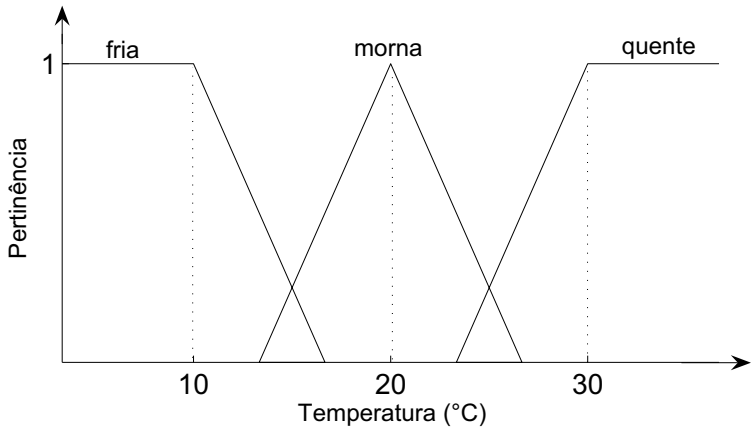


Figura 6: Funções de Pertinência para a variável temperatura

B (morna), que era zero até um valor de temperatura um pouco maior que dez, cresce à medida que a temperatura aumenta, até chegar a um quando a temperatura é de 20ºC.

1.1.3 Princípios de Lógica *Fuzzy*

Definição 14 (Proposições *Fuzzy*). *Seja x o nome de uma variável linguística e A um conjunto fuzzy (ou termo linguístico), uma proposição fuzzy é uma frase da forma x é A.*

Alguns exemplos de proposições: *temperatura* é *quente*, *estatura* é *baixa* e *risco* é *alto*.

Proposições podem ser combinadas utilizando-se de operadores, como os conectivos lógicos *e* e *ou*, ou o operador de negação *não*, além de serem relacionadas pelo operador de implicação *se...então*, neste caso tem-se uma declaração condicional *fuzzy*, também chamada de regra *fuzzy* ou regra linguística (TANSCHEIT, 2003).

A operação de negação, *não*, é utilizada da mesma forma que a linguagem natural. Por exemplo, pressão é *não alta*. E em termos de função de transferência: $A = \mu_A(x)/x \Rightarrow$ *não* $A = (1 - \mu_A(x))/x$.

Um exemplo de regra *fuzzy* para um sistema de temperatura e pressão poderia ser, *se temperatura* é *quente* e *volume* é *pequeno então pressão* é *alta*.

Cada proposição da primeira parte de uma regra R, antes da palavra *então*, é chamada de *antecedente*, enquanto cada proposição da segunda parte, após a mesma palavra, é chamada de *consequente* de R.

Um sistema *fuzzy* possui uma ou mais regras, que são utilizadas para gerar a saída do sistema *fuzzy* no processo de inferência (Seção 1.1.4). O agrupamento dessas regras é

chamado de conjunto de regras do sistema. As regras deste conjunto podem ser obtidas de um especialista, ou extraídas de uma base de conhecimento.

A combinação das proposições do antecedente e do consequente da regra *fuzzy* é realizada pelas operações de implicação e por operações de conjunção (norma-t) e disjunção (co-norma-t) (ARAUJO, 2009).

A lógica difusa tem seu núcleo, como outros sistemas lógicos, em um sistema de regras de inferência. O mecanismo de inferência difuso utiliza os princípios da lógica para determinar como os fatos e as regras devem ser combinados para gerar novos fatos. A inferência em avanço baseada em dados normalmente empregada aos sistemas difusos é a regra de implicação difusa denominada modus ponens generalizado (ARAUJO, 2009), Equação 15.

$$\begin{cases} \text{premissa 1:} & x \text{ é } A' \\ \text{premissa 2:} & \text{se } x \text{ é } A \text{ então } y \text{ é } B \\ \text{consequente:} & y \text{ é } B' \end{cases} \tag{15}$$

1.1.4 Sistema de Inferência *Fuzzy*

Através do sistema de inferência as regras são processadas utilizando as variáveis e conjuntos *fuzzy*, e uma saída é gerada. Os valores, entradas precisas, que foram atribuídos às variáveis de entrada, passam pelo *fuzificador* que os transforma em conjuntos *fuzzy* de entrada, ou seja, o *fuzificador* é que define um valor real de entrada em função de conjuntos *fuzzy* utilizando o gráfico de pertinência das variáveis. Caso a entrada seja fornecida em função de conjuntos *fuzzy* não há necessidade de um *fuzzificador*.

Em seguida este resultado passa pelo módulo de inferência, que utiliza o conjunto de regras para definir os conjuntos *fuzzy* de saída. No processo de inferência algumas regras são ativadas e outras não, e isso é o que define a saída fuzificada. São ativadas apenas as regras em que algum dos conjuntos de suas proposições possui pertinência maior que zero.

A Figura 7 mostra um sistema de inferência *fuzzy* (TANSCHEIT, 2003).

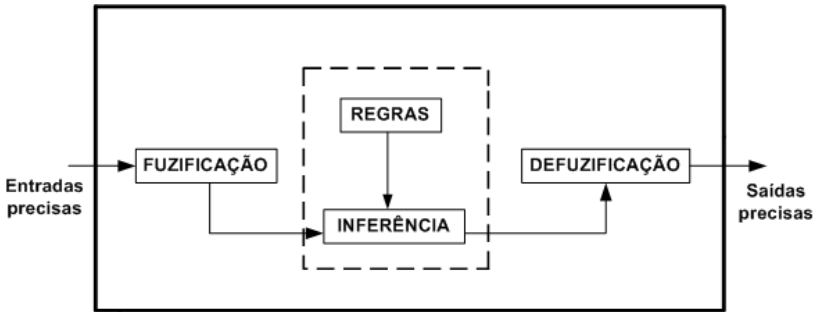


Figura 7: Sistema de Inferência *Fuzzy*

Supondo um sistema *fuzzy* de exemplo, em que tenha três variáveis linguísticas, x, y e z. A variável x tenha conjuntos *fuzzy* denominados $A1$ e $A2$, a variável y, conjuntos denominados $B1$ e $B2$, e a variável z os conjuntos $C1$ e $C2$. Se para uma determinada entrada, a variável x tivesse pertinência diferente de zero apenas no conjunto $A1$, y apenas em $B1$ e z apenas em $C1$, a única regra a ser ativada seria a regra $R1$ da Equação 16.

$$\begin{array}{lllllll} R1: & se & (x \text{ é } A1) & e & (y \text{ é } B1) & \text{então} & (z \text{ é } C1) \\ R2: & se & (x \text{ é } A2) & e & (y \text{ é } B2) & \text{então} & (z \text{ é } C2) \end{array} \tag{16}$$

Supondo agora que x possui 0.20 de pertinência no conjunto $A1$ e 0.75 em $A2$, e ainda que y possui 0.33 de pertinência em $B1$ e 0.60 em $B2$. Selecionando o operador max ($\vee$) para união e min ($\wedge$) para a intersecção, utilizando a Equação 16 podem ser montadas equações que relacionem as pertinências dos antecedentes e consequentes, assim teremos a Equação 17.

$$\begin{array}{l} \mu_{C1^*}(z) = (\mu_{A1}(x) \wedge \mu_{B1}(y)) \wedge \mu_{C1}(z) = (0.20 \wedge 0.33) \wedge \mu_{C1}(z) \\ \mu_{C2^*}(z) = (\mu_{A2}(x) \wedge \mu_{B2}(y)) \wedge \mu_{C2}(z) = (0.75 \wedge 0.60) \wedge \mu_{C2}(z) \end{array} \tag{17}$$

A partir da Equação 17, aplicando o primeiro operador min, chegamos às relações definidas na Equação 18.

$$\begin{array}{l} \mu_{C1^*}(z) = 0.20 \wedge \mu_{C1}(z) \\ \mu_{C2^*}(z) = 0.60 \wedge \mu_{C2}(z) \end{array} \tag{18}$$

Para este exemplo o gráfico mostrado na Figura 8 será considerado como representação das funções de pertinência da variável de saída z, nele pode-se observar a forma das funções $C1$ e $C2$.

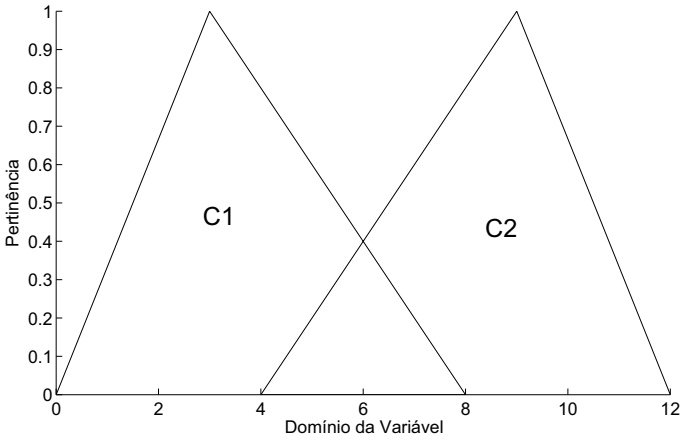


Figura 8: Funções da Variável z

Analisando a Equação 18 e utilizando o gráfico da Figura 8 chega-se ao resultado da inferência, que foi representado na Figura 9. No caso deste exemplo $C1$ e $C2$ são funções triangulares, mas poderiam ser de outra forma, conforme foi descrito na Seção 1.1.2.3. O

gráfico representa a participação destes conjuntos na saída do sistema em linha mais grossa e contínua, enquanto as funções originais estão representadas em linha pontilhada.

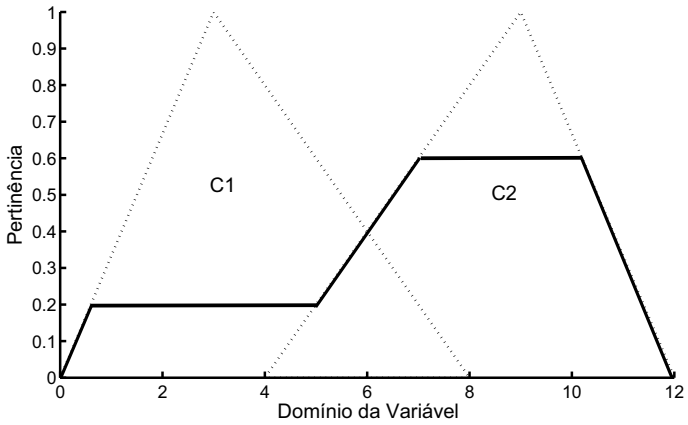


Figura 9: Resultado da Inferência

Finalizado o processo de inferência, o próximo passo é passar pelo *defuzificador*, que é o processo que gera um valor preciso de saída, utilizando-se para isso do gráfico resultante da inferência, no caso deste exemplo, este resultado é mostrado na Figura 9. Existem diversos métodos de defuzificação, dois dos mais utilizados são o *centro de gravidade* e a *média dos máximos* (TANSCHEIT, 2003). O que se faz no processo de defuzificação é utilizar um destes métodos no gráfico resultante da inferência e assim obter o valor de saída.

O sistema de inferência abordado neste trabalho, foi aquele concebido por Zadeh e outros pesquisadores, entre os quais Mamdani, que deu início a trabalhos práticos na década de 70 (MAMDANI, 1974) (MAMDANI, 1977). Por isso, este tipo de sistema de inferência vem sendo referenciado como sendo do tipo *Mamdani* (TANSCHEIT, 2003). Mas existe um outro sistema de inferência, que se tornou extremamente bem sucedido tendo sido concebido por H. Takagi e M. Sugeno (TAKAGI; SUGENO, 1985), o qual difere do sistema de Mamdani na parte do consequente, que é uma função linear das variáveis dos antecedentes, como mostra a Equação 19 (TANSCHEIT, 2003).

$$se \quad (x \text{ é } A1) \quad e \quad (y \text{ é } B1) \quad \text{então} \quad z = f(x, y) \tag{19}$$

A função f é, em geral, um polinômio e o sistema de inferência é geralmente referenciado em função do grau deste polinômio. Por exemplo, em um sistema de inferência *Takagi-Sugeno* de ordem zero a saída z é uma constante (TANSCHEIT, 2003).

1.2 Considerações Finais do Capítulo

Neste capítulo foi apresentada uma breve descrição de sistemas *fuzzy*, o que são os termos e as variáveis linguísticas, os conjuntos *fuzzy*, suas características e as funções de pertinência, e por fim uma explicação sobre o sistema de inferência.

Capítulo 2

OTIMIZAÇÃO BASEADA EM ENXAME DE PARTÍCULAS

INTELIGÊNCIA de enxame (*Swarm Intelligence*) é uma área da inteligência artificial baseada no comportamento coletivo e descentralizado de indivíduos que interagem uns com os outros e com o ambiente. O termo foi definido por Gerardo Beni e Jing Wang em 1989 no contexto dos sistemas robóticos celulares (BENI; WANG, 1989). É um paradigma inovador de inteligência distribuída, inspirado, originalmente, em exemplos biológicos como os bandos, rebanhos e cardumes de animais (ENGELBRECHT, 2005).

Existem diversos modelos baseados neste conceito, alguns descrevem partículas em nuvens, outros são baseados em grupos e comportamento social do ser humano em geral. Além destes, há também modelos baseados no comportamento social de bactérias, aranhas, abelhas e tubarões (ENGELBRECHT, 2005). Todavia, a otimização baseada em enxame de partículas, em inglês *particle swarm optimization* (PSO), tem recebido grande atenção nos últimos tempos.

No PSO procura-se imitar o comportamento social de grupos de animais, mais especificamente de bandos de pássaros. Se um dos elementos do grupo descobre um caminho onde há facilidade para encontrar o alimento, os outros componentes do grupo tendem, instantaneamente, a seguir também este caminho (KENNEDY; EBERHART, 1995).

Hoje em dia os desenvolvedores preferem trabalhar com diversos pequenos agentes autônomos ao invés de desenvolver complexos sistemas centralizados. Um agente reage a regras simples. Os vários agentes cooperando com o sistema podem resolver sistemas muito complexos com um mínimo esforço de desenvolvimento. No geral, sistemas multi-agentes que utilizam alguma forma de inteligência de grupo são chamados de sistemas de inteligência coletiva. Eles são muito utilizados como ferramentas de busca e de otimização (KHOSLA et al., 2006).

Este capítulo apresenta a otimização baseada em enxame de partículas (PSO). Na Seção 2.1 há uma introdução ao PSO, e na Seção 2.1.1 é descrito como é feita a representação de

um problema, na Seção 2.1.2 são apresentados os dois modelos básicos de PSO, na Seção 2.2.1 cada termo que compõe a equação da velocidade, na Seção 2.2.2 como é feita a atualização da posição das partículas, na Seção 2.2.3 possíveis valores para os parâmetros utilizados neste algoritmo, na Seção 2.3 como se processa o algoritmo básico, e na Seção 2.4 algumas das possíveis topologias.

2.1 Introdução ao PSO

O PSO é um algoritmo baseado em inteligência coletiva, estocástico e iterativo, o qual procura a solução de problemas de otimização em um determinado espaço de busca e é capaz de emular o comportamento social de indivíduos de acordo com objetivos definidos. Foi inicialmente desenvolvido como uma ferramenta de simulação dos padrões de vôo dos pássaros em busca de comida e proteção (KENNEDY; EBERHART, 1995), mas Kennedy e Eberhart (KHOSLA et al., 2006)(KENNEDY; EBERHART, 1995) observaram que este comportamento dos pássaros poderia ser adaptado e utilizado em processos de otimização, criando assim a primeira versão simples do PSO. Em PSO, cada partícula possui velocidade e direção adaptativas (KENNEDY; EBERHART, 1995) que determinam sua movimentação no espaço de busca. A partícula é também dotada de uma memória que a torna capaz de lembrar sua melhor posição anterior.

O algoritmo PSO pode ser utilizado em diversas áreas do conhecimento para resolver diversos problemas, como o do caixeiro viajante (LOPES; COELHO, 2005), controle de veículo subaquático (GUO et al., 2007), particionamento de hardware e software (ABDELHALIM; SALAMA; HABIB, 2006) etc. Além disso, por suas características, observa-se que a técnica de PSO pode ser uma boa ferramenta para auxiliar no desenvolvimento dos sistemas Fuzzy, na busca e definição de alguns ou todos os parâmetros necessários ao bom funcionamento destes, que definem o espaço de busca do algoritmo (KHOSLA et al., 2006).

2.1.1 Representação

O PSO é formado por um conjunto de partículas, sendo cada uma delas uma solução potencial do problema, possuindo coordenadas de posição em um espaço de busca n-dimensional. Deste modo, cada partícula possui um vetor de posição, um vetor de melhor posição, um campo para a aptidão e outro para a melhor aptidão. O PSO foi inicialmente definido para espaços de busca contínuos, apesar de também terem sido desenvolvidas versões para espaço de busca discreto (ENGELBRECHT, 2005).

2.1.2 Modelo completo do PSO

Para a atualização da posição de cada partícula i do algoritmo PSO é definida uma velocidade para cada dimensão j desta posição. A velocidade é o elemento que promove a capacidade de locomoção das partículas, e pode ser calculada conforme as Equações 20 e 21 (KENNEDY; EBERHART, 1995) (ENGELBRECHT, 2005) (KHOSLA et al., 2006). Nesta seção serão descritos os dois algoritmos básicos do PSO, nos quais a velocidade depende dos componentes cognitivo e social (vizinhança), que foram denominados de modelo completo do PSO (ENGELBRECHT, 2005). Estes algoritmos são chamados de *melhor global* (*global best* PSO) e *melhor local* (*local best* PSO).

Melhor global Neste algoritmo a vizinhança de cada partícula são todas as partículas da população. Desta forma, ele reflete uma topologia em estrela, mostrada na Seção 2.4, em que a componente social da velocidade de uma partícula é influenciada por todas as outras partículas (ENGELBRECHT, 2005).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(\hat{x}_{ij}(t) - x_{ij}(t)) + c_2 r_2(\bar{x}_j(t) - x_{ij}(t)), \tag{20}$$

onde w é chamado de coeficiente de inércia, r1 e r2 são números aleatórios definidos no intervalo [0, 1], c1 e c2 são constantes positivas, $\hat{x}_{ij}(t)$ é a melhor posição atingida pela partícula i, dimensão j, no passado e $\bar{x}_j(t)$ é a melhor posição, na dimensão j, atingida no passado, entre todas as partículas.

Melhor local Neste algoritmo é definida uma pequena vizinhança para cada partícula. Desta forma, ele reflete uma topologia em anel, mostrada na Seção 2.4, em que a componente social da velocidade de uma partícula é influenciada por esta pequena vizinhança (ENGELBRECHT, 2005).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(\hat{x}_{ij}(t) - x_{ij}(t)) + c_2 r_2(\bar{x}_{ij}(t) - x_{ij}(t)), \tag{21}$$

onde $\bar{x}_{ij}(t)$ é a melhor posição, na dimensão j, atingida no passado, entre todas as partículas na vizinhança da partícula i.

O coeficiente de inércia não existia no PSO proposto inicialmente por Kennedy e Eberhart em 1995 (KENNEDY; EBERHART, 1995), mas foi inserido pelo próprio Eberhart em 1998 (SHI; EBERHART, 1998). O efeito de não ter o coeficiente de inércia é o mesmo de fazer $w = 1$ nas Equações 20 e 21.

2.2 Exploração do espaço de busca

Um aspecto importante dos algoritmos de otimização é a relação entre *exploration* e *exploitation* (ENGELBRECHT, 2005). *Exploration* é a habilidade do algoritmo em explorar diferentes regiões do espaço de busca na tentativa de encontrar uma solução ótima, ou seja, explorar novas soluções. *Exploitation*, ao contrário, é a habilidade do algoritmo executar uma busca mais refinada em determinada região do espaço de busca que apresente resultados mais relevantes, ou seja, explorar os resultados já encontrados.

Um bom algoritmo de otimização deve balancear da melhor forma possível estes dois critérios, que são contraditórios. No PSO este balanceamento é feito pela equação da velocidade, Equações 20 e 21. Sendo limitado pela velocidade máxima, como informa a Seção 2.2.2, Equação 23.

Na Seção 2.1.2, pode-se observar duas equações para a velocidade, que definem o tipo de relação de cada partícula com sua vizinhança, representadas pelos algoritmos do melhor global e melhor local.

2.2.1 Componentes da Velocidade

Nas Equações 20 e 21 podem ser observados três termos que compõem o cálculo da velocidade (ENGELBRECHT, 2005), que são:

- A **velocidade anterior**, $wv_{ij}(t)$, que é como uma memória da direção do movimento no passado mais recente. Este termo pode ser visto como um momento, prevenindo a partícula de mudanças drásticas em sua direção. Esta componente também é chamada de componente de inércia.

- O **componente cognitivo**, $c_1 r_1(\hat{x}_{ij}(t) - x_{ij}(t))$, que quantifica o desempenho da partícula i em relação a performances anteriores. Este termo tem o efeito de atrair a partícula para sua melhor posição no passado. Esta componente foi definida por Kennedy e Eberhart como "nostalgia" da partícula (KENNEDY; EBERHART, 1995).

- O **componente social**, $c_2 r_2(\bar{x}_j(t) - x_{ij}(t))$ ou $c_2 r_2(\bar{x}_{ij}(t) - x_{ij}(t))$, que quantifica a performance da partícula i em relação o desempenho de sua vizinhança. Este termo tem o efeito de atrair a partícula para a melhor posição encontrada pelo grupo de partículas.

2.2.2 Posição da partícula

A posição de cada partícula é alterada conforme a Equação 22.

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1). \tag{22}$$

A velocidade guia o processo de otimização refletindo tanto a experiência da partícula, quanto a troca de informação entre partículas (ENGELBRECHT, 2005). O conhecimento experimental de cada partícula refere-se ao componente cognitivo, que é proporcional à distância entre a partícula e sua melhor posição, encontrada desde a primeira iteração. A troca de informação entre partículas refere-se ao componente social das equações da velocidade (Equações 20 e 21).

Ao atualizar a posição de uma partícula pode ocorrer um fato indesejável, que é a nova posição não estar dentro do domínio definido para o espaço de busca. Este fato pode ser minimizado limitando a velocidade com um parâmetro chamado de velocidade máxima v_{max} (ENGELBRECHT, 2005) (KHOSLA et al., 2006). Tal parâmetro limita o salto na alteração da posição das partículas, de forma a permitir uma maior granularidade no controle da busca, apesar de não evitar que a partícula saia do espaço de busca. Com isso, antes de executar a atualização definida pela Equação 22, a velocidade passa pelo critério definido na Equação 23.

$$v_i(t+1) = \begin{cases} v_i(t+1) & \text{se} \quad v_i(t+1) < v_{max} \\ v_{max} & \text{se} \quad v_i(t+1) \geq v_{max} \end{cases} \tag{23}$$

Para evitar que a partícula realmente deixe o espaço de busca, faz-se necessário um controle da posição, onde pode-se limitar o valor da nova posição ao limite do domínio da variável, ou ainda, utilizar a reflexão.

2.2.3 Valores de parâmetros

O valor de cada parâmetro do algoritmo PSO é fundamental na evolução do processo de busca, e por isso a importância de se definir valores adequados. Abaixo são relacionados alguns valores utilizados em trabalhos anteriores.

- O **coeficiente de inércia** w, controla a relação entre explorar grandes espaços ou uma determinada localidade (SHI; EBERHART, 1998). Em geral, são utilizados valores próximos de um, mas não maiores (ENGELBRECHT, 2005), e nem muito próximos de zero. Valores maiores que um tendem a deixar as partículas com uma aceleração muito alta tendendo a haver grande divergência, enquanto valores muito menores, próximos de zero,

podem desacelerar demais a busca. Ultimamente tem-se utilizado este parâmetro variando durante o processamento do algoritmo, começando com valor alto e diminuindo gradativamente.

- O **coeficiente cognitivo** (c_1) e o **coeficiente social** (c_2) levam o algoritmo a um desempenho melhor se forem balanceados, ou seja, $c_1 \cong c_2$ (ENGELBRECHT, 2005). Além disso, de acordo com (KHOSLA et al., 2006), trabalhos recentes indicam que é melhor que o coeficiente cognitivo seja maior que o social, e que $c_1 + c_2 \leq 4$. Esta referência indica também que bons resultados foram alcançados utilizando $c_1 = c_2 = 1.49$.

- Os **fatores** r_1 e r_2 definem o teor estocástico das contribuições cognitiva e social do algoritmo. São selecionados valores aleatórios no intervalo [0, 1] (ENGELBRECHT, 2005) para cada um dos fatores.

- A **velocidade máxima** é definida para cada dimensão do espaço de busca, e pode ser formulada como um percentual do domínio (ENGELBRECHT, 2005) (KHOSLA et al., 2006), $v_{max} = \delta(x_{max} - x_{min})$, onde x_{max} e x_{min} são, respectivamente, o valor máximo e mínimo do domínio, e δ é um valor no intervalo [0, 1].

- A **quantidade de partículas** define a possibilidade de abranger uma determinada porção do espaço de busca em cada iteração do algoritmo. Um número grande de partículas permite uma maior abrangência, mas exige um maior poder computacional. De acordo com (ENGELBRECHT, 2005) estudos empíricos mostraram que o PSO consegue chegar a soluções ótimas utilizando pequeno número de partículas, entre dez e trinta.

2.3 O Algoritmo PSO

Considerando pBest a melhor posição encontrada para uma determinada partícula e gBest a melhor posição encontrada entre todas as partículas de uma determinada vizinhança (informações sobre vizinhanças na seção 2.1.2), em um momento específico, pode-se definir como se processa o PSO (ENGELBRECHT, 2005) (KHOSLA et al., 2006).

No Algoritmo 1, pode-se observar que primeiro as partículas são inicializadas com valores aleatórios de posição e velocidade. A inicialização das informações da partícula, refere-se à estrutura do problema que cada uma representa. Esta inicialização pode dar-se de forma aleatória, ou pode ser definida por algum algoritmo específico.

Algoritmo 1 PSO

```
1: Para i = 1 até total_particulas Faça
2:    Inicializa informações da partícula;
3:    Inicializa posição da partícula aleatoriamente;
4:    Inicializa velocidade da partícula aleatoriamente;
5: Fim Para
6: Repita
7:    Para i = 1 até total_particulas Faça
8:       Calcula o fitness da partícula;
9:       Se (fitness melhor que de pBest) Então
10:         Atualiza pBest com a nova posição;
11:      Fim Se
12:      Se (fitness melhor que de gBest) Então
13:         Atualiza gBest com a nova posição;
14:      Fim Se
15:      Atualiza a velocidade da partícula;
16:      Atualiza a posição da partícula;
17:   Fim Para
18:   Verifica critério de parada;
19: Até (criterio_parada = true)
```

Após a inicialização, o algoritmo entra no processo iterativo, onde cada partícula é avaliada (relação entre a partícula e o problema). Caso a avaliação seja melhor do que a melhor avaliação encontrada para a partícula anteriormente, a referência à melhor posição é alterada para a posição atual (pBest). Caso a avaliação seja melhor do que a melhor avaliação encontrada anteriormente, entre todas as partículas da vizinhança, a referência a esta posição é alterada para a posição atual (gBest). Em seguida, a velocidade da partícula é atualizada conforme a Equação 20 ou a Equação 21 e a posição é atualizada conforme a Equação 22. Finalmente o critério de parada é verificado, caso seja alcançado o algoritmo pára e o resultado do processamento encontra-se na partícula que está na melhor posição encontrada, caso contrário segue-se para a próxima iteração.

O critério de parada é totalmente dependente do problema em questão, um valor máximo de avaliação e/ou um número máximo de iterações, são parâmetros que podem fazer parte do critério de parada.

2.4 Topologias

A característica principal de um algoritmo PSO é a interação social (ENGELBRECHT, 2005), que faz com que os indivíduos aprendam com o grupo, e utilizem o conhecimento obtido, em seus próprios movimentos, procurando seguir o caminho daquele que obteve mais sucesso. Análogo

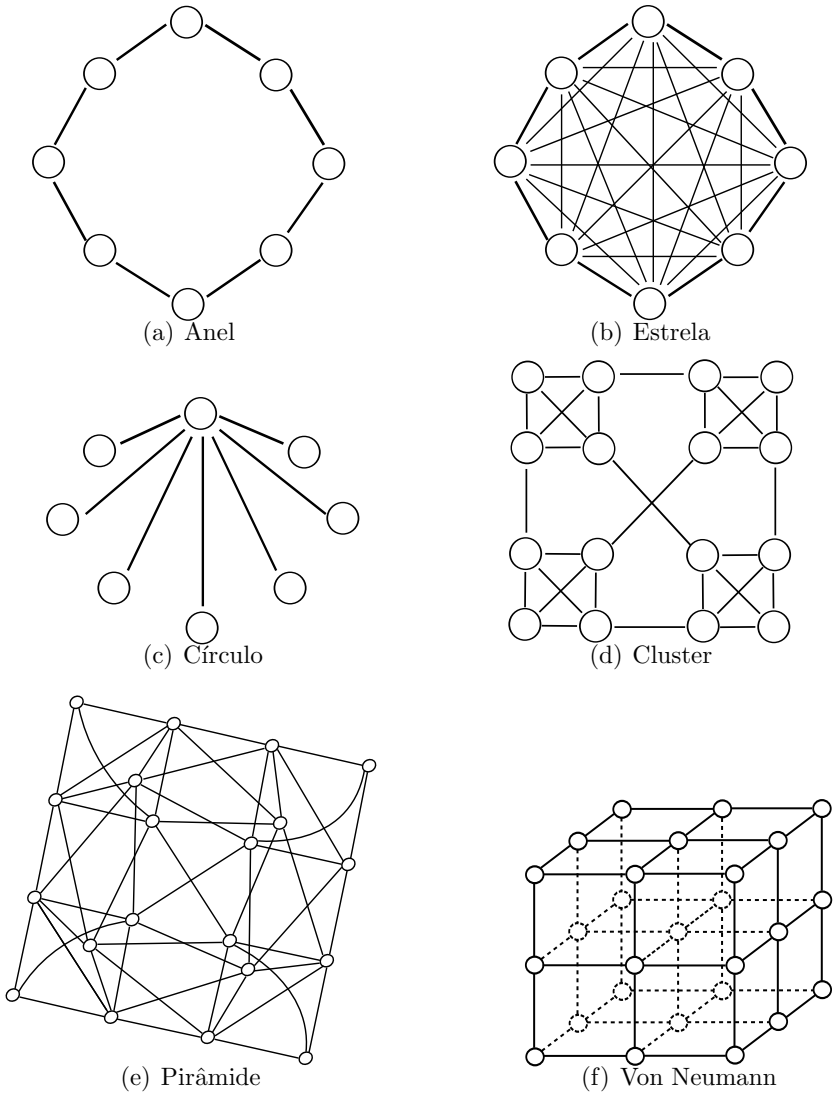


Figura 10: Topologias do PSO

ao que ocorre na natureza.

A topologia (ENGELBRECHT, 2005) (KHOSLA et al., 2006) é a forma como podemos compreender esta interação de cada partícula com o grupo. Em termos práticos, a topologia no PSO é definida pela maneira como é feita a atualização da posição, ou seja, pelo cálculo da velocidade, pois é na velocidade que o conhecimento do melhor caminho é transmitido entre as partículas.

A Figura 2.10(a) mostra uma topologia em anel (Local Best PSO), na qual cada partícula está conectada a outras duas que são sua vizinhança imediata, enquanto a Figura 2.10(b) mostra a topologia em estrela (Global Best PSO), na qual cada partícula está conectada a todas as partículas do grupo, ou seja, sua vizinhança são todas as partículas existentes.

Além destes dois tipos de topologia, podemos ver na Figura 2.10(c) a ligação em círculo, nela todas as partículas estão conectadas a uma partícula central que comanda o fluxo da informação, as outras são isoladas entre si. Na Figura 2.10(d) é mostrada a ligação em clusters, na qual cada partícula é vizinha de todas as outras de seu cluster, e algumas propiciam ligação aos outros clusters. Na Figura 2.10(e) observa-se a forma chamada de pirâmide, que é uma estrutura de interligação tri-dimensional, que forma vários triângulos. Por último, na Figura 2.10(f) tem-se a estrutura Von Neumann, na qual as partículas estão conectadas em uma estrutura de grade (ENGELBRECHT, 2005).

2.5 Considerações Finais do Capítulo

Neste capítulo foi apresentada a otimização baseada em enxame de partículas (PSO), que é utilizada neste trabalho, e é um dos algoritmos da área de inteligência coletiva. Ele foi inspirado no comportamento de bandos de pássaros e cardumes de peixes.

Capítulo 3

TRABALHOS RELACIONADOS

NESTE capítulo serão apresentados alguns trabalhos que serviram de inspiração no desenvolvimento desta dissertação. Assim como, trabalhos que foram utilizados para comparação com os resultados obtidos.

A Seção 3.1 mostra um trabalho sobre sistemas com três variáveis. Na Seção 3.2 é descrito um método de criação de bases de regras *fuzzy* utilizando AG e heurística de pré-seleção de regras candidatas. A Seção 3.3 mostra um trabalho em que o objetivo é construir sistemas *fuzzy* do tipo *Takagi-Sugeno* utilizando AG.

3.1 Sistemas Fuzzy Bi-Dimensionais

Um trabalho de grande interesse para esta dissertação é o (RIVAS et al., 2003), pois nele os autores trazem estudo de quatro diferentes sistemas de três variáveis para os quais devem ser encontrados automaticamente sistemas *fuzzy*, utilizando algoritmos evolucionários. São sistemas com duas variáveis de entrada e uma de saída, que são adequados para teste, pois permitem a visualização gráfica do resultado. Os autores de (RIVAS et al., 2003) trazem uma proposta similar a deste trabalho, que é a geração automática de sistemas fuzzy visando encontrar simultaneamente o conjunto de regras e as funções de pertinência que melhor se adequem.

Para desenvolver o algoritmo evolucionário os autores utilizaram uma biblioteca de objetos evolucionários (OE), aproveitando-se da orientação a objetos.

3.1.1 Representação matricial

O sistema fuzzy de (RIVAS et al., 2003) foi implementado na forma de uma matriz de duas dimensões, armazenando duas informações: (a) os centros das funções de pertinência das variáveis de entrada X e Y, e (b) os valores da variável de saída Z. Desta forma, a matriz bidimensional armazena, tanto os precedentes (ou antecedentes), quanto os consequentes das

regras do sistema fuzzy. A Equação 24 mostra como é feita a representação descrita acima em uma matriz m por n.

$$M(m+1,n+1) = \begin{vmatrix} - & Y_1 & Y_2 & ... & Y_n \\ X_1 & Z_{1,1} & Z_{1,2} & ... & Z_{1,n} \\ X_2 & Z_{2,1} & Z_{2,2} & ... & Z_{2,n} \\ ... & ... & ... & ... & ... \\ X_m & Z_{m,1} & Z_{m,2} & ... & Z_{m,n} \end{vmatrix} \tag{24}$$

A matriz assim definida possui $m+1$ linhas e $n+1$ colunas, sendo o número mínimo de colunas e linhas igual a três, ou seja, $m_{min} = 2$ e $n_{min} = 2$, enquanto os valores de máximo destas dimensões são alguns dos parâmetros que o algoritmo evolucionário deve encontrar. Nesta matriz, m corresponde ao número de funções de pertinência relacionadas à variável de entrada X, enquanto n é o número de funções de pertinência relacionadas à variável de entrada Y. Em uma geração do Algoritmo Evolucionário a população é formada por sistemas *fuzzy* com diferentes números de linhas e colunas.

Pode-se então dividir a matriz M da seguinte forma:

- $M[1,0]$ a $M[m,0]$ (primeira coluna). Armazena os centróides das funções de pertinência da variável de entrada X. A variável X possui valores na faixa $[x_{min}, x_{max}]$, desta forma as condições da Equação 25 devem ser sempre verdadeiras.

$$x_{min} = M[1,0] < M[2,0] < ... < M[m,0] = x_{max} \quad \text{e} \quad m \geq 2 \tag{25}$$

- $M[0,1]$ a $M[0,n]$ (primeira linha). Similar à primeira coluna, mas é relativo a variável Y. As condições da Equação 26 devem ser sempre verdadeiras.

$$y_{min} = M[0,1] < M[0,2] < ... < M[0,n] = y_{max} \quad \text{e} \quad n \geq 2 \tag{26}$$

- $M[1,1]$ a $M[m,n]$ (matriz de consequentes). Todas as células desta submatriz armazenam um valor da variável de saída Z, onde as condições da Equação 27 são observadas.

$$z_{min} \leq M[i,j] \leq z_{max} \quad \forall i, 1 \leq i \leq m \quad \text{e} \quad \forall j, 1 \leq j \leq n \tag{27}$$

- A célula $M[0,0]$ não representa nada, desta forma ela não é utilizada.

O sistema *fuzzy* aqui utilizado é do tipo *Mamdani* e, baseando-se no modo de implementação citado acima, as regras para este sistema são definidas da seguinte forma: "SE X é $M[i,0]$ E Y é $M[0,j]$ ENTÃO Z é $M[i,j]$".

Os valores de x_{max}, x_{min}, y_{max}, y_{min}, z_{max} e z_{min} são parâmetros que devem ser fornecidos para o algoritmo.

3.1.2 Operadores genéticos

No trabalho citado, os autores utilizaram operadores de mutação e cruzamento (recombinação) desenvolvidos especialmente para o problema, pelo fato da representação dos cromossomos não ser da forma de bitstring.

3.1.2.1 Recombinação

Este operador combina valores de uma matriz em outra. Como as matrizes não possuem necessariamente as mesmas dimensões, a recombinação não é feita de forma trivial. Assim, a operação segue os seguintes passos:

1. Escolher aleatoriamente o indivíduo a ser modificado (chamado de receptor, M_r), e um outro que fornecerá os genes a serem recombinados (chamado doador, M_d). Apenas M_r será modificado.

2. Escolher um bloco aleatório de células da matriz de consequentes de M_r. Para especificar este bloco seleciona-se duas células, que representam, respectivamente, o canto superior esquerdo e o canto inferior direito do bloco. As duas células serão chamadas de $M_r[r_1, c_1]$ e $M_r[r_2, c_2]$. Os valores das células deste bloco serão modificados pelos valores vindos de M_d.

3. Para cada célula $M_r[r_i, c_j]$, onde r_i vai de r_1 a r_2, e c_j vai de c_1 a c_2, o operador executa os passos seguintes.

4. Da primeira coluna do doador, M_d, seleciona-se a célula cujo valor é mais próximo de $M_r[r_i, 0]$. Projeta-se esta célula como $M_d[r_d, 0]$.

5. Da primeira linha do doador, M_d, seleciona-se a célula cujo valor é mais próximo de $M_r[0, c_j]$. Projeta-se esta célula como $M_d[0, c_d]$.

6. Finalmente, alterar o valor de $M_r[r_i, c_j]$ pelo valor armazenado em $M_d[r_d, c_d]$. Como pode ser visto, o doador não é alterado.

A Figura 11 mostra um exemplo do operador de recombinação, na qual o sistema *fuzzy* receptor e o sistema *fuzzy* doador geram o sistema *fuzzy* resultado. Nesta Figura, o sistema fuzzy gerado possui em seu consequente a contribuição, tanto do doador, quanto do receptor.

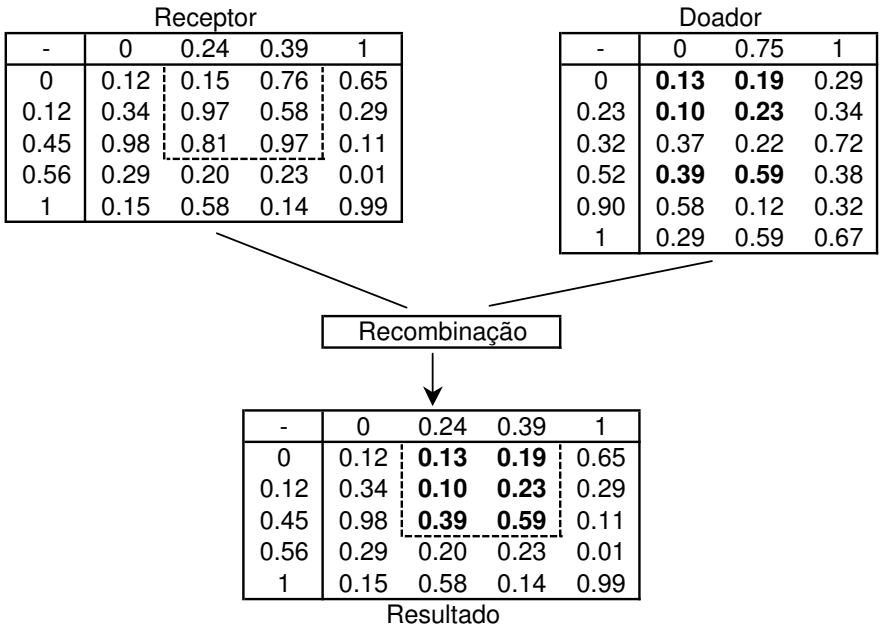


Figura 11: Exemplo da operação de recombinação

3.1.2.2 Mutação de adição

Esta operação consiste em adicionar função de pertinência à variável X, incluindo uma linha na matriz, ou adicionar função à variável Y, incluindo uma coluna. Para isso segue-se os seguintes passos:

1. Inserir uma linha em branco em uma posição aleatória, diferente da primeira e da última.
2. Preencher as células da nova linha com valores aleatórios gerados por uma função Gaussiana localizada no ponto central entre a que corresponde à linha anterior e a que corresponde a linha posterior.

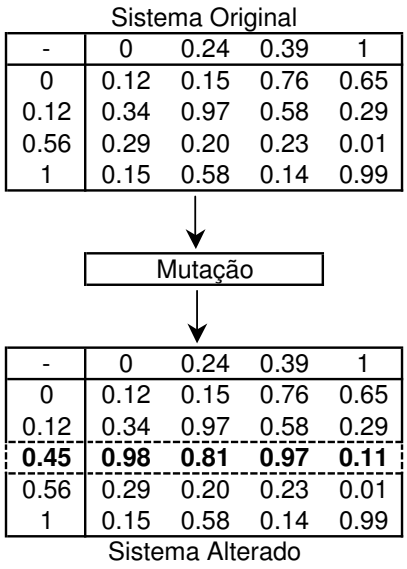
A Figura 12 mostra um exemplo de alteração por mutação de adição, na qual uma nova linha é criada na matriz. Isto significa que uma uma nova função de pertinência foi adicionada à variável X do sistema *fuzzy*.

O algoritmo que adiciona colunas trabalha da mesma forma do operador apresentado acima, mas ao invés de incluir uma linha, ele inclui uma coluna.

3.1.2.3 Mutação de remoção

A operação de mutação de remoção consiste em remover uma função de pertinência da variável X, excluindo uma linha da matriz, ou remover uma função da variável Y, excluindo uma coluna.

Este operador funciona selecionando uma linha ou coluna a ser removida e então a remove. Ele não permite que sejam removidas a última e a primeira linha ou coluna, e só



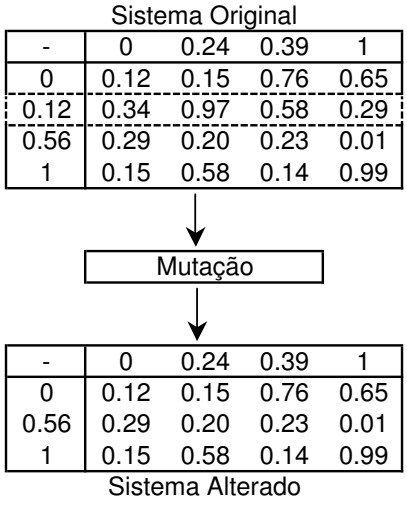
-	0	0.24	0.39	1
0	0.12	0.15	0.76	0.65
0.12	0.34	0.97	0.58	0.29
0.56	0.29	0.20	0.23	0.01
1	0.15	0.58	0.14	0.99

-	0	0.24	0.39	1
0	0.12	0.15	0.76	0.65
0.12	0.34	0.97	0.58	0.29
0.45	**0.98**	**0.81**	**0.97**	**0.11**
0.56	0.29	0.20	0.23	0.01
1	0.15	0.58	0.14	0.99

Figura 12: Exemplo da operação de mutação de adição

remove uma linha, ou coluna, se houver mais de três linhas ou colunas na matriz. Assim há a garantia de que o sistema *fuzzy* gerado seja sempre válido.

A Figura 13 mostra um exemplo de alteração por mutação de remoção, na qual uma linha é removida da matriz. Isto significa que uma função de pertinência foi removida da variável X do sistema *fuzzy*.



-	0	0.24	0.39	1
0	0.12	0.15	0.76	0.65
0.12	0.34	0.97	0.58	0.29
0.56	0.29	0.20	0.23	0.01
1	0.15	0.58	0.14	0.99

-	0	0.24	0.39	1
0	0.12	0.15	0.76	0.65
0.56	0.29	0.20	0.23	0.01
1	0.15	0.58	0.14	0.99

Figura 13: Exemplo da operação de mutação de remoção

3.1.2.4 Mutação de centróide

Esta operação consiste em modificar um dos valores de centróide da variável X ou Y, armazenados na primeira linha ou coluna, respectivamente, da matriz. Para isso são executados os seguintes passos:

1. Selecionar aleatoriamente uma célula da primeira linha ou coluna, mas não pode ser nem a primeira, nem a última célula.

2. Colocar um valor aleatório na célula selecionada, que seja maior do que o da coluna ou linha anterior e menor do que o da coluna ou linha seguinte.

3.1.2.5 Mutação de consequente

Este operador funciona da seguinte forma:

1. Seleciona-se aleatoriamente uma célula da matriz de consequentes.

2. Introduz-se nesta célula um novo valor aleatório determinado por uma função Gaussiana centralizada no valor existente, e variando entre z_{min} e z_{max}.

3.1.3 Função de aptidão

Para calcular a aptidão de cada indivíduo, um conjunto (chamado de conjunto de treino) de pares de entrada e saída da função conhecida é apresentado.

A aptidão definida para os sistemas *fuzzy* é o inverso do erro médio quadrático normalizado da saída correta conhecida para a saída produzida pelo sistema encontrado, conforme Equação 28.

$$F_j = \frac{N}{\sum_{i=1}^{N}(\frac{z_i - z_{ij}}{z_{max} - z_{min}})^2} \tag{28}$$

Onde F_j é a aptidão do indivíduo de número j, N é o número de exemplos no conjunto de treinamento, z_i é a saída correta, e z_{ij} é a saída obtida pelo sistema *fuzzy*. Um caso especial é quando $z_i = z_{ij}$, $\forall i$, $1 \leq i \leq m$, porque F_j seria igual a $N/0$; mas neste caso, Fj é definido com um valor muito alto.

3.1.4 O algoritmo

O algoritmo evolucionário apresenta seleção elitista, população fixa e os operadores descritos na Seção 3.1.2. Obedece, na geração inicial, a um limite superior do número de linhas e colunas. A probabilidade de seleção para reprodução de um indivíduo do conjunto elite depende da aptidão.

Todos os operadores possuem uma probabilidade de aplicação associada que não muda durante a execução do algoritmo. Todos os novos indivíduos são criados aplicando-se apenas um dos operadores à cópia de seus pais, ou recombinação, ou uma das mutações. O algoritmo pára ao atingir o número máximo de gerações definido.

3.2 Geração de Regras Utilizando AG com Seleção Auto-Adaptativa

O trabalho (CINTRA; CAMARGO, 2007b) propõe o uso de um algoritmo auto-adaptativo para o cálculo da aptidão no AG como uma melhoria do trabalho (CINTRA; CAMARGO, 2007a) que descreve um método sobre a criação de bases de regras *fuzzy* usando algoritmo genético (AG) em associação a uma heurística de pré-seleção de regras candidatas.

3.2.1 Geração genética de regras *fuzzy*

Na geração automática de bases de regras fuzzy (BRF) utilizando AG, o espaço de busca é formado por combinações de um certo número de regras a partir de todas as regras possíveis, considerando as variáveis do problema e os conjuntos *fuzzy* já definidos.

Conforme o número de variáveis aumenta, o conjunto de regras possíveis aumenta exponencialmente, interferindo no resultado do processo de aprendizado, e em algumas situações tornando-o impossível.

O algoritmo consegue reduzir a dimensão do espaço de busca e simplificar a codificação da base de regras no cromossomo. A heurística de pré-seleção é associada ao grau de cobertura das regras (em inglês degree of coverage, DoC), pois este permite o descarte de um grande número de regras.

Seja $E = e_1, e_2, ..., e_M$ um conjunto de exemplos do problema, o DoC de uma regra R com relação a $E(DoC_R)$ é definido na Equação 29.

$$DoC_R = \sum_{i=1}^{M} (DoC_{(R,e_i)}) \tag{29}$$

Uma vez que as partições *fuzzy* dos domínios dos atributos são definidos, o valor de DoC é calculado para todas as regras possíveis, as quais são então ordenadas em ordem crescente, pelo valor de DoC.

O método proposto por Wang e Mendell (WM) (WANG; MENDEL, 1992)(WANG, 2003) foi utilizado como uma referência para a definição de diversos parâmetros no método proposto. Os dois critérios utilizados para pré-seleção de regras candidatas foram:

1. Considerar o conjunto ordenado de regras do topo, acima do ponto onde todas as regras apresentadas na BRF geradas pelo método WM são incluídas;

2. Considerar o conjunto ordenado de regras contendo todas as regras com DoC não nulo.

3.2.1.1 Codificação do cromossomo

Cada regra é identificada unicamente por sua posição nas regras ordenadas, o que induz a uma codificação binária.

As regras que estão na base de regras são representadas no cromossomo com um dígito um no gene correspondente, significando que a regra está ativa. Regras que estão inativas, ou que não pertençam à BRF representada pelo cromossomo, são codificadas com zero no gene correspondente.

Para a população inicial os cromossomos foram criados com uma porcentagem de regras ativas baseada no número de regras geradas pelo método WM. Os cromossomos foram gerados aleatoriamente e regras conflitantes foram eliminadas descartando-se os cromossomos com este conflito e gerando novos.

3.2.1.2 Cálculo da aptidão

No trabalho (CINTRA; CAMARGO, 2007b) os autores refinaram o método utilizado em (CINTRA; CAMARGO, 2007a) com o objetivo específico de melhorar o passo da avaliação feita pelo AG de forma a reduzir a quantidade de regras na BRF durante o processo de busca. O valor de aptidão é calculado utilizando a taxa de classificação correta (TCC) e o número de regras na BRF representado por cada cromossomo.

O processo de avaliação é mostrado no Algoritmo 2. O algoritmo atualiza o valor de referência de TCC e do número de regras (NR), sendo que NR é usado como penalização.

Os valores iniciais de referência do número de regras (*Best*_NR) e da taxa de classificação correta (*Best*_TCC) são definidos como sendo os valores gerados pelo método WM. Os valores são atualizados em cada geração e então utilizados no cálculo de aptidão.

O processo geral de avaliação aplicado em cada iteração do AG pode ser descrito como:

1. Calcular a TCC para cada cromossomo e definir o valor inicial de aptidão;
2. Atualizar os valores de referência *Best*_NR e *Best*_TCC utilizando NR e TCC encontrados no passo anterior;
3. Calcular o valor de aptidão final de cada cromossomo, aplicando o mecanismo de penalização.

As taxas de penalização são mostradas na Tabela 1. Estas taxas foram definidas empiricamente. O Algoritmo 2 descreve com detalhes o processo de avaliação.

Tabela 1: Taxa de penalização pelo número de regras

Número de Regras	Valor de Aptidão
$\leq$ NRWM	TCC
$\leq$ NRWM $\times$ 1.5	TCC/1.25
$\leq$ NRWM $\times$ 2	TCC/1.5
$\leq$ NRWM $\times$ 3	TCC/2
$>$ NRWM $\times$ 3	TCC/3

Algoritmo 2 Algoritmo para definição do número de regras ótimo da BRF gerada

C = cromossomo
NR = número de regras
TCCWM = TCC da BRF gerada pelo WM
NRWM = NR na BRF gerada pelo WM
$Best_NR$ = NR de referência
$Best_TCC$ = TCC de referência
$Best_NR = NRWM$;
$Best_TCC = TCCWM$;
Para ($a = 0$ **até** total_de_cromossomos) **Faça**
  Aptidão de $C_a = TCCdeC_a$;
Fim Para
Para ($a = 0$ **até** total_de_cromossomos) **Faça**
  Se (NR de $C_a \leq Best_NR$) **Então**
    Se (TCC de $C_a \leq Best_TCC$) **Então**
      $Best_NR = NRdeC_a$;
      $Best_TCC = TCCdeC_a$;
    Fim Se
  Fim Se
Fim Para
Para ($a = 0$ **até** total_de_cromossomos) **Faça**
  Se (NR de $C_a > Best_NR$) **Então**
    Se (NR de $C_a \leq Best_NR * 1.5$) **Então**
      Aptidão de C_a = Aptidão de $C_a/1.25$;
    Senão
      Se (NR de $C_a \leq Best_NR * 2$) **Então**
        Aptidão de C_a = Aptidão de $C_a/1.5$;
      Senão
        Se (NR de $C_a \leq Best_NR * 3$) **Então**
          Aptidão de C_a = Aptidão de $C_a/2$;
        Senão
          Aptidão de C_a = Aptidão de $C_a/3$;
        Fim Se
      Fim Se
    Fim Se
  Fim Se
Fim Para

3.3 Modelo *Fuzzy Takagi-Sugeno*

O trabalho (KIM; KIM; LEE, 2006) baseia-se no modelo de sistemas *fuzzy Takagi-Sugeno* (TS) (TAKAGI; SUGENO, 1985). A diferença entre os sistemas do tipo *Mandani* e *Takagi-Sugeno* pode ser encontrada na Seção 1.1.4.

Os autores definem como objetivo a otimização da estrutura e dos parâmetros do modelo TS na modelagem *fuzzy* de sistemas não-lineares.

Para desenvolver um modelo TS, geralmente o primeiro passo é particionar o espaço de entrada. Três aproximações de partição *fuzzy* costumam ser utilizadas: partição de grade, partição de árvore e partição de dispersão (HO et al., 2004). No trabalho citado os autores adotadaram a partição de grade. Neste caso, quando o número de variáveis de entrada (dimensão de entrada) é N_v, particionar o i-ésimo espaço de entrada em m_i conjuntos *fuzzy* equivale a definir $m_i(i = 1, 2, ..., N_v)$ funções de pertinência para a i-ésima entrada.

A parte dos antecedentes de cada regra *fuzzy* é obtida pela combinação de cada função de pertinência de cada variável de entrada, como mostra a Figura 14.

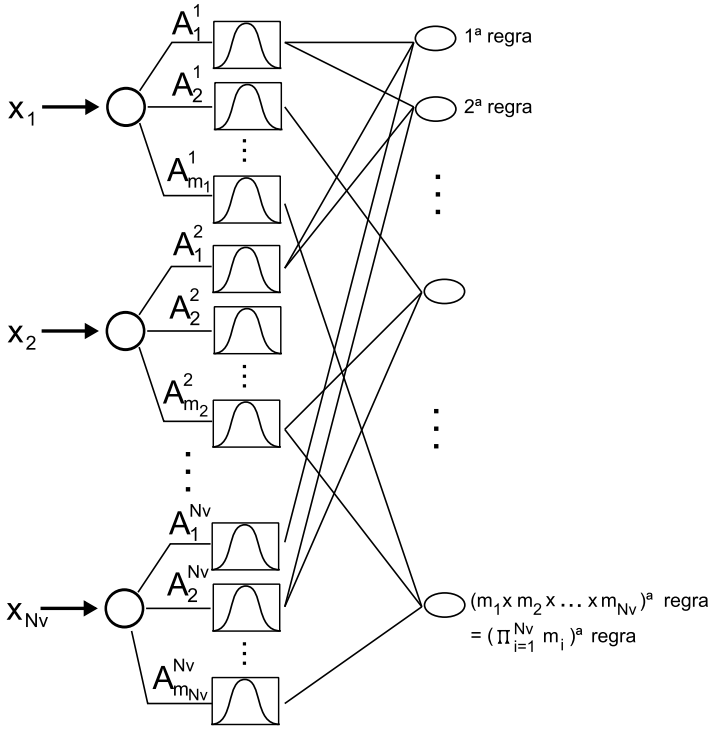


Figura 14: Estrutura de regras com partição de grid

Consequentemente, se o número de funções para cada variável de entrada for determinado, então o número de regras *fuzzy* é determinado pela Equação 30. Última regra na Figura

14.

$$N_R = \prod_{i=1}^{N^v} m_i \tag{30}$$

Quando a j-ésima função de pertinência da i-ésima entrada é definida por $A_j^i (j = 1, 2, ..., m_i)$, a função de pertinência é caracterizada por uma Gaussiana $\mu_{A_j^i}(x_i)$, como mostra a Equação 31, onde c_j^i e σ_j^i representam, respectivamente, o valor do centro e o valor da largura da j-ésima função de pertinência da i-ésima entrada.

$$\mu_{A_j^i}(x_i) = exp\left(\frac{-(x_i - c_j^i)^2}{(\sigma_j^i)^2}\right) \tag{31}$$

Tipicamente, as regras fuzzy do modelo TS são descritas da forma indicada na Equação 32.

$$\begin{aligned} R_k: \ & \text{SE } x_1 \text{ é } A_j^i \text{ E ... E } x_{N_v} \text{ é } A_j^{N_v} \\ & \text{ENTÃO } y_k = w_k^0 + w_k^1 x_1 + ... + w_k^{N_v} x_{N_v} \\ & \text{para } k = 1, 2, ..., N_R \end{aligned} \tag{32}$$

Neste caso, y_k é a saída da k-ésima regra, e w_k^i são os valores reais dos parâmetros dos consequentes da k-ésima regra para a i-ésima entrada. w_k^0 representa um termo constante. Usualmente, w_k^i pode ser obtido pelo método de otimização linear do mínimo quadrado (SETNES; ROUBOS, 2000) (KIM; KIM; LEE, 2004).

Quando utiliza-se partição de grade, o índice de regra k pode ser calculado conforme a Equação 33.

$$k = \sum_{i=1}^{N_v - 1} \left((j^i - 1). \prod_{h=i+1}^{N_v} m_h \right) + j^{N_v} \tag{33}$$

Assim, $j^i (j = 1, ..., m_i)$ representa o índice corrente para a j-ésima função de pertinência da i-ésima entrada, e m_h é o número de funções de pertinência para a h-ésima entrada.

Fornecido um dado de entrada, cada regra *fuzzy* produz um grau para o dado. Este grau da k-ésima regra R_k é denotado por τ_k e é calculado conforme a Equação 34.

$$\tau_k = A_l^1(x_1) \times A_m^2(x_2) \times ... \times A_n^{N_v}(x_{N_v}) = \prod_{i=1}^{N_v} \mu_{A_j^i}(x_i) \tag{34}$$

Neste caso, os índices l, m e n são utilizados para representar de forma mais precisa os índices das funções de pertinência correspondentes para a k-ésima regra *fuzzy*. Na Equação 34 o operador $\times$ representa a operação "*fuzzy* E" e o produto algébrico é utilizado. A saída $\hat{y}$ do modelo *fuzzy* é computada por defuzificação baseada na média de pesos como mostra a Equação 35.

$$\hat{y} = \frac{\sum_{k=1}^{N_R} \tau_k y_k}{\sum_{k=1}^{N_R} \tau_k} = \frac{\sum_{k=1}^{N_R} \tau_k (w_k^0 \times w_k^1 x_1 \times ... \times w_k^{N_v} x_{N_v})}{\sum_{k=1}^{N_R} \tau_k} \tag{35}$$

Tendo sido fornecidos N_T pares de dados de entrada e saída $(\mathbf{x}_k, y_k^d)$ com $x_k \in R^{N_v}$, $y_k^d \in R$, e $k = 1, 2, ..., N_T$, o problema consiste em encontrar uma estrutura otimizada de forma que o erro entre a saída do modelo *fuzzy* $\hat{y}_k$ e a saída desejada y_k^d seja minimizado.

Desde que o desenvolvimento de um modelo *fuzzy* com aproximação de partição de grade sofre um grave problema de "explosão de regras", foi proposto um novo esquema de codificação para minimizá-lo. O desenvolvimento de AG requer três fatores: um esquema de codificação, um método de avaliação, e operações evolucionárias apropriadas.

3.3.1 Codificação do modelo *fuzzy* em um cromossomo

A codificação proposta utiliza um cromossomo que consiste em três partes. A primeira parte lida com a seleção de regra enquanto as outras partes lidam com a otimização da quantidade e parâmetros das funções de pertinência.

3.3.1.1 Codificação da estrutura de regras

Neste trabalho um novo esquema de codificação é proposto, no qual o tamanho do cromossomo não aumenta exponencialmente com o acréscimo do número de variáveis de entrada.

A Figura 14 mostra que a combinação das funções de pertinência de cada entrada representa uma possível regra *fuzzy*. Isso pode ser traduzido de forma diferente se a estrutura de regra for redefinida, tal qual a Figura 15. Nesta figura cada função de pertinência de uma entrada específica possui uma conexão com todas as funções da entrada vizinha. As funções de pertinência da última entrada estão ligadas às funções da primeira entrada desde que as duas entradas são vizinhas.

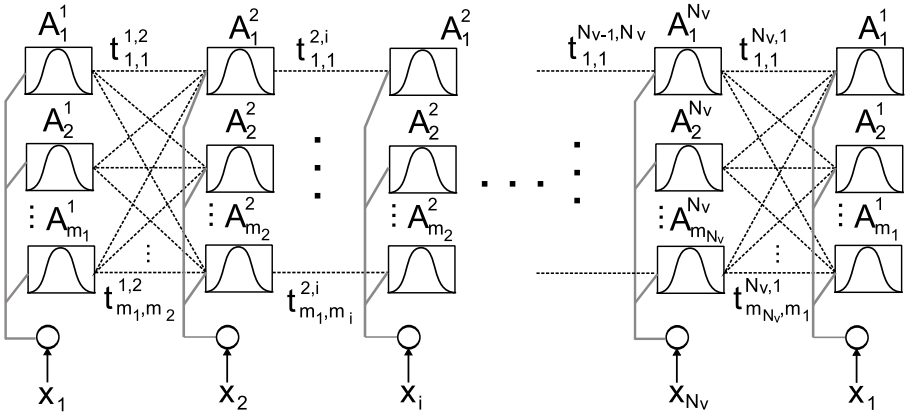


Figura 15: Estrutura de regras de ligações possíveis com partição de grid

Através da Figura 15 pode-se facilmente inferir que um "caminho fechado" é definido como um caminho no qual a posição inicial e a posição final são a mesma função de pertinência

da mesma entrada. Como exemplo, um caminho que consiste de $A_1^1, A_1^2, ..., A_1^{N_v}, A_1^1$, ou seja, conectando todas as primeiras funções de pertinência de cada entrada, na Figura 15 é um caminho fechado e corresponde à primeira regra na Figura 14.

Naturalmente, existem $\prod_{i=1}^{N_v} m_i$ caminhos fechados o que corresponde ao número máximo de regras. Dessa forma, a estrutura de regras pode ser especificada selecionando um conjunto de caminhos fechados do total de conexões. Para a seleção de regras, pesos de conexões $t_{m,n}^{i,j}$ são definidos. $t_{m,n}^{i,j}$ representa um valor de peso de conexão entre a m-ésima função de pertinência da i-ésima entrada e a n-ésima função da j-ésima entrada.

O intervalo de valores de pesos de conexão é $[0, 1]$. Quanto maior o valor do peso de conexão, maior a possibilidade de que duas funções de pertinência conectadas com este peso sejam utilizadas para formar uma regra. Por este conceito, o conjunto de todos os pesos de conexão é definido pela cadeia de validade da possibilidade de conexão de regras, em inglês *"Chained Possibilistic Rule Validity String"* (CPRVS). A CPRVS pode ser descrita conforme a Equação 36.

$$CPRVS = [t_{1,1}^{1,2}, t_{1,2}^{1,2}, ..., t_{1,m_2}^{1,2}, ..., t_{m_1,m_2}^{1,2}, ..., t_{1,1}^{N_v-1,N_v}, t_{m_{N_v-1},m_{N_v}}^{N_v-1,N_v}, t_{m_{N_v},m_1}^{N_v,1}] \tag{36}$$

O comprimento da CPRVS, L_{CPRVS}, é calculado da forma definida na Equação 37.

$$L_{CPRVS} = m_{N_v}.m_1 + \sum_{i=1}^{N_v-1} m_i.m - i + 1 \tag{37}$$

Utilizando a CPRVS, a validade de uma regra é determinada multiplicando o valor de todos os pesos de conexão que formam a regra.

Quando o número de entradas N_v é igual a dois, a conexão partindo da última entrada para a primeira não é necessária, pois a conexão entre as duas entradas já existe. Neste caso, ainda que não esteja explicitamente fechado, o caminho é considerado fechado.

3.3.1.2 Codificação dos parâmetros de função de pertinência

Desde que os parâmetros dos consequentes na Equação 32 podem ser obtidos pelo método de otimização linear do mínimo quadrado, não é necessária a inclusão explícita dos parâmetros do consequente no cromossomo. Deste modo, somente é necessária a codificação dos parâmetros das funções de pertinência dos antecedentes. Por exemplo, dois valores reais, c_j^i e σ_j^i na Equação 31, são necessários para representar uma função de pertinência Gaussiana no trabalho em questão. Esta cadeia de valores reais é definida como uma cadeia representativa de funções de pertinência, em inglês *"membership function representative string"* (MFRS). O comprimento

de uma MFRS, L_{MFRS}, é definido na Equação 38, onde N_{pmf} denota o número de parâmetros reais para representar um único antecedente da função de pertinência.

$$L_{MFRS} = N_{pmf} \times \sum_{i=1}^{N_v} mi \tag{38}$$

3.3.1.3 Codificação de seleção de função de pertinência

Desde que o número de funções de pertinência da i-ésima entrada, m_i, é definida *a priori*, outro vetor é necessário para representar a seleção de função de pertinência. Com este propósito, um vetor de valores binários de comprimento L_{MFVS} definido na Equação 39, é utilizado. Este vetor foi chamado de cadeia de validade de função de pertinência, em inglês *"membership function validity string"* (MFVS), e representa se cada função é usada ou não. Se em uma determinada posição de MFVS o valor for zero, então a função correspondente não é válida, caso contrário o valor é um. Em outras palavras MFVS representa se uma determinada função é utilizada ou não na estrutura de regras atual de um indivíduo.

$$L_{MFVS} = \sum_{i=1}^{N_v} mi \tag{39}$$

3.3.1.4 Codificação do cromossomo

Com os três componentes definidos nas seções anteriores pode-se montar o cromossomo. O comprimento do cromossomo que representa o modelo *fuzzy* é definido como $L_{MFRS}+L_{MFVS}+L_{CPRVS}$. Isto é, o p-ésimo indivíduo s_p com $p = 1, 2, ..., N_P$ pode ser representado por um vetor como mostra a Equação 40.

$$s_p(g) = [v_1^1 v_2^1 \; ... \; v_{m_1}^1 \; ... \; v_1^{N_v} \; ... \; v_{m_{N_v}}^{N_v} \qquad u_1^1 u_2^1 \; ... \; u_{m_{N_v}}^{N_v} \qquad t_1 t_2 \; ... \; t_{L_{CPRVS}}] \tag{40}$$

Na equação 40, $v_j^i = [c_j^i, \sigma_j^i]$ representa os parâmetros da j-ésima função de pertinência da i-ésima variável de entrada, u_j^i representa a validade da j-ésima função de pertinência da i-ésima variável de entrada, e $t_l (l = 1, ..., L_2)$ representa os pesos de conexão. g é o número de geração.

Deve-se notar que o número de regras atual, N_R', é diferente do número total de regras N_R, de acordo com o conteúdo de CPRVS e MFVS. No caso extremo, é possível que todas as regras sejam válidas de acordo com o conteúdo de CPRVS e MFVS. Isto pode ser evitado utilizando-se de um limite superior N_R^{max}. Um indivíduo em que o número de regras válidas ultrapassar este limite é removido. Quando o número de regras é zero o indivíduo também é removido. No algoritmo proposto, estes indivíduos são substituídos por um membro da elite da geração anterior.

3.3.2 Função de avaliação

Após a codificação do esquema (cromossomo) o próximo passo é providenciar um método de avaliação. O desenvolvimento da função de avaliação é importante, pois ela que direciona o processo de busca e posiciona uma solução de acordo com seu potencial em relação ao problema. A função de avaliação utilizada é mostrada na Equação 41.

$$\begin{aligned} F = \; & -K1.log(MSE) \\ & +K2.100.[0.5.(1-COMP_r)+0.5.(1-COMP_f)] \\ & +K3.100.[0.5.(1-P_D)+0.5.(1-P_C)] \end{aligned} \tag{41}$$

O primeiro termo é a média quadrática dos erros (em inglês mean square error, MSE) mede a acurácia do sistema *fuzzy*, os outros termos medem a interpretabilidade. $COMP_r$ e $COMP_f$ medem, respectivamente, a quantidade de regras e de funções de pertinência. Enquanto, P_D e P_C, medem a distinguibilidade e a completude. O conceito de distinguibilidade e completude são descritos na Seção 4.2.4.

3.3.3 Operadores evolucionários

O próximo passo após definição do esquema e a determinação da função de avaliação é providenciar os operadores evolucionários. Em geral existem três operadores principais em AEs: reprodução, cruzamento e mutação. Como a codificação do cromossomo neste trabalho é única, os operadores devem ser cuidadosamente elaborados.

3.3.3.1 Operador de reprodução

Este operador é baseado em método de classificação. Todos os indivíduos da população são ordenados de acordo com seu valor de avaliação. Depois da ordenação, os 30% melhores da população são utilizados para gerar 50% da próxima população selecionando os indivíduos aleatoriamente. Os 70% restantes da população são utilizados para gerar 50% da próxima geração. Foi utilizado elitismo para preservar o melhor indivíduo. Além disso, 5% dos indivíduos da nova população são selecionados aleatoriamente para serem substituídos por cópia de indivíduos da elite.

3.3.3.2 Operador de cruzamento

Três operações de cruzamento foram aplicadas para alterar tanto estrutura, quanto valores de parâmetro de cada indivíduo: duas alteram a MFRS e a outra a CPRVS.

Cruzamento CPRVS: combina a estrutura de dois indivíduos pais alterando a configuração da CPRVS, que resulta implicitamente na alteração do atual número de regras e de

funções. Quando dois pais $s_v(g)$ e $s_w(g)$ são selecionados para cruzamento com probabilidade P_{sx}, é selecionado aleatoriamente o ponto m para cruzamento, que se dá da forma mostrada na Equação 42.

$$\begin{array}{rl} s_v(g) = & [v_1...v_L...p_1...p_m\ p_{m+1}...p_{LCPRVS}] \\ s_w(g) = & [w_1...w_L...q_1...q_m\ q_{m+1}...q_{LCPRVS}] \\ & \Downarrow \\ s_v(g+1) = & [v_1...v_L...p_1...p_m\ q_{m+1}...q_{LCPRVS}] \\ s_w(g+1) = & [w_1...w_L...q_1...q_m\ p_{m+1}...p_{LCPRVS}] \end{array} \tag{42}$$

Cruzamento MFRS: o primeiro cruzamento para MFRS altera a configuração da MFRS trocando um conjunto de campos entre os pais. Após seleção dos cromossomos, são selecionados os pontos aleatórios k de forma que um intervalo dentro do comprimento de MFRS seja selecionado para troca, como mostra a Equação 43.

$$\begin{array}{rl} s_v(g) = & [v_1...v_k\ v_{k+1}...v_L...p_1...p_{LCPRVS}] \\ s_w(g) = & [w_1...w_k\ w_{k+1}...w_L...q_1...q_{LCPRVS}] \\ & \Downarrow \\ s_v(g+1) = & [v_1...v_k\ w_{k+1}...w_L...p_1...p_{LCPRVS}] \\ s_w(g+1) = & [w_1...w_k\ v_{k+1}...v_L...q_1...q_{LCPRVS}] \end{array} \tag{43}$$

Neste caso, apenas funções ativas são incluídas no cruzamento.

O segundo cruzamento para MFRS troca os valores de parâmetro da MFRS dos indivíduos. Como estes são valores reais pode ser aplicado o cruzamento aritmético. Este cruzamento produz os filhos usando combinação linear dos pais com probabilidade P_{px}, como mostra a Equação 44.

$$\begin{array}{rl} s_v(g+1) = & \alpha.s_v(g) + (1-\alpha).s_w(g) \\ s_w(g+1) = & \alpha.s_w(g) + (1-\alpha).s_v(g) \end{array} \tag{44}$$

Na Equação 44 o parâmetro α é gerado aleatoriamente cada vez que o cruzamento é aplicado. s_v e s_w representam os parâmetros em cada indivíduo, como centro e largura.

3.3.3.3 Operador de mutação

Uma vez que MFRS e CPRVS consistem em parâmetros reais, um operador de mutação similar pode ser aplicado aos dois com diferença na probabilidade e na variação de parâmetro. A mutação de CPRVS com probabilidade P_{sm} é mostrada na Equação 45.

$$t_{m,n}^{i,j}(g+1) = t_{m,n}^{i,j}(g) + N\left(0, \delta_s.exp\left(\frac{-t}{t_{chk}}\right)\right) \tag{45}$$

Nesta equação, $N(.,.)$ denota um valor aleatório unidimensional normalmente distribuído com meio zero e variação δ_s. t_{chk} é um parâmetro de projeto para ajustar a taxa de diminuição da variação. O termo exponencial é inserido para diminuir o intervalo do número aleatório gerado conforme aumenta a geração, desta forma a área de busca concentra-se em determinada região conforme as evoluções procedem.

De forma similar cada valor de parâmetro da MFRS é mutada com probabilidade P_{pm} como mostra a Equação 46.

$$\begin{aligned} \theta(g+1) &= \theta(g) + N\left(0, \delta_p.exp\left(\frac{-t}{t_{chk}}\right)\right) \\ \delta_p &= \rho.({}_{max}Entrada_i - {}_{min}\ Entrada_i) \end{aligned} \tag{46}$$

Neste caso, θ é um dos valores de parâmetro: o centro c_j^i ou a largura σ_j^i de uma função de pertinência. ρ é um parâmetro de projeto para ajustar o valor da variação. Quando MFVS é utilizada na forma ativa, a mutação de cada campo consiste em alterar um campo da MFVS de zero para um ou vice-versa. Os parâmetros ${}_{max}Entrada_i$ e ${}_{min}Entrada_i$ são valores de máximo e mínimo dos dados da i-ésima entrada.

3.4 Considerações Finais do Capítulo

Este capítulo apresentou um resumo dos trabalhos (RIVAS et al., 2003), (CINTRA; CAMARGO, 2007b) e (KIM; KIM; LEE, 2006). Os três trabalhos serviram como fonte de conhecimento, sendo que os dois primeiros serviram como base para comparação dos resultados com o algoritmo proposto neste trabalho e o terceiro serviu como base para a função de avaliação utilizada. Em (RIVAS et al., 2003) os autores geram sistemas *fuzzy* para funções tridimensionais, em (CINTRA; CAMARGO, 2007b) são gerados sistemas *fuzzy* sobre repositórios de bases de dados de entrada e saída para alguns problemas de classificação, e em (KIM; KIM; LEE, 2006) os autores utilizam uma função de avaliação que leva em consideração o erro entre a saída do modelo e a saída desejada, o tamanho do conjunto de regras, a quantidade de funções, além da descontinuidade e sobreposição entre as funções de pertinência das variáveis.

Capítulo 4

MODELAGEM PROPOSTA

PARA implementar o modelo, duas questões fundamentais que necessitavam ser respondidas se impuseram. A primeira era saber qual problema o modelo se proporia a tratar, não apenas a idealização, mas seus detalhes, como por exemplo, se seriam alvo de busca apenas o conjunto de regras, ou apenas as funções de pertinência, ou todos os componentes que formam um sistema *fuzzy*. A segunda era qual ferramenta seria utilizada para implementar tal modelo, e o que definiria a escolha entre uma ferramenta ou outra.

Respondendo à primeira questão, o interesse era o de desenvolver sistemas *fuzzy* completos, com todos os seus parâmetros, fornecendo um algoritmo que necessitasse apenas das informações do problema para criar o sistema *fuzzy* que o solucionasse.

A segunda questão, a escolha da ferramenta para utilizar na modelagem, levou em consideração a quantidade de trabalhos já realizados sobre o assunto. As opções mais interessantes eram Algoritmo Genético (AG) e Otimização por Enxame de Partículas (PSO), Capítulo 2. O primeiro possui uma gama de trabalhos já realizados com o propósito de otimização de sistemas *fuzzy*, inclusive trabalhos sobre a modelagem de toda a estrutura do modelo *fuzzy* (RIVAS et al., 2003). Em relação ao segundo também foram encontrados diversos trabalhos já realizados com o propósito de otimização de sistemas *fuzzy*, mas nenhum foi encontrado com o mesmo objetivo deste trabalho, ou seja, desenvolver automaticamente toda a estrutura do modelo *fuzzy*. O trabalho (ESMIN; AOKI; LAMBERT-TORRES, 2002), por exemplo, possui o objetivo de otimizar as funções de pertinência, e o trabalho (MARINKE et al., 2005) utiliza PSO na otimização de regras para um sistema de termo-vácuo. Sendo assim, um dos propósitos deste trabalho passou a ser a verificação da utilização do algoritmo PSO com este objetivo.

Neste capítulo é apresentado em detalhes o modelo do algoritmo criado neste trabalho. A Seção 4.1 apresenta o problema a ser tratado. A Seção 4.2 mostra as definições de implementação do modelo, incluindo a representação do problema, como foi definido o posicionamento

de cada partícula, como é feita a inicialização do algoritmo, cada parte da função de avaliação, e como foram definidas as alterações das partículas.

4.1 O Problema

A modelagem de sistemas *fuzzy* tem demonstrado grande utilidade quando se fala em problemas complexos e de difícil modelagem matemática (TANSCHEIT, 2003).

Mas modelar um sistema *fuzzy* pode não ser uma das tarefas mais fáceis, principalmente quando o problema possui muitas variáveis e termos linguísticos. Um dos maiores problemas é a necessidade de obter o conhecimento dos especialistas no negócio para poder definir o modelo *fuzzy* de forma adequada. Além de muitas vezes ser ineficiente, pode levar um tempo considerável (KIM; KIM; LEE, 2006), pois nem sempre os especialistas sabem explicar porque tomaram uma determinada decisão ao invés de outra, ou ainda a desconfiança pode fazer com que estes especialistas não forneçam todas as informações necessárias à modelagem do problema.

Para minimizar a dependência em relação aos especialistas, alguns métodos vêm sendo empregados na tentativa de desenvolver automaticamente os componentes do sistema fuzzy. No caso de definição das funções de pertinência, algoritmos baseados em clusterização, por exemplo, tais como *Fuzzy C-means* e suas generalizações como *Pre-shaped fuzzy c-means* (CHEN; CHEN, 2007) são bastante empregados, mas também existem outras abordagens (KRONE; SLAWINSKI, 1998). No caso de modelagem automática de regras são utilizados métodos baseados em exemplos, como o método de Wang-Mendel (WM) (WANG; MENDEL, 1992)(WANG, 2003). Além disso, também são empregados Algoritmos Evolucionários (AE), tanto para otimização da base de regras, quanto para a de funções de pertinência. O artigo (CINTRA; CAMARGO, 2007b), utiliza Algoritmos Genéticos (AG) para definir base de regras, com pré-seleção de regras candidatas. No artigo (KIM; KIM; LEE, 2006) os autores utilizam AE para desenvolver sistemas *fuzzy* mais compactos e melhor interpretáveis ao ser humano. Em (SETNES; ROUBOS, 2000) os autores utilizam clusterização e AG para definir bons conjuntos de regras em problemas de classificação. Nos artigos (CORDóN; HERRERA, 1996) e (XUE; CHONG; JAMSHIDI, 1994) os autores utilizam técnica evolucionária e AG para desenvolver sistemas *fuzzy* a partir de bases de conhecimento.

Além disso, é necessário que o sistema seja de fácil entendimento para um ser humano, caso contrário uma das principais características de um modelo *fuzzy* é perdida, que é exatamente a facilidade de explicar seus resultados (KIM; KIM; LEE, 2006). Pois este é um dos grandes

argumentos que explica tratar um problema utilizando-se de um sistema *fuzzy* ao invés de uma rede neural, por exemplo.

Assim, o problema consiste em desenvolver automaticamente sistemas *fuzzy* que retratem o objetivo a ser alcançado com a maior precisão possível, e que mantenha a característica de ser facilmente compreendido por um ser humano. O modelo criado neste trabalho busca exatamente este equilíbrio, utilizando-se de uma função de avaliação que leva em consideração estes aspectos, conforme descreve a Seção 4.2.4.

O maior desafio é desenvolver um algoritmo para criar sistemas *fuzzy* para diferentes tipos de problema, desde que se possa definir uma função objetivo (contínua ou discreta) para o mesmo. Com o fornecimento de informações simples, como o domínio das variáveis e a função objetivo, este algoritmo deve ser capaz de encontrar um sistema *fuzzy* adequado para um determinado problema.

Para alcançar a finalidade descrita, foi utilizado um algoritmo de inteligência coletiva (Capítulo 2) conhecido como PSO. Este algoritmo é hoje utilizado em diversas aplicações de otimização.

4.2 O Modelo Proposto

Neste trabalho, para definição das funções membro do sistema *fuzzy* associado a cada partícula do algoritmo PSO, foi utilizada uma simplificação do conceito de clusterização, que é descrita na Seção 4.2.3.1. Além disso, o método WM, descrito na Seção 4.2.3.2, foi implementado para definir as regras destes sistemas *fuzzy*, apenas na inicialização do algoritmo. Desta forma, é possível fazer testes, tanto utilizando a inicialização por meio destes métodos, quanto utilizando inicialização aleatória.

O algoritmo PSO deste trabalho foi totalmente implementado em Java, orientado a objeto, sendo utilizada uma biblioteca chamada *jFuzziLogic* (CINGOLANI, 2008) apenas para a inferência dos sistemas *fuzzy*.

4.2.1 Representação

A representação de um problema no algoritmo PSO é feita através das partículas como indica a Seção 2.1.1. Cada partícula, neste modelo, possui um sistema *fuzzy* associado e uma posição no espaço de busca, que é representada por um vetor de n-posições, onde n depende do número de variáveis.

O sistema *fuzzy*, neste projeto, é definido por uma estrutura hierárquica da forma mostrada na Figura 16.

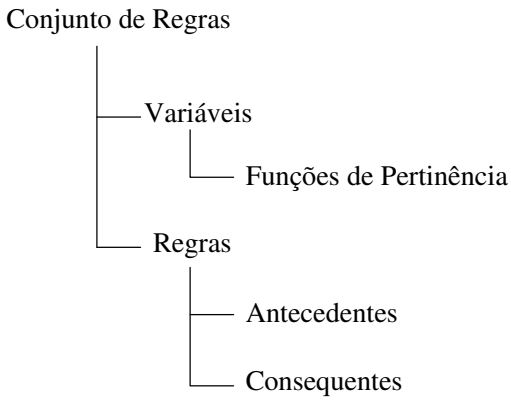


Figura 16: Estrutura do Objeto *Fuzzy*

Nesta representação o conjunto de regras é o elemento raiz da estrutura, um nível abaixo dele, e no mesmo nível entre si, temos as variáveis e as regras. As variáveis possuem funções de pertinência, as quais ficam um nível abaixo delas, assim como as regras possuem antecedentes e consequentes, que também ficam um nível abaixo delas e no mesmo nível entre si. O conjunto de regras, no geral, contém várias regras e variáveis, assim como as variáveis podem possuir várias funções de pertinência e as regras podem ter vários antecedentes e consequentes.

A estrutura dos sistemas *fuzzy* foi implementada desta forma por causa do pacote *jFuzzyLogic*, que utiliza esta hierarquia de objetos.

4.2.2 Posição das partículas

O vetor posição possui seu tamanho definido de acordo com o espaço de busca, e no modelo se dá da seguinte forma:

- um campo referente a dimensão da quantidade de regras do sistema *fuzzy*;
- um campo para o fator completude (Seção 4.2.4);
- m campos para a dimensão da quantidade de funções de pertinência de cada variável, onde m é o número de variáveis do sistema.

As dimensões de quantidade de regras e quantidade de funções de pertinência controlam a inclusão e exclusão de regras e funções de pertinência do sistema *fuzzy* associado a determinada partícula. A Figura 17 mostra como é um vetor de posição, e através dela pode-se visualizar a dependência desta estrutura em relação ao número de variáveis do sistema.

1	Quantidade de regras
2	Quantidade de funções da variável 1
⋮	⋮
m+1	Quantidade de funções da variável m
m+2	Fator completude

Figura 17: Vetor posição

O motivo pelo qual a dimensão da posição é dependente do número de variáveis do problema é o fato de cada variável poder ter uma quantidade de funções de pertinência diferente, independente de ser variável de entrada ou de saída, e deseja-se atualizar estas quantidades durante o processamento do PSO.

A atualização de cada um dos itens do vetor de posição é que proporciona a movimentação das partículas no espaço de busca.

4.2.3 Inicialização

O primeiro passo na execução do algoritmo PSO é a inicialização de todas as partículas do enxame. Neste caso deve-se inicializar todos os sistemas fuzzy, a posição de cada partícula, além dos parâmetros utilizados no algoritmo.

Neste trabalho as partículas podem ser inicializadas de duas formas:

A primeira forma de inicialização é totalmente aleatória, ou seja, não apenas os parâmetros do PSO, como posição, velocidade e melhores posições, mas também os parâmetros dos sistemas *fuzzy*, como regras e funções de pertinência, associados a cada partícula, são definidos aleatoriamente.

A segunda forma de inicialização é utilizando o método WM para definir as regras de cada sistema *fuzzy*, assim como a utilização de um controle, baseado em partições (como uma clusterização simplificada), para obter as funções de pertinência de cada variável do sistema.

4.2.3.1 Inicialização das funções de pertinência

No caso da definição de funções de pertinência, o algoritmo define aleatoriamente, para cada variável, a quantidade de funções de pertinência a definir e divide o domínio da variável pelo número de funções de pertinência mais um (isto porque cada suporte de função de pertinência poderá ocupar duas partições, inicialmente), o quociente desta divisão é o tamanho da partição.

Como exemplo, a Figura 18 ilustra o caso de uma variável em que o domínio varia de zero a doze.

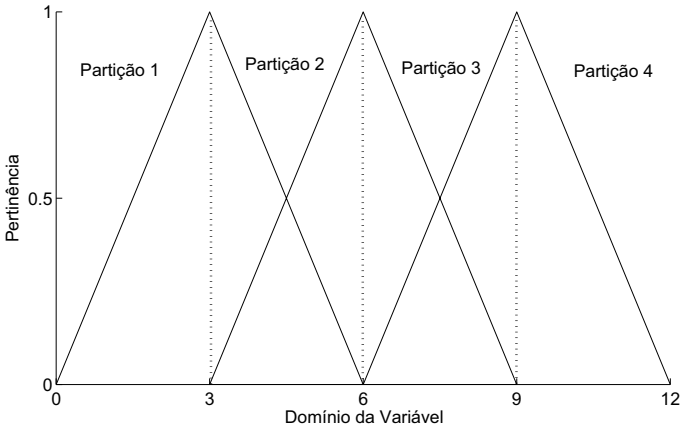


Figura 18: Particionamento Ruspini

Supondo que foi definido aleatoriamente que serão criadas três funções de pertinência, o domínio da variável seria dividido em quatro partições de tamanho três, e o suporte de cada uma das funções de pertinência seria criado dentro do intervalo de duas partições consecutivas, como na Figura 18. Mas esta figura é apenas para melhor visualização das partições, pois na verdade o algoritmo não cria as funções de pertinência neste enquadramento Ruspini, e sim define aleatoriamente onde será o centro e o tamanho do suporte de cada função de pertinência.

A Figura 19, mostra um gráfico mais próximo da realidade do algoritmo, onde pode-se perceber que as funções de pertinência definidas não estão perfeitamente centralizadas, nem com seus pontos mínimo e máximo do suporte obrigados a tocar no mínimo ou máximo de cada partição.

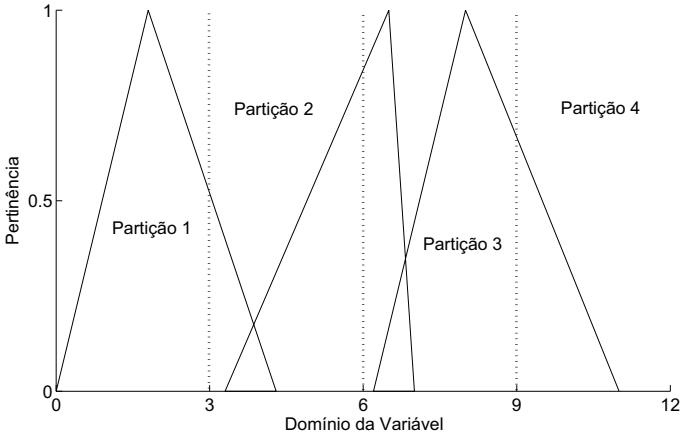


Figura 19: Particionamento Real

O fato da Figura 18 representar um "Particionamento Ruspini", não sugere que seja o que o algoritmo deve alcançar, é apenas uma diferenciação. Este controle por particionamento é executado quando as funções de pertinência estão sendo definidas. Durante o processamento do PSO, no procedimento de alteração de parâmetros das funções de pertinência, este controle não é executado, mas é utilizado quando o algoritmo cria novas funções de pertinência (Seção 4.2.5). Neste caso, o algoritmo verifica quantas funções de pertinência existem e cria as partições considerando o total de funções de pertinência como sendo este valor mais um, que é a nova função de pertinência, e executa os passos descritos acima. Mas há uma diferença neste processo que é a escolha da partição com maior espaço vazio para a criação da nova função de pertinência.

4.2.3.2 Inicialização das regras

A inicialização das regras do conjunto de regras pelo método WM requer que as funções de pertinência já tenham sido criadas, conforme descrito acima. Após a criação das funções de pertinência, um conjunto de dados de entrada e saída conhecidos é utilizado para a criação das regras. Supondo um sistema com duas variáveis de entrada e uma de saída, cada item de dados de entrada e saída possui o formato indicado na Equação 47.

$$(x_1^1, x_1^2, y_1); (x_2^1, x_2^2, y_2); ... \tag{47}$$

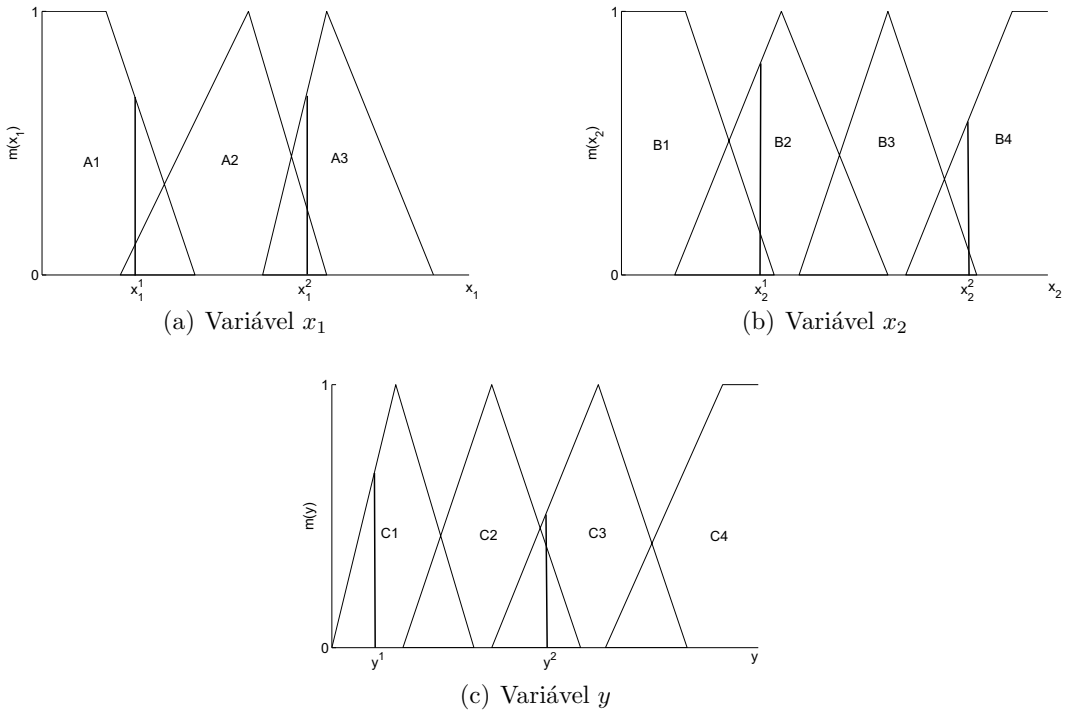
O primeiro passo para se definir uma regra é determinar o grau de pertinência para cada valor de variável (x_1^i, x_2^i, y^i), de um determinado item de dado nas diferentes regiões, ou seja, nos diferentes conjuntos *fuzzy*. A Figura 20 mostra um possível gráfico de pertinência para as três variáveis. Por exemplo, na Figura 4.20(a) podemos verificar que x_1^1 possui grau de pertinência igual a 0.67 no conjunto $A1$, 0.11 no conjunto $A2$ e zero no conjunto $A3$. Do mesmo modo, pode-se observar pela Figura 4.20(b) que x_2^2 possui grau de pertinência igual a 0.08 no conjunto $B3$, 0.58 no conjunto $B4$ e zero nos outros conjuntos.

O segundo passo na definição da regra é considerar para o item (x_1^i, x_2^i, y^i) apenas as regiões em que possua maior grau de pertinência. Por exemplo, x_1^1 na Figura 4.20(a) é considerado como sendo $A1$, e x_2^2 na Figura 4.20(b) é considerado como sendo $B4$.

Finalmente, após encontrar a região de cada variável de entrada e saída, pode-se definir a regra, como mostram as Equações 48 e 49 para os itens de dados 1 e 2, respectivamente, os quais estão representados no exemplo da Figura 20.

$$(x_1^1, x_2^1, y^1) \Rightarrow [x_1^1(0.67 \text{ em } A1),\ x_2^1(0.80 \text{ em } B2);\ y^1(0.66 \text{ em } C1)]$$

$$\text{Regra 1: SE } x_1 \text{ é } A1 \text{ e } x_2 \text{ é } B2 \text{ ENTÃO } y \text{ é } C1 \tag{48}$$



(a) Variável x_1

(b) Variável x_2

(c) Variável y

Figura 20: Funções de pertinência para exemplo do método WM

$$(x_1^2, x_2^2, y^2) \Rightarrow [x_1^2(0.67 \text{ em } A3),\ x_2^2(0.58 \text{ em } B4);\ y^2(0.51 \text{ em } C3)]$$
$$\text{Regra 2: SE } x_1 \text{ é } A3 \text{ e } x_2 \text{ é } B4 \text{ ENTÃO } y \text{ é } C3 \tag{49}$$

No processo de desenvolvimento de regras pode ocorrer de serem obtidas regras conflitantes, ou seja, regras com o mesmo antecedente, mas com diferentes consequentes. Para resolver este conflito cria-se um valor de grau para cada regra definida, utilizando-se do grau de pertinência de cada componente da regra, e então descarta-se a regra que possuir o menor grau. Esta solução, não apenas resolve o conflito, mas também diminui o tamanho da base de regras com o descarte de algumas regras. A Equação 50 mostra como é calculado o grau (G) de uma regra, com base nas informações do exemplo acima.

$$G(Regra) = m_i(x_1).m_i(x_2).m_i(y) \tag{50}$$

Desta forma, a Equação 51 mostra o cálculo do grau de regra para a Regra 1 definida na Equação 48.

$$G(Regra) = m_1(x_1).m_1(x_2).m_1(y) = 0.67 \times 0.80 \times 0.66 = 0.306 \tag{51}$$

E a Equação 52 mostra o cálculo para a Regra 2 definida na Equação 49.

$$G(Regra) = m_2(x_1).m_2(x_2).m_2(y) = 0.67 \times 0.58 \times 0.51 = 0.198 \tag{52}$$

4.2.4 Função de Avaliação

Para que se possa definir se uma determinada partícula se encontra em uma posição boa ou ruim é preciso diferenciar cada uma por sua "aptidão", assim, deve-se criar um critério de pontuação (*fitness*) que seja condizente com o problema, neste caso, uma avaliação para o sistema *fuzzy*. Com este propósito foi definida a Equação 53, baseada no artigo (KIM; KIM; LEE, 2006).

$$\begin{aligned} F = \; & -100.K1.log(MSE) \\ & +50.K2(1 - COMP_r) \\ & +50.K3(1 - COMP_f) \\ & +50.K4(1 - P_D) \\ & +50.K5(1 - P_C) \end{aligned} \tag{53}$$

MSE é a média dos erros quadráticos, ou seja, da diferença entre o valor retornado pela função objetivo (y_h) e o retornado pelo modelo *fuzzy* (y_h^F), conforme Equação 54.

$$MSE = \frac{1}{N_D} \sum_{h=1}^{N_D} (y_h - y_h^F)^2, \tag{54}$$

onde N_D é o número de dados. O sinal negativo no primeiro termo da Equação 53, faz com que o valor do fitness aumente, conforme o valor de erro MSE diminui. Por exemplo, para $K1 = 1$ e $MSE = 10^{-6}$ o valor deste primeiro termo seria 600, agora se $MSE = 10^6$, então o valor deste termo iria para -600, que é bem menor que o anterior.

O termo $COMP_r$ é definido como sendo a relação entre a quantidade de regras presentes no sistema *fuzzy* e o total de regras possíveis, e $COMP_f$ como a relação entre a quantidade de funções de pertinência presentes no sistema e o total de funções de pertinência possíveis, conforme Equação 55. O valor destes componentes pode variar entre 0 e 1.

$$COMP_r = \frac{N_R}{N_R^{max}} \quad COMP_f = \frac{N_F}{N_F^{max}} \tag{55}$$

O número máximo de regras N_R^{max} é a soma de cada combinação possível de variáveis de entrada e saída, levando-se em consideração que uma regra pode conter todas, algumas ou apenas uma das variáveis de entrada em seu antecedente. Um exemplo poderia ser o caso de um sistema com três variáveis de entrada e uma de saída. Supondo que as variáveis de entrada x_1, x_2 e x_3, possuem, respectivamente, $m(x_1) = 2$, $m(x_2) = 3$ e $m(x_3) = 3$ conjuntos *fuzzy*, e que a variável de saída y possui $m(y) = 3$ conjuntos *fuzzy*, o N_R^{max} seria dado pela Equação 56.

$$\begin{aligned} &1) \quad m(x_1) \times m(x_2) \times m(x_3) \times m(y) = 2 \times 3 \times 3 \times 3 = 54 \\ &2) \quad m(x_1) \times m(x_2) \times m(y) = 2 \times 3 \times 3 = 18 \\ &3) \quad m(x_1) \times m(x_3) \times m(y) = 2 \times 3 \times 3 = 18 \\ &4) \quad m(x_2) \times m(x_3) \times m(y) = 3 \times 3 \times 3 = 27 \\ &5) \quad m(x_1) \times m(y) = 2 \times 3 = 6 \\ &6) \quad m(x_2) \times m(y) = 3 \times 3 = 9 \\ &7) \quad m(x_3) \times m(y) = 3 \times 3 = 9 \end{aligned} \tag{56}$$

Assim, o valor de N_R^{max} é a soma do resultado dos passos 1 ao 7 da Equação 56, ou seja, $N_R^{max} = 54 + 18 + 18 + 27 + 6 + 9 + 9 = 141$. Isso quer dizer que há 141 possibilidades de regras para este exemplo. Se houvesse uma segunda variável de saída, a quantidade de funções de pertinência da mesma seria multiplicado em todos os passos da Equação 56, pois todas as variáveis de saída devem estar presentes na regra.

Na verdade o caso apresentado na Equação 56, é de regras possíveis, mas em um sistema *fuzzy* real não deve haver regras com antecedentes iguais. Deste modo, o número de regras máximo que pode haver em um conjunto de regras N_{CR}^{max} deverá considerar apenas uma vez cada consequente. Para o exemplo acima a Equação 57 mostra como ficaria.

$$\begin{array}{ll} 1) & m(x_1) \times m(x_2) \times m(x_3) = 2 \times 3 \times 3 = 18 \\ 2) & m(x_1) \times m(x_2) = 2 \times 3 = 6 \\ 3) & m(x_1) \times m(x_3) = 2 \times 3 = 6 \\ 4) & m(x_2) \times m(x_3) = 3 \times 3 = 9 \\ 5) & m(x_1) = 2 \\ 6) & m(x_2) = 3 \\ 7) & m(x_3) = 3 \end{array} \tag{57}$$

Neste caso, o valor máximo de regras no conjunto de regras será $N_{CR}^{max} = 18 + 6 + 6 + 9 + 2 + 3 + 3 = 47$. A interpretação para isto é que tem-se N_R^{max} possibilidades de regras, mas destas, apenas N_{CR}^{max} poderão constar no conjunto de regras de um sistema *fuzzy* desenvolvido pelo algoritmo.

O valor de N_F^{max} depende dos parâmetros do algoritmo que definem a quantidade de funções de pertinência possíveis para cada variável, o valor de N_F^{max} é a soma desses valores. Se no exemplo acima fosse definido que cada variável, tanto de entrada, quanto de saída, poderia ter até três funções de pertinência, então $N_F^{max} = 4 \times 3 = 12$, onde o valor 4 corresponde às quatro variáveis, três de entrada e uma de saída, e o valor 3 é a quantidade de funções de pertinência de cada uma.

O termo P_D é um critério que mede a distinguibilidade entre as funções de pertinência das variáveis, conforme Equação 58. Esta avaliação é importante para tentar impedir a formação de funções de pertinência com suportes sobrepostos, onde não se pode definir claramente a que conjunto *fuzzy* um determinado valor de variável pertence.

$$P_D = \frac{1}{N_V} \sum_{i=1}^{N_V} \left(\frac{1}{N_S^i} \sum_{h=1}^{N_S^i} \frac{\lambda_i h}{|\chi_i|} \right) \tag{58}$$

onde N_V é o número de variáveis, N_S^i é a quantidade total possível de sobreposição entre funções de pertinência da i-ésima variável, λ_{ih} é a largura da h-ésima sobreposição e χ_i é a largura do domínio da variável. Para a medida de λ_{ih}, é necessário definir o nível ξ_D, traçando-se uma reta

constante em pertinência, cruzando todas as funções de pertinência, como mostra a Figura 21. O valor de N_S^i é calculado pela combinação matemática $C_2^{m_i}$, onde m_i é o número de funções de pertinência da i-ésima variável (KIM; KIM; LEE, 2006).

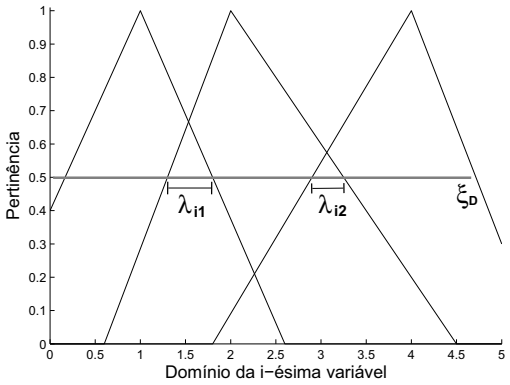


Figura 21: Definição da sobreposição

O termo P_C é um critério que mede a completude das funções de pertinência em relação ao domínio, conforme Equação 59. Esta avaliação é importante para tentar impedir a formação de espaços vazios entre os conjuntos *fuzzy*, onde um determinado valor de variável não pertenceria a nenhum deles.

$$P_C = \frac{1}{N_V} \sum_{i=1}^{N_V} \left(\frac{1}{N_D^i} \sum_{h=1}^{N_D^i} \frac{\gamma_i h}{|\chi_i|} \right) \tag{59}$$

onde N_D^i é a quantidade total possível de descontinuidade entre funções de pertinência da i-ésima variável, γ_{ih} é a largura da h-ésima descontinuidade e χ_i é a largura do domínio da variável. Para a medida de γ_{ih}, é necessário definir o nível ξ_C, de forma similar a ξ_D, como mostra a Figura 22. O valor de N_D^i é calculado da mesma forma que o de N_S^i (KIM; KIM; LEE, 2006).

As constantes $K1, K2, K3, K4$ e $K5$, controlam a participação de cada termo da Equação 53 na avaliação das partículas. Desta forma, pode-se definir o quanto cada um deles irá influenciar no processo de busca. Estes termos propiciam uma grande flexibilidade na execução dos testes.

Esta forma de pontuação foi selecionada pelo fato de abranger os diversos critérios de um sistema *fuzzy*, como a acurácia, pela medida do erro; a compacidade, pela relação entre o número de regras e funções de pertinência do modelo e o total possível; e a interpretabilidade, pela medida da distinguibilidade e da completude entre as funções de pertinência, propiciando

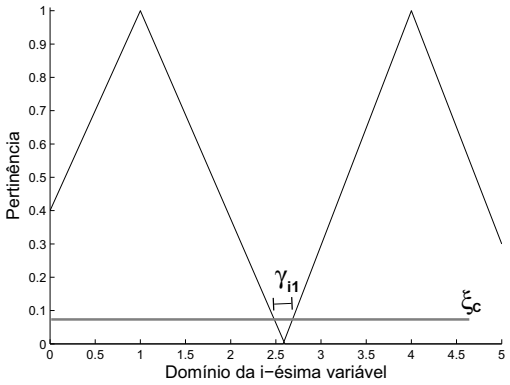


Figura 22: Definição da descontinuidade

uma avaliação mais completa. Maiores detalhes desta forma de avaliação é encontrada em (KIM; KIM; LEE, 2006).

No Artigo (CINTRA; CAMARGO, 2007b), os autores citam uma forma de avaliação do sistema *fuzzy* pelo conjunto de regras, que serviu de inspiração na utilização do método WM na definição de regras do algoritmo deste trabalho, mas por não contemplar todos os aspectos de um sistema *fuzzy* citados acima, não foi considerado o melhor método para esta implementação.

4.2.5 Operações para Modificação dos Sistemas *Fuzzy*

Foram definidos quatro tipos de operações para alteração dos sistemas *fuzzy* das partículas em cada iteração do algoritmo, que são apresentadas abaixo:

- Alteração da quantidade de regras do sistema *fuzzy*. Se a velocidade referente ao número de regras for positiva, aumenta-se a quantidade de regras. Caso contrário deve-se diminuir a quantidade de regras. No caso da velocidade positiva os valores de $\hat{x}_{ij}(t)$ (melhor posição da partícula i) e, $\bar{x}_j(t)$ ou $\bar{x}_{ij}(t)$ (melhor posição entre todas as partículas da vizinhança), nas Equações 20 e 21 do Capítulo 2, provavelmente são maiores do que o valor de $x_{ij}(t)$, e como regra esta posição deve se aproximar daquelas. As regras são criadas ou excluídas aleatoriamente.

- Alteração da quantidade de funções de pertinência de cada variável do sistema *fuzzy*. Segue o mesmo critério da alteração da quantidade de regras, descrito acima. Ao criar novas funções de pertinência é utilizado o controle de partições, conforme descrito na Seção 4.2.3, o algoritmo verifica quantas funções de pertinência existem e cria as partições considerando o total de funções de pertinência como sendo este valor mais um, que é a função de pertinência a ser criada, e executa os passos descritos na Seção 4.2.3. A nova

função de pertinência é colocada na partição com maior espaço vazio. Ao remover uma função de pertinência busca-se retirar a que esteja no local onde existe menos espaço vazio no domínio da variável, ou seja, onde exista maior concentração de suporte de funções de pertinência.

- Alteração da completude (Seção 4.2.4). Aumentando o suporte de uma função de pertinência a tendência é diminuir os espaços vazios no domínio da variável, caso existam (ver Figura 22). Quanto mais positiva a velocidade, maior o aumento no suporte de uma das funções de pertinência, se o sinal for negativo diminui o suporte. Neste caso, quando é para aumentar o suporte de uma função de pertinência, escolhe-se a função de pertinência que possui o menor suporte e esteja em um espaço onde há um "buraco" maior no domínio da variável, quando for para diminuir o suporte de uma função de pertinência faz-se o contrário.

- Alteração na posição da função de pertinência. Neste caso, é selecionada uma das funções de pertinência aleatoriamente, e sua posição é alterada negativa, ou positivamente, em um valor percentual definido no algoritmo. O fato da alteração ser positiva ou negativa, ou seja, a função de pertinência ser levada em direção ao valor mínimo ou máximo do domínio, também é decidido de forma a diminuir os "buracos" no domínio.

No caso da alteração de quantidade de regras e de funções de pertinência, como os valores de velocidade são números reais e não servem para definir a quantidade a ser alterada, é necessário construir uma forma para poder definir essa quantidade. Para isso, primeiro é necessário definir o valor mínimo e máximo possível para cada alteração desses parâmetros, então executa-se a normalização da velocidade dentro deste limite, e finalmente o valor obtido na normalização é arredondado para um valor inteiro, que será a quantidade de regras ou funções de pertinência a serem criadas ou removidas. Por exemplo, numa situação hipotética, supondo que o valor mínimo (R_{min}) para alterar regras seja definido como zero, e o máximo (R_{max}) seja cinco, e que a velocidade máxima, descrita na Seção 2.2.2 do Capítulo 2 (V_{max}) seja igual a 15.5, ou seja, a velocidade mínima (V_{min}) é igual a -15.5, e que a velocidade de uma partícula (V_p) em determinado momento seja igual a três. Através da Equação 60 obtém-se o valor normalizado da velocidade dentro dos limites de R_{min} e R_{max}.

$$\eta = \frac{V_p + V_{max}}{2 * V_{max}}(R_{max} - R_{min}) + R_{min} \tag{60}$$

Substituindo-se os valores acima na Equação 60, chega-se ao resultado mostrado na Equação 61.

$$\eta = \frac{3 + 15.5}{2 * 15.5}(3 - 0) + 0 = 1.79 \tag{61}$$

Agora é necessário arredondar este valor para obter a quantidade de regras que serão criadas, já que a velocidade é positiva. Arredondando o valor $\eta = 1.79$, para o inteiro mais próximo, encontramos a quantidade $Q = 2$ regras a serem criadas.

Este mesmo processo pode ser utilizado para encontrar a quantidade de funções de pertinência a serem criadas ou removidas, mas neste caso é feito para cada variável do sistema fuzzy, tanto de entrada quanto de saída.

Para a alteração definida pelo critério completude alguns passos devem ser seguidos, como descobrir o maior ou menor espaço no domínio da variável e qual a função de pertinência com maior ou menor suporte. Para isso, consideremos X_i como sendo o valor entre o início do suporte da primeira função de pertinência e o limite inferior, L_i, do domínio da variável do problema, X_e como sendo o valor entre o fim do suporte de uma função de pertinência e o início do suporte da função de pertinência seguinte, e X_f como sendo o valor entre o fim do suporte da última função de pertinência e o limite superior, L_s, do domínio da variável do problema, como mostra a Figura 23.

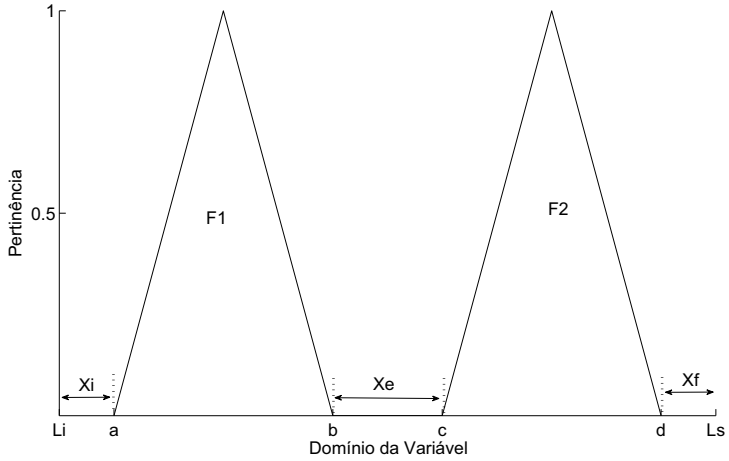


Figura 23: Distâncias com X_e Positivo

Consegue-se então chegar às conclusões mostradas na Equação 62, para calcular estas distâncias.

$$\begin{aligned} X_i &= a - L_i \\ X_e &= c - b \\ X_f &= L_s - d \end{aligned} \tag{62}$$

Assim, pela Equação 62, pode-se verificar que $X_e > 0$ quando $c > b$ e $X_e < 0$ quando $c < b$. Ou seja, se X_e é positivo quer dizer que existe "buraco" no domínio da variável. Dessa forma se for considerado que uma variável possui n funções de pertinência, ela possuirá $n - 1$ valores de X_e, sendo $X_e(i, j)$ o valor entre o fim da i-ésima função de pertinência e o início da j-ésima função de pertinência, sendo j sempre a função de pertinência subsequente a i.

Observa-se pela Figura 24 que X_e negativo significa que uma função de pertinência cruza a outra, ou seja, entre as funções de pertinência i e j não existe "buraco". Mas também pode-se concluir que quanto menor for o valor de X_e mais as duas funções de pertinência se confundem.

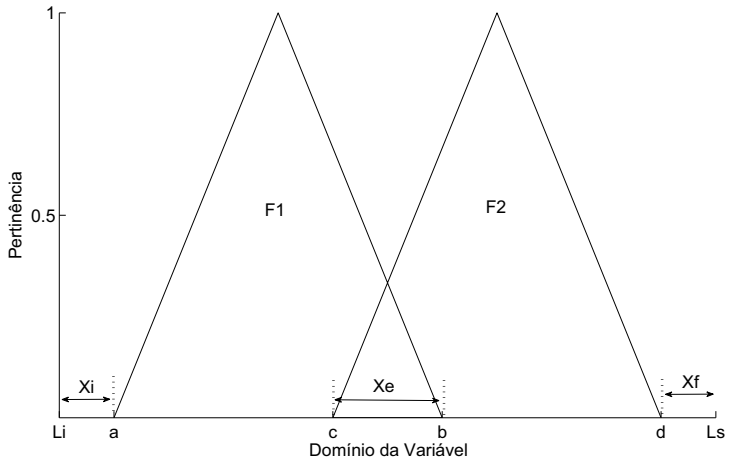


Figura 24: Distâncias com X_e Negativo

Desta forma o critério completude tenta diminuir o valor de X_e positivo, mas também aumentar o valor de X_e negativo, neste caso trabalhando na verdade a distinguibilidade. Por isso a utilização do critério desta forma.

Deve-se pensar de forma similar em relação aos valores de X_i e X_f, sendo que neste caso a comparação não é entre duas funções de pertinência, mas entre função de pertinência e limite do domínio da variável.

Quando é aplicado o critério da completude, verifica-se no caso da velocidade ser positiva qual o maior valor entre X_i, X_f e cada $X_e(i, j)$, a fim de aumentar o suporte da função de pertinência mais adequada a reduzir o maior destes espaços. Se o maior valor for de X_i ou de X_f então a função de pertinência a ser selecionada é a primeira, ou a última, respectivamente, mas se o maior valor for de $X_e(i, j)$, deve ser verificada qual das duas funções de pertinência i ou j, possui o menor suporte.

Quando a velocidade é negativa, o algoritmo verifica qual o menor valor entre este três, a fim de diminuir o suporte da função de pertinência mais adequada a tornar mais distintas as funções de pertinência. Se o menor valor for de X_i ou de X_f, será escolhida a primeira ou a última função de pertinência respectivamente. Mas se o menor valor for de $X_e(i, j)$, deve ser verificado qual das duas funções de pertinência, i ou j, possui o maior suporte.

O valor no qual uma função de pertinência será alterada é definido como um percentual do suporte da própria função de pertinência. A velocidade da dimensão completude representa este valor percentual, sendo a velocidade máxima desta dimensão um valor que limita este percentual. Por exemplo, se a velocidade for de 0.25 e a variável permite valores de 0 a 50, temos o domínio da variável $D_v = 50 - 0 = 50$. Supondo que uma função de pertinência tenha seu primeiro ponto do suporte na origem do domínio da variável e o último ponto no valor 10 então temos o suporte desta função de pertinência $D_{f1} = 10 - 0 = 10$, o valor de alteração será $V_a = 0.25 * 10 = 2.50$, e o novo valor de suporte será $D_{f2} = 10 + 2.50 = 12.50$. Isto equivaleria, no caso da função de pertinência $F1$ da Figura 23, a incrementar o valor de b de 2.50, ou ainda em decrementar o valor de c, desta mesma figura em 2.50.

4.3 Considerações Finais do Capítulo

Neste capítulo foram apresentados os detalhes do modelo elaborado no trabalho. Nele foi descrita a representação do problema, o posicionamento das partículas no espaço, a forma de inicialização do algoritmo e suas partículas, quais são os critérios de avaliação, além de como ela se processa e seus parâmetros, e finalmente, como as soluções são modificadas de forma a desenvolver novas soluções, ou seja, como as partículas são alteradas.

Capítulo 5

TESTES E RESULTADOS

NESTE capítulo são apresentados os resultados encontrados nos testes com o PSO e a comparação com os trabalhos (RIVAS et al., 2003) e (CINTRA; CAMARGO, 2007b), que utilizam algoritmo genético para encontrar parâmetros de sistemas *fuzzy*.

Os testes com o algoritmo foram separados em duas partes, uma com funções contínuas e outra com funções discretas. Na Seção 5.1, são definidos os parâmetros utilizados no PSO, em seguida são apresentados resultados obtidos em experimentos com funções contínuas e a comparação com o trabalho (RIVAS et al., 2003), enquanto a Seção 5.2 expõe os resultados encontrados quando a função objetivo é uma função discreta e a comparação com o trabalho (CINTRA; CAMARGO, 2007b).

5.1 Resultados de Testes com Funções Contínuas

Nesta seção serão apresentados os testes e resultados com funções contínuas, mas primeiro será mostrado como foram definidos os parâmetros do PSO a serem utilizados.

O ambiente utilizado para executar o algoritmo foram máquinas com processador core 2 quad com 2.4Ghz de clock, 4Gb de memória ram e sistema operacional Windows Vista 64 bits. O programa foi feito na plataforma Java, utilizando a IDE Eclipse GANIMEDE.

Cada execução do algoritmo utilizou uma semente diferente para gerar números aleatórios, pois a linguagem Java utiliza a hora do sistema, em milissegundos, como semente.

5.1.1 Definição dos parâmetros do PSO

Para descobrir os parâmetros do PSO que demonstrassem resultados significativos foi usada a função seno ($y = seno(x)$), com o domínio da variável x, dado por:

$$\begin{cases} x \in \mathbb{R} \mid 0 \leq x \leq 2\pi \\ y \in \mathbb{R} \mid -1 \leq y \leq 1 \end{cases}$$

Os testes foram executados para valores de parâmetros mostrados nas Figuras 25 a 27, com a quantidade de funções de pertinência variando de três a nove. Foram feitas 1080 execuções, com 5000 iterações cada uma, variando-se os parâmetros conforme mostram as figuras. Na execução do algoritmo, o domínio da variável x foi dividido em cem valores igualmente espaçados para comparar a saída y dos sistemas *fuzzy* gerados com a saída original.

O número de execuções de cada caso de teste é o total de execuções, 900, dividido pelo número de casos de teste. Cada parâmetro possui um determinado número de casos de teste que é na verdade a quantidade de valores testados para tal parâmetro, que são mostrados na legenda das Figuras 25 a 28.

A Tabela 2 e a Figura 25 apresentam os valores utilizados para os coeficientes cognitivo e social. Os valores destes dois parâmetros foram escolhidos de forma que a soma dos dois não ultrapassasse o valor quatro, pelos motivos descritos na Seção 2.2.3. O gráfico desta figura mostra a média de aptidão para cada combinação destes coeficientes.

Tabela 2: Resultados para os coeficientes cognitivo e social

Coeficiente Cognitivo	Coeficiente Social	Execuções	Avaliação Média
1.1	1.1	120	332.345000
1.1	1.5	120	331.420715
1.1	1.9	120	332.593947
1.5	1.1	120	332.900051
1.5	1.5	120	334.206482
1.5	1.9	120	332.630159
1.9	1.1	120	332.628457
1.9	1.5	120	332.120244
1.9	1.9	120	332.000102

A quantidade de partículas foi definida com valores baseados na Seção 2.2.3, e com a intenção de tornar o custo computacional o menor possível. A Tabela 3 e a Figura 26 mostram as quantidades de partículas utilizadas (legenda). Os testes são combinados com os coeficientes cognitivo e social, ou seja, para cada quantidade de partículas foram executadas as nove combinações de coeficientes cognitivo e social.

Tabela 3: Resultados para a quantidade de partículas

Qtd. Partículas	Execuções	Avaliação Média
7	270	328.339845
10	270	331.589471
15	270	334.248710
20	270	335.854931

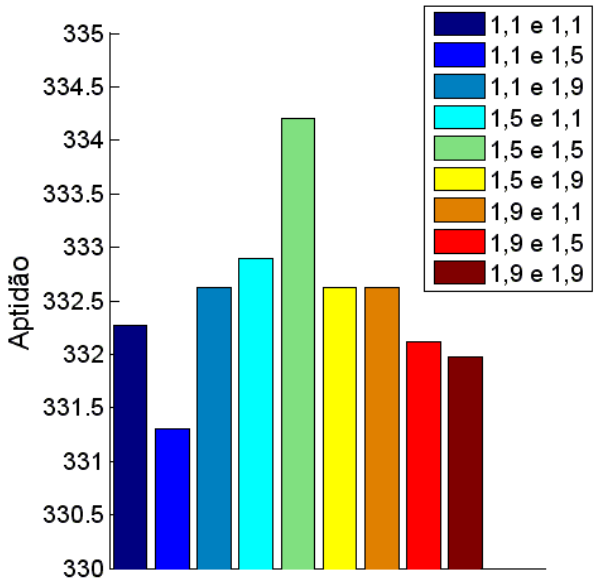


Figura 25: Resultados para os coeficientes cognitivo e social

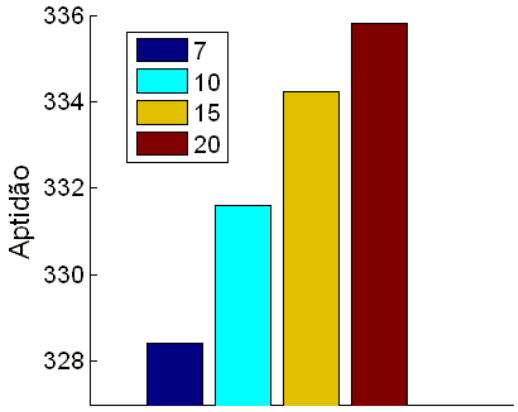


Figura 26: Resultados para a quantidade de partículas

O coeficiente de inércia do PSO também é descrito na Seção 2.2.3, na qual é indicado que valores que apresentam bom desempenho costumam ser bem próximos de um, sendo este o valor máximo para tal parâmetro. As informações dos testes estão na Tabela 4 e na Figura 27, que apresentam a média de aptidão nos testes para diferentes valores deste parâmetro. Estes testes também são combinados com os nove casos de teste dos coeficientes social e cognitivo.

A Seção 4.2.5 define a atualização do algoritmo utilizando o critério completude. Este critério foi definido como uma taxa a ser aplicada em relação ao suporte da função de pertinência, assim fez-se necessário definir a taxa máxima permitida no algoritmo para esta atualização. A Tabela 5 e a Figura 28 apresentam o resultado dos testes com alguns valores para a completude.

Tabela 4: Resultados para o coeficiente de inércia

Coeficiente de Inércia	Execuções	Avaliação Média
0.10	216	332.898775
0.15	216	332.008201
0.25	216	331.873014
0.50	216	331.621648
1.00	216	334.108953

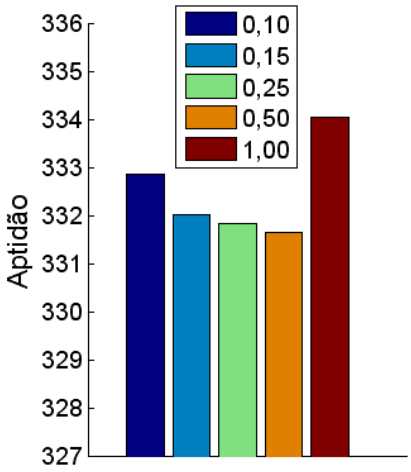


Figura 27: Resultados para o coeficiente de inércia

Com o auxílio da Figura 25 pode-se verificar que para a combinação dos valores de coeficiente cognitivo e social o melhor resultado apresentado foi de 1.5 para cada um, levando-se em conta a média da avaliação. Enquanto a Figura 26 mostra que para a quantidade de partículas o melhor resultado foi com uma população de vinte. Para o coeficiente de inércia, a Figura 27 indica que dos valores testados o melhor resultado foi conseguido quando ele foi igual a um. Por último a Figura 28 demonstra que a completude obteve melhor resultado com valor igual a 0.25.

Assim, para todos os testes e resultados que serão apresentados nas seções seguintes, os valores utilizados como parâmetros do PSO foram aqueles que se saíram melhor nestes testes, e que foram relacionados na Tabela 6.

O número total de iterações foi definido também de forma empírica observando-se que poucas vezes obteve-se mudança no resultado depois de mil iterações, ainda assim, o total de iterações utilizado foi bem maior que este valor para englobar casos em que houvesse melhoras depois de mil iterações. Além disso, a experiência do artigo (COSTA; NEDJAH; MOURELLE, 2009) mostrou que a inicialização com o método WM alcança os melhores resultados, e em um menor intervalo de tempo, e que um conjunto pequeno de regras geradas pelo método WM deve ser mantido durante o processamento, o valor utilizado está na Tabela 6.

Tabela 5: Resultados para a completude

Taxa da Completude	Execuções	Avaliação Média
0.25	216	335.002011
0.5	216	332.401789
0.75	216	330.195992
0.85	216	330.010671
0.95	216	331.190054

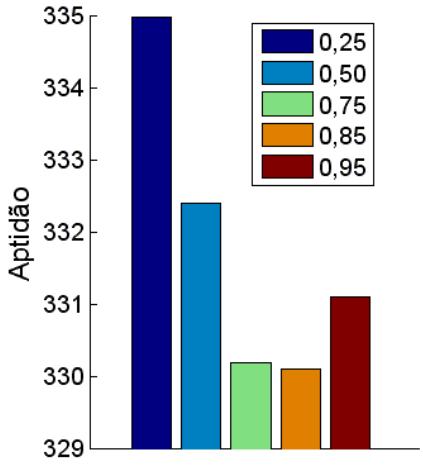


Figura 28: Resultados para a completude

Alguns parâmetros internos do algoritmo não foram descritos por não ter havido maiores testes para definí-los, como a taxa em que a alteração de funções de pertinência é realizada, pois em todas as iterações o algoritmo altera o conjunto de regras, mas foi definido que apenas após dez iterações sem alteração no melhor resultado as funções de pertinência das variáveis são alteradas. Outro parâmetro nesta situação é o total de regras do WM, citado acima.

Outros parâmetros são aqueles utilizados para definir o número de regras e funções a serem alteradas, o que é feito por uma normalização na qual o valor mínimo e máximo de regras e funções a serem alterados são os parâmetros, conforme informa a Seção 4.2.5. Foi utilizado o valor mínimo de zero e máximo de cinco para regras, e mínimo de zero e máximo de três para funções de pertinência a serem alteradas (incluídas ou removidas) durante o processamento do algoritmo. Valores menores de máximo demonstraram uma estagnação no algoritmo, e valores maiores fizeram com que o algoritmo se perdesse.

5.1.2 Função seno de duas variáveis

A função seno de duas variáveis $z = seno(xy)$, cujo gráfico é mostrado na Figura 29, obtida do artigo (RIVAS et al., 2003) para comparação com o algoritmo proposto, é uma função contínua com diferentes pontos de máximos e mínimos.

Tabela 6: Relação dos valores dos parâmetros do PSO

Parâmetro	valor
Coeficiente cognitivo	1.5
Coeficiente social	1.5
Qtd. de partículas	20
Coeficiente de inércia	1
Completude	0.25
Total de iterações	5000
Total de Regras WM	20% do total

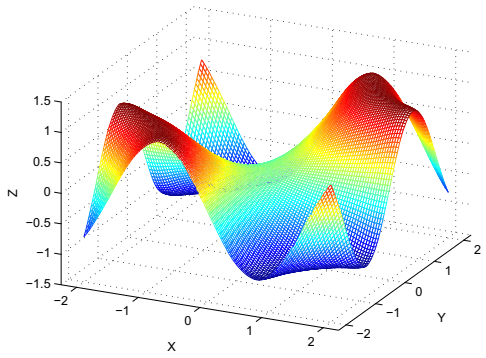


Figura 29: Função $z = seno(xy)$ utilizada para comparação

Os domínios das coordenadas desta função são definidos abaixo.

$$\begin{cases} x \in \mathbb{R} \mid -2 \leq x \leq 2 \\ y \in \mathbb{R} \mid -2 \leq y \leq 2 \\ z \in \mathbb{R} \mid -1.02 \leq z \leq 1.02 \end{cases}$$

Uma das grande dificuldades é fazer com que o sistema *fuzzy* encontre os pontos mais próximos possíveis da curva original, de forma a não haver grandes distorções.

Os testes com essa curva foram feitos, hora utilizando funções de pertinência na forma de Gaussianas, hora utilizando funções de pertinência triangulares. Além disso, em todos os testes com funções contínuas de duas variáveis os domínios das variáveis foram divididos em vinte valores, igualmente espaçados. Desta forma, o total de pontos utilizados para comparar a saída dos sistemas *fuzzy* gerados com as saídas originais foi de quatrocentos. Inicialmente, nos testes com a sigmóide, estes domínios eram divididos em dez partes, em um total de cem pontos, mas os resultados apresentados eram tão ruins ao ponto de as curvas não apresentarem nenhuma característica da original.

5.1.2.1 Testes com funções Gaussianas

Os testes com funções Gaussianas foram realizados para limites variados de funções de pertinência para as variáveis de entrada e de saída. Foram executados trezentos testes onde

eram permitidas apenas duas funções de pertinência por variável, outros trezentos permitindo para cada variável uma variação de duas a quatro funções de pertinência, e assim por diante, conforme Figura 30.

Nos testes realizados com duas a quatro funções de pertinência, estes são na verdade o limite inferior e o superior de quantidade de funções de pertinência que cada variável pode ter, a quantidade que estará realmente presente no sistema *fuzzy* para cada uma delas dependerá do processamento do algoritmo. Por exemplo, o programa pode encontrar um sistema *fuzzy* que possua duas funções para a primeira variável de entrada, três funções para a segunda, e quatro para a variável de saída. E esta configuração pode variar de uma partícula para outra.

A Tabela 7 e a Figura 30 mostram resultados, média de aptidão, de testes, por quantidade de funções de pertinência, onde a legenda da figura indica a quantidade de funções de pertinência que cada sistema *fuzzy* pode conter, podendo ser um valor fixo ou um intervalo.

Tabela 7: Valor de avaliação para cada caso de teste

Qtd. Função	Maior Avaliação	Avaliação Média
2	275.0109	273.2989
2 a 4	285.3102	282.1905
3	279.9470	275.1261
5 a 7	461.1995	445.5563
7 a 9	451.8544	425.3665

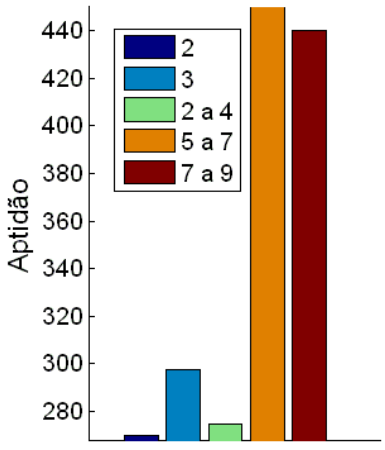


Figura 30: Valor de avaliação para cada caso de teste

O desvio padrão da avaliação de todos os resultados é dado por $\delta = 13.2780$, o que demonstra não haver uma variação tão grande no valor dos resultados obtidos.

Pela Figura 30 pode-se concluir que testes com quantidade de funções variando de cinco a sete e de sete a nove obtiveram, na média, resultados melhores do que com menos funções.

Alguns resultados dos testes com funções Gaussianas são apresentados graficamente nas Figuras 5.31(a) a 5.31(c). Em todas as figuras pode-se observar que o sistema *fuzzy* conseguiu aproximar-se do comportamento da função, ainda que havendo distorções consideráveis em relação a seus pontos.

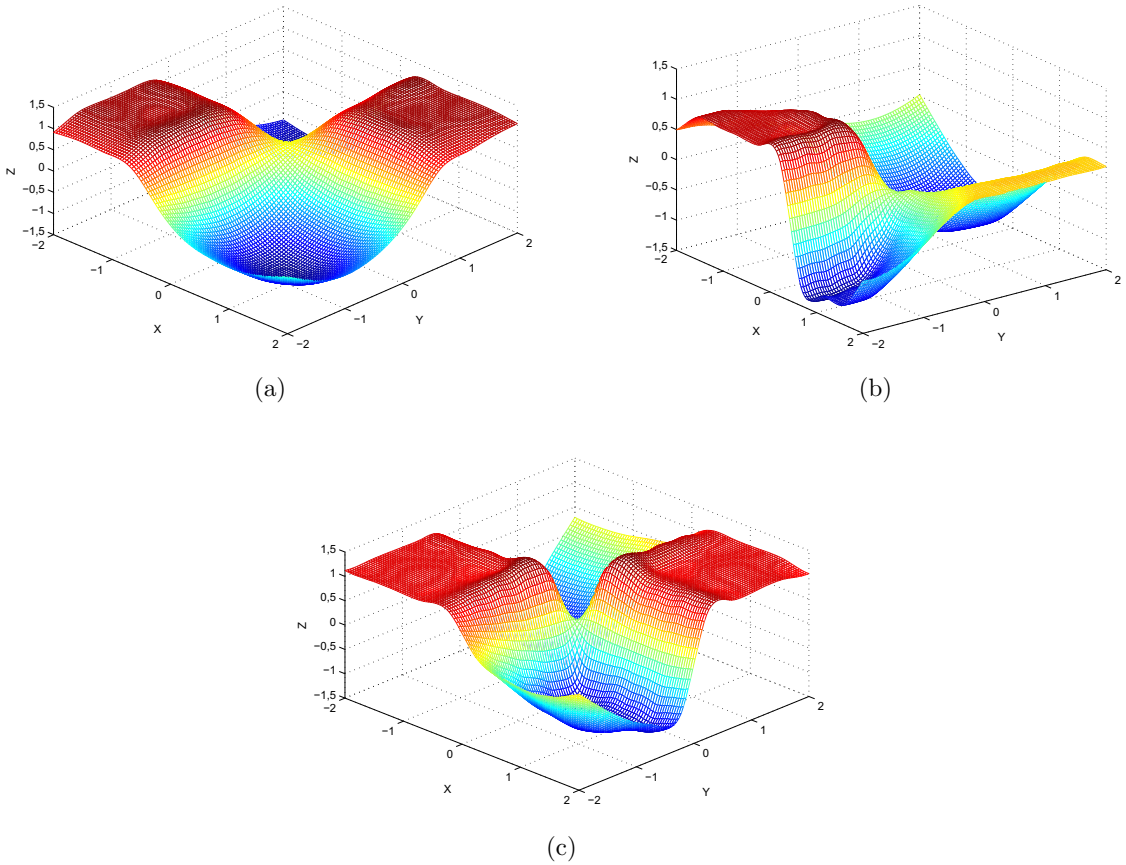
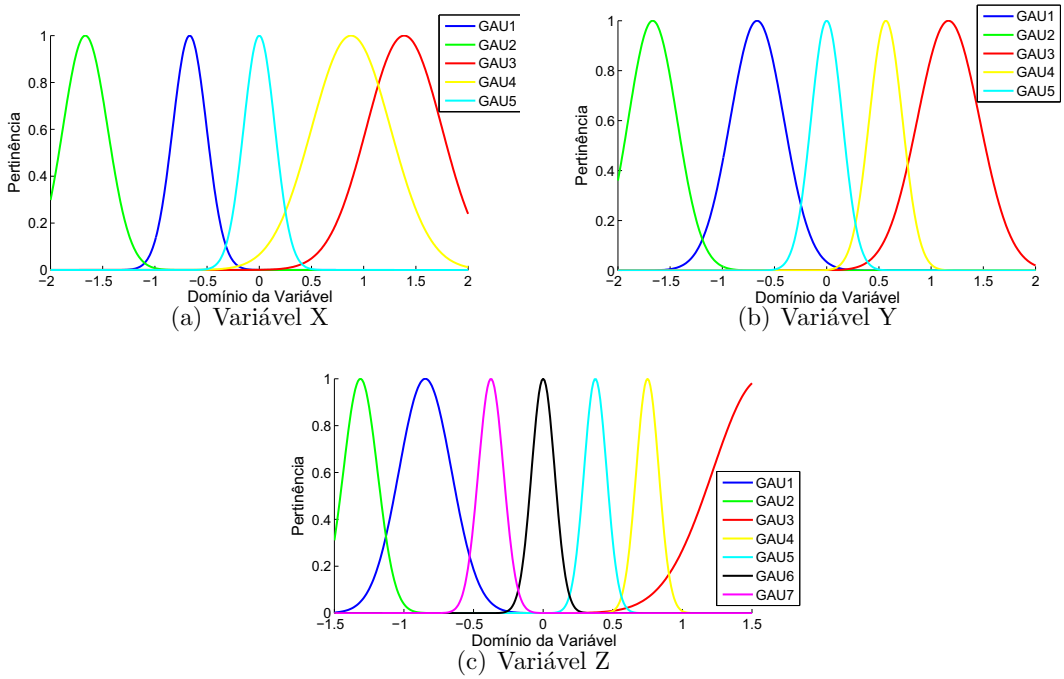


Figura 31: Superfícies resultado de sistemas *fuzzy* função seno 3D

As três figuras mostram claramente que os sistemas *fuzzy* encontrados não conseguem reproduzir fielmente a curva original, apesar de apresentarem um comportamento aproximado.

Diversas outras curvas foram encontradas durante todos os testes, todas apresentaram resultados similares às três curvas mostradas acima.

A Figura 32 mostra a configuração das variáveis de entrada e saída de um dos sistemas *fuzzy* que foram resultado do algoritmo. A numeração das funções de pertinência não segue a ordem em que elas aparecem no domínio devido ao fato do objeto Java utilizado no programa não respeitar esta ordem, e para não incorrer em erros decidiu-se mostrar o resultado na forma que aparece no sistema.



(a) Variável X (b) Variável Y

(c) Variável Z

Figura 32: Gráfico de pertinência das variáveis fuzzy para o teste da função seno

Na Seção A.1 do Apêndice A encontra-se o conjunto de regras gerado para este sistema *fuzzy*. Neste caso foram geradas 21 regras, quando o número máximo de regras que poderiam fazer parte deste conjunto de regras, baseado na quantidade de funções de pertinência de cada variável da Figura 32, e conforme cálculo descrito na Seção 4.2.4, é igual a 35, e o máximo de regras que podem ser geradas para esta configuração é de 245.

A Figura 33 mostra um gráfico onde a média quadrática dos erros normalizados (do inglês normalized mean square error, NMSE) é calculada para cada ponto utilizado para testar os sistemas *fuzzy* resultado. São os mesmos cem pontos, de cada resultado, que são utilizados para gerar as curvas de saída. O erro é normalizado para valores entre zero e um, e o eixo das abscissas simboliza cada um dos pontos.

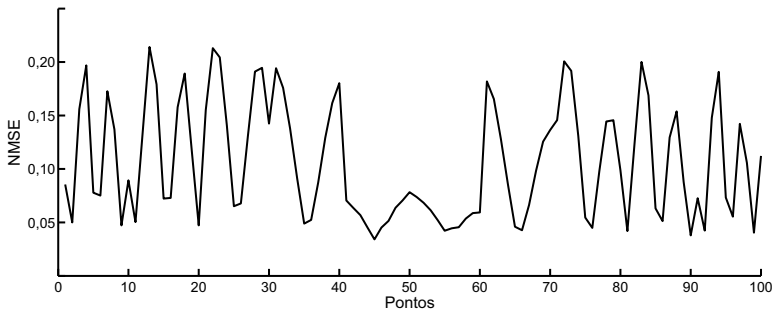


Figura 33: Média dos erros do sistema *fuzzy* considerando cada ponto em todas as execuções

A média geral dos erros, considerando todos os pontos é dada por $\nabla = 0.1359$, e o desvio padrão do erro é dado por $\delta = 0.0537$.

5.1.2.2 Testes com funções triangulares

Os testes com funções triangulares foram executados para as mesmas configurações dos testes com gaussianas, e os resultados são apresentados na Tabela 8 e na Figura 34.

Tabela 8: Valor de avaliação para cada caso de teste

Qtd. Função	Maior Avaliação	Avaliação Média
2	185.2561	180.3458
2 a 4	191.9804	195.3600
3	199.2149	195.0351
5 a 7	288.0078	273.9364
7 a 9	266.6347	258.3207

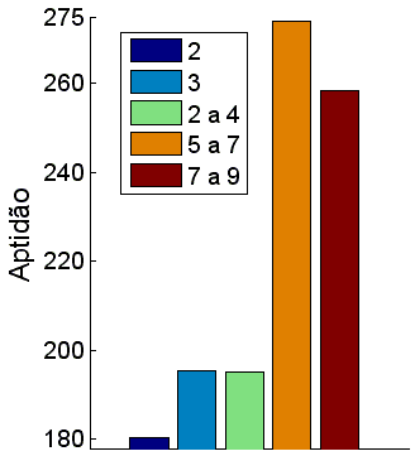


Figura 34: Valor de avaliação para cada caso de teste

Analisando-se as Figuras 34 e 30 pode-se verificar que os testes com funções triangulares demonstram aptidão bem inferior à alcançada pelos testes com Gaussianas. Desta forma, constata-se que para o problema em questão o melhor é utilizar funções Gaussianas.

5.1.3 Função exponencial seno de duas variáveis

A função exponencial seno de duas variáveis $z = e^{xsin(\pi y)}$, cujo gráfico é mostrado na Figura 35, também foi retirada do artigo (RIVAS et al., 2003) para comparação com o algoritmo proposto. É uma função contínua com diferentes pontos de máximo e mínimo.

Os domínios das coordenadas desta função são definidos abaixo.

$$\begin{cases} x \in \mathbb{R} \mid -1 \leq x \leq 1 \\ y \in \mathbb{R} \mid -1 \leq y \leq 1 \\ z \in \mathbb{R} \mid 0.37 \leq z \leq 2.69 \end{cases}$$

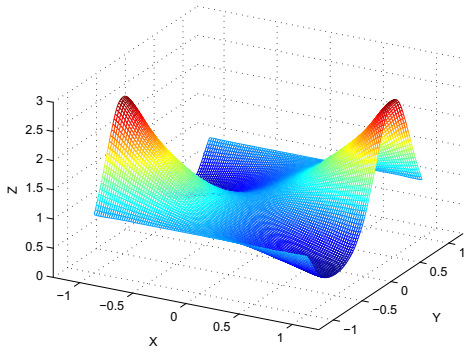


Figura 35: Função exponencial seno utilizada para comparação

As mesmas dificuldades enfrentadas na função seno, são também encontradas nesta. Fazer com que o sistema *fuzzy* encontre os pontos mais próximos possíveis da curva original, de forma a não haver grandes distorções.

Os testes com essa curva foram feitos, hora utilizando funções de pertinência na forma de Gaussianas, hora utilizando funções de pertinência triangulares.

5.1.3.1 Testes com funções Gaussianas

Os testes com funções Gaussianas, neste caso, seguem os mesmos critérios que os testes da função seno de duas variáveis da Seção 5.1.2.1. A Tabela 9 e a Figura 36 mostram os resultados de aptidão para cada caso de teste.

Tabela 9: Avaliação para cada caso de teste

Qtd. Função	Maior Avaliação	Avaliação Média
2	292.3378	269.7578
2 a 4	320.5847	297.5061
3	285.3102	274.8667
5 a 7	471.3641	466.7589
7 a 9	475.6507	440.0464

O desvio padrão da avaliação de todos os resultados é dado por $\delta = 17.7590$, o que demonstra haver uma variação maior do que o caso da função seno, mas ainda assim, em comparação com o valor da média das avaliações, não é uma variação tão grande no valor dos resultados obtidos.

Pela Figura 36 pode-se concluir que testes com quantidade de funções de pertinência variando de cinco a sete e de sete a nove obtiveram, na média, resultados melhores do que com menos funções de pertinência.

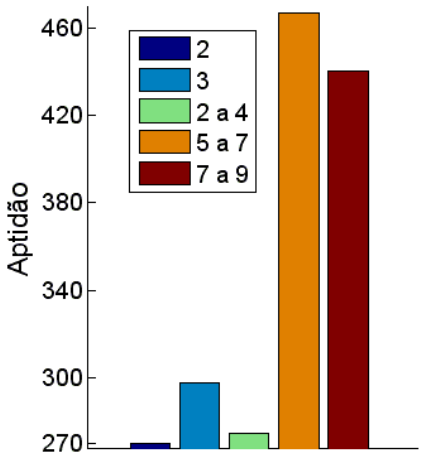


Figura 36: Avaliação para cada caso de teste

As Figuras 5.37(a) a 5.37(d) mostram as curvas resultantes, para o caso da função exponencial, dos testes executados utilizando Gaussianas nas funções de pertinência do sistema *fuzzy*.

Assim como os testes com a função seno, as curvas aqui apresentadas mostram uma tendência de comportamento aproximado ao da curva original, mas também demonstram distorções entre seus pontos.

As quatro figuras mostram claramente que os sistemas *fuzzy* tiveram dificuldade em acompanhar o comportamento da função original.

As diversas outras curvas encontradas durante os testes apresentaram resultados similares às curvas apresentadas acima.

A Figura 38 mostra a configuração das variáveis de entrada e saída de um dos sistemas *fuzzy*. Como foi dito na Seção 5.1.2.1, a numeração destas funções não segue a ordem em que elas aparecem no domínio devido ao fato do objeto Java utilizado no programa não respeitar esta ordem, e para não incorrer em erros decidiu-se mostrar o resultado da mesma forma que aparece no sistema.

Na Seção A.2 do Apêndice A é apresentado o conjunto de regras gerado para este sistema *fuzzy*. Neste caso foram geradas 29 regras, quando o número máximo de regras que poderiam fazer parte deste conjunto de regras, conforme descrito na Seção 4.2.4, é igual a 79. Além disso, o número máximo de regras que podem ser geradas, para este caso, é igual a 711.

A Figura 39 mostra um gráfico onde a média dos erros é calculada para cada ponto utilizado para testar os sistemas *fuzzy* resultado. São os mesmos cem pontos de cada resultado que são utilizados para gerar as curvas de saída. O erro foi normalizado para valores entre zero e um, e o eixo das abscissas simboliza cada um dos pontos.

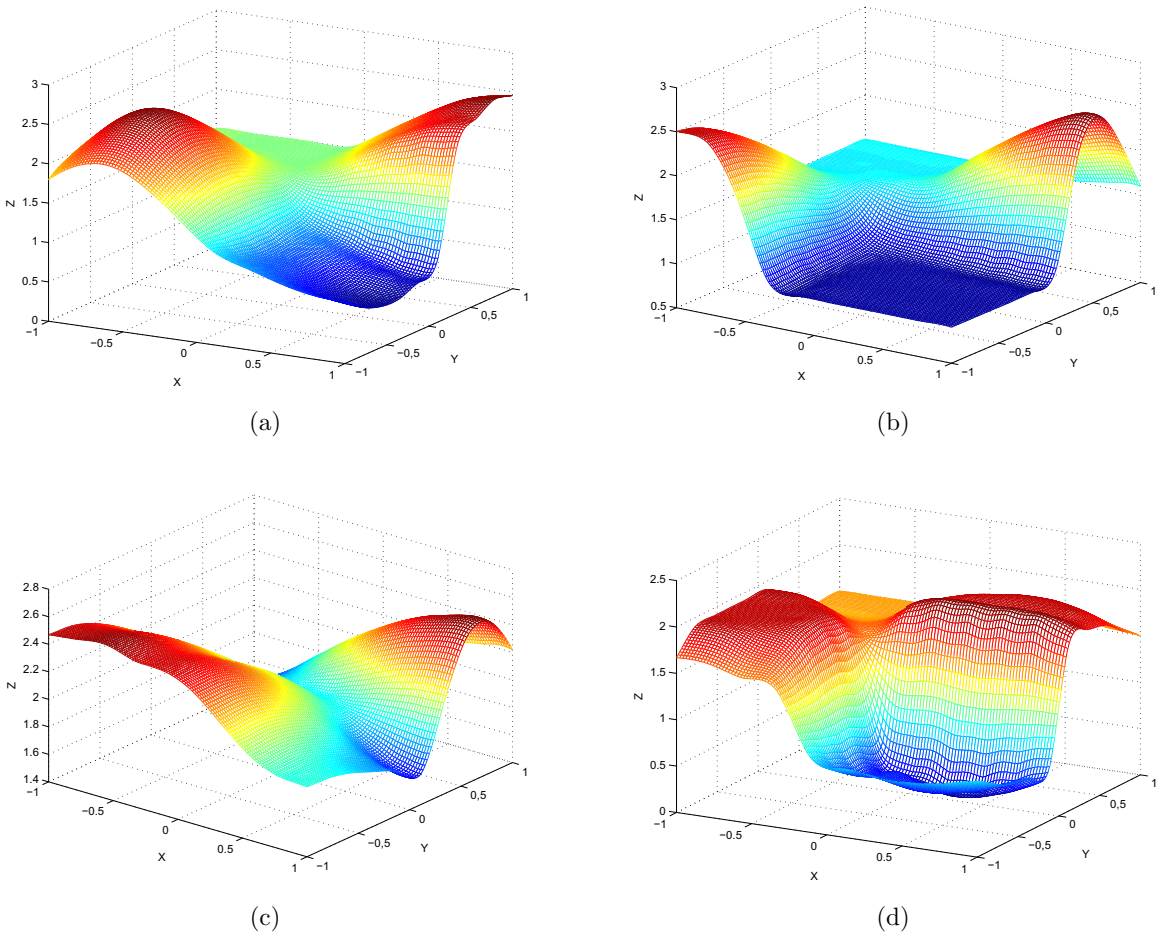


Figura 37: Superfícies resultado de sistemas *fuzzy* função exponencial

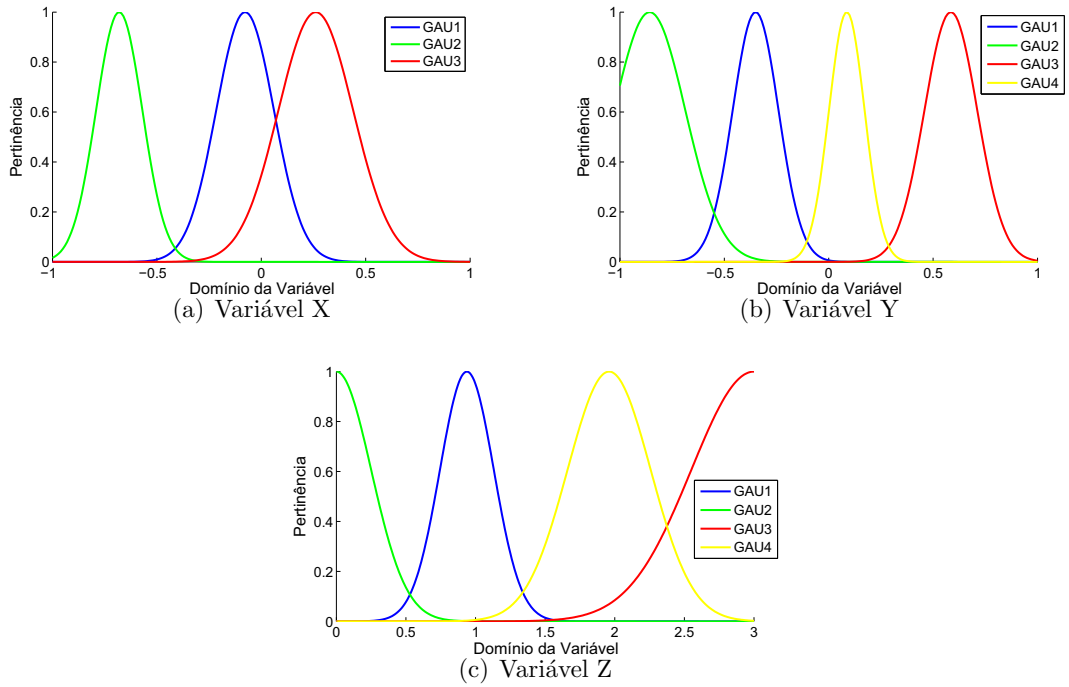
A média geral dos erros, considerando todos os pontos, é dada por $\nabla = 0.2514$, e o desvio padrão do erro é dado por $\delta = 0.1364$.

Comparando este resultado com a função seno na Seção 5.1.2.1, a função exponencial obteve um erro maior e um desvio padrão também maior, o que significa que os testes mostraram um comportamento menos regular neste caso, do que no caso da função seno.

5.1.3.2 Testes com funções triangulares

Os testes com funções triangulares foram executados para as mesmas configurações dos testes com gaussianas, conforme mostram a Tabela 10 e a Figura 40.

Analisando-se as Figuras 40 e 36 pode-se verificar que, confirmando o ocorrido com a função $z = seno(xy)$, os testes com funções triangulares demonstram valores de aptidão bem inferiores aos alcançados pelos testes com Gaussianas. Desta forma, constata-se que para este problema também é melhor utilizar funções Gaussianas.



(a) Variável X

(b) Variável Y

(c) Variável Z

Figura 38: Gráfico de pertinência das variáveis fuzzy para o teste da função exponencial

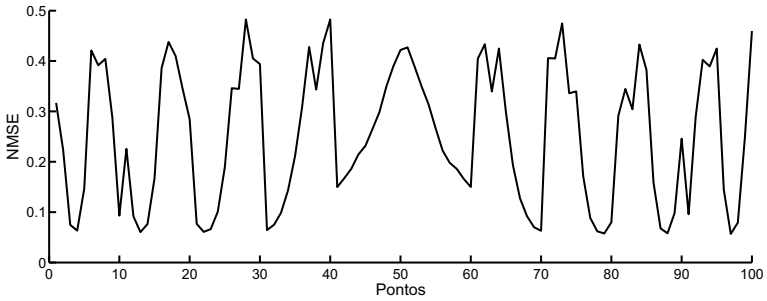


Figura 39: Média dos erros do sistema *fuzzy* considerando cada ponto em todas as execuções

Tabela 10: Valor de avaliação para cada caso de teste

Qtd. Função	Maior Avaliação	Avaliação Média
2	191.0085	188.1245
2 a 4	195.3506	192.0102
3	201.4500	193.6474
5 a 7	294.4974	274.7726
7 a 9	275.9514	265.1596

5.1.3.3 Comparação

O artigo (RIVAS et al., 2003) foi descrito na Seção 3.1 e o resultado relatado é mostrado nesta seção. Neste artigo os autores informam que foram utilizados quatrocentos pontos, com os valores das variáveis X e Y igualmente espaçados. O algoritmo sempre rodou com 320 pon-

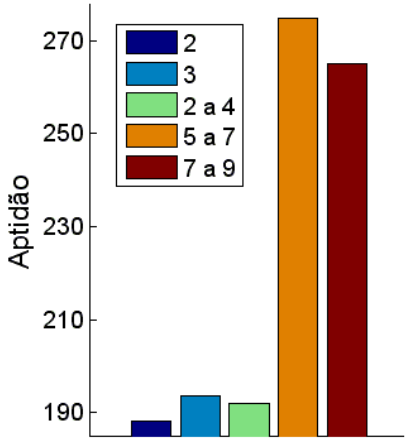


Figura 40: Avaliação para cada caso de teste

tos escolhidos aleatoriamente, e os outros oitenta pontos foram utilizados como conjunto de validação.

A Tabela 11 mostra a média e o desvio padrão do erro da melhor solução encontrada (o melhor indivíduo da última geração), comparando o AG do trabalho citado com o PSO. Para calcular o erro, uma vez que o algoritmo foi finalizado, é apresentado o conjunto de pares de entrada e saída de validação, diferente do que foi utilizado pelo algoritmo, para o melhor sistema *fuzzy*. A média dos erros quadráticos normalizados é computada utilizando as saídas produzidas pelo sistema *fuzzy* e a saída conhecida.

Na Tabela 11 existe ainda uma informação de Tamanho, que em (RIVAS et al., 2003) os autores definem como sendo a multiplicação entre o número de linhas (variável X, Seção 3.1.1) e o de colunas (variável Y, Seção 3.1.1) do resultado do algoritmo, o que equivaleria a multiplicar a quantidade de funções das variáveis de entrada. Este valor é considerado como o tamanho do sistema *fuzzy*, ou melhor, do seu conjunto de regras.

Tabela 11: Comparação entre AG e PSO das funções contínuas

Função	AG			PSO		
	Erro	Desvio	Tamanho	Erro	Desvio	Tamanho
Seno	0.0288	±0.0007	34	0.1359	±0.0537	33.5396
Exponencial	0.0060	±0.0030	41	0.2514	±0.1364	34.4625

Pela tabela, pode-se verificar que, para a função seno a diferença do erro entre o PSO e o GA foi cerca de cinco vezes, além disso o desvio padrão do erro no PSO também mostrou-se mais alto, apresentando assim um comportamento mais irregular entre os testes do que o GA.

Para a função exponencial, pode-se verificar que a diferença do erro entre o PSO e o GA foi consideravelmente alto, assim como o desvio padrão, apresentando estes testes, assim

como o da função seno, um comportamento bem irregular.

O tamanho do conjunto de regras para a função seno ficou bem próximo nos dois algoritmos, PSO e GA, enquanto para a função exponencial, em média, o PSO gerou um conjunto menor do que o GA.

5.1.4 Superfície de Controle de Veículo Subaquático

Um artigo relativo a este trabalho foi submetido para o Congresso Ibero-Latino-Americano de Métodos Computacionais em Engenharia (CILAMCE) no ano de 2009 (COSTA; NEDJAH; MOURELLE, 2009). Este artigo foi resultado de diversos experimentos realizados com a primeira função a ser testada no algoritmo.

Naquele momento o objetivo foi o de testar o algoritmo proposto no sentido de gerar sistemas *fuzzy* que mostrassem como resultado curvas similares a curva utilizada como original, os testes não seguiam ainda uma organização como descrito nos outros experimentos da Seção 5.1. Neste artigo foi utilizada a curva sigmoidal tridimensional mostrada na Figura 5.41(a).

Os domínios utilizados para esta função são mostrados abaixo.

$$\begin{cases} x \in \mathbb{R} \mid -1 \leq x \leq 0.5 \\ y \in \mathbb{R} \mid -1 \leq y \leq 1 \\ z \in \mathbb{R} \mid -1 \leq z \leq 1 \end{cases}$$

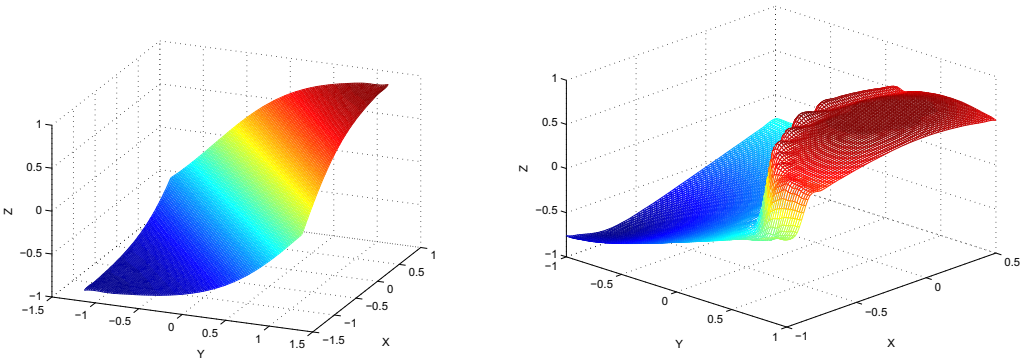
A Equação 63 mostra a função que gera esta curva.

$$z = \frac{2}{1 + e^{-2x-2y}} - 1 \tag{63}$$

Esta é uma superfície utilizada em controle de veículos subaquáticos e foi retirada do artigo (GUO et al., 2007). O melhor resultado para esta curva, que demonstrou menor valor na média dos erros quadráticos, é mostrado na Figura 5.41(b).

Os testes utilizaram os parâmetros definidos na Tabela 12.

Os testes com esta sigmoide mostraram que deixar o algoritmo muito livre, ou seja, com o número de funções membro variando até sete funções para cada variável e o conjunto de regras de uma a cinquenta regras, não era uma boa prática, pois o espaço de busca era tão grande que o algoritmo não conseguiu chegar a resultados muito satisfatórios, demorando muito tempo para evoluir. Em (KRONE; SLAWINSKI, 1998) os autores informam que tentar encontrar o conjunto de regras e as funções de um sistema fuzzy é uma tarefa muito complexa. Por isso foram feitos testes fixando determinada quantidade de funções e regras, o que mostrou um desempenho melhor, em especial quando foi definida a quantidade de uma a quatro regras e duas funções membro para cada variável. Além disso, no caso da curva testada, mostrou-se



(a) Superfície original (b) Melhor resultado

Figura 41: Função sigmoide tridimensional

Tabela 12: Parâmetros do algoritmo obtidos nos experimentos da sigmoide

Parâmetro	Valor
Coeficiente de inércia (w)	0, 0.5 e 1
Coeficiente cognitivo (c_1)	1, 1.1, 1.5, 1.9, 2, 3 e 4
Coeficiente social (c_2)	1, 1.1, 1.5, 1.9, 2, 3 e 4
Quantidade de partículas	10, 20, 40 e 100
Total de iterações	1000, 10000 e 50000
Número mínimo de regras	1, 2 e 4
Número máximo de regras	1, 4, 16 e 50
Número mínimo de funções	1 e 2
Número máximo de funções	2, 4 e 7
Tipo de Função	Vários tipos de função Apenas funções Gaussianas Apenas funções triangulares
Inicialização	Com WM e sem WM
Total Regras WM	0, 2 e 4
Função de avaliação (K_1 – K_5)	0, 0.5 e 1

mais producente utilizar apenas um tipo de função membro, que foi a Gaussiana. E também pôde-se verificar que até próximo de mil iterações o algoritmo apresenta alguma evolução, mas raras vezes apresentou alguma melhora depois de mil iterações.

Estes preceitos obtidos desta prévia experiência foram úteis e se fizeram presentes ao executar os testes posteriores, que foram expostos nesta seção. Nos novos testes procurou-se sempre reduzir o intervalo de variação, tanto do número de regras, quanto do número de funções de pertinência de forma a auxiliar o algoritmo na busca.

5.1.4.1 Testes com funções Gaussianas

Novos testes foram realizados com a superfície em questão de forma mais organizada para possibilitar a elaboração da Tabela 13 e do gráfico da Figura 42 com seus resultados.

Tabela 13: Valor de avaliação para cada caso de teste da sigmoide

Qtd. Função	Maior Avaliação	Avaliação Média
2	298.0712	285.6008
2 a 4	388.4907	379.4781
3	381.4580	375.9850
5 a 7	404.1842	402.5031
7 a 9	405.4200	404.7341

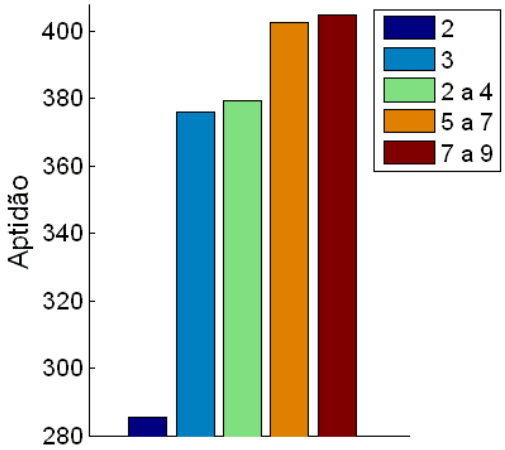


Figura 42: Avaliação para cada caso de teste da sigmoide

O desvio padrão da avaliação de todos os resultados é dado por $\delta = 10.5340$, que demonstrou ser menor do que os dois outros casos de função contínua. Os resultados para este problema não foram muito melhores do que aqueles apresentados no artigo (COSTA; NEDJAH; MOURELLE, 2009).

5.2 Resultados de testes com funções discretas

Nesta seção serão apresentados os testes do algoritmo desenvolvido quando aplicado a funções discretas. Foram selecionados dois problemas para os quais havia uma base com dados para os testes, além de um número de variáveis não muito grande para que fosse possível realizar uma boa quantidade de experimentos.

5.2.1 Classificação da flor Iris

Para testes com funções discretas podem ser utilizados problemas de classificação. A classificação da flor Iris, por exemplo, foi um problema tratado no artigo (CINTRA; CAMARGO, 2007b), e relatado na Seção 3.2 deste trabalho.

Este problema, assim como os domínios das variáveis foram obtidos do repositório *UCI Machine Learning* (MERZ; MURPHY, 1998). Os autores do artigo (CINTRA; CAMARGO, 2007b) informam que escolheram este problema pelo fato de seus atributos serem numéricos, o que também é importante no trabalho em questão.

O repositório contém 150 instâncias para o problema, que possui cinco atributos, ou seja cinco variáveis, sendo quatro de entrada e uma de saída. As quatro variáveis de entrada são valores reais, enquanto a variável de saída define o nome da classe. A tabela 14 mostra as variáveis de entrada e seus domínios.

Tabela 14: Variáveis de entrada do problema da Iris

Nome da Variável	Domínio Mínimo	Domínio Máximo
Comprimento da Sépala (CS)	4.3	7.9
Largura da Sépala (LS)	2.0	4.4
Comprimento da Pétala (CP)	1.0	6.9
Largura da Pétala (LP)	0.1	2.5

A variável de saída define a classe (CL) de uma determinada entrada em uma das três classificações descritas na Tabela 15.

Tabela 15: Classes da flor Iris nestes experimentos

Nome da Classe	Código
Iris-setosa	1.0
Iris-versicolor	2.0
Iris-virginica	3.0

O campo código na Tabela 15 é o valor utilizado para identificar a classe após a inferência e defuzificação do sistema *fuzzy* já que esta saída é numérica. Desta forma, se a saída do sistema *fuzzy* for um, ou bem próximo deste valor, a classificação da Iris é Iris-setosa, e a mesma lógica é aplicada às outras classes. Este valor, na verdade, é para que se possa definir o domínio da variável de saída, que neste caso varia de um a três.

5.2.1.1 Testes com funções Gaussianas

Estes testes, diferente dos realizados para as funções contínuas, seguiram o padrão definido no artigo (CINTRA; CAMARGO, 2007b) para facilitar a comparação. Assim, foram executados

definindo para os sistemas *fuzzy* as configuração de três, quatro e sete funções de pertinência para as variáveis de entrada. Foram realizados então três experimentos, cada um com uma das configurações.

O conjunto de testes possui 150 instâncias, sendo 105 utilizadas no processamento do algoritmo e 45 no final para testar o melhor sistema *fuzzy* encontrado.

Os parâmetros do PSO utilizados aqui, foram os mesmos definidos para as funções contínuas. Os testes com a função iris demonstraram não haver necessidade de alteração destes parâmetros, apenas o número de iterações passou para mil, pois o algoritmo apresentou bons resultados. A Tabela 16 e a Figura 43 mostram resultados para cada caso de teste do problema da flor Iris.

Tabela 16: Valores de avaliação para cada caso de teste da Iris

Qtd. Função	Maior Avaliação	Avaliação Média
3	482.7007	452.0899
5	501.7773	456.7786
7	476.0999	451,1822

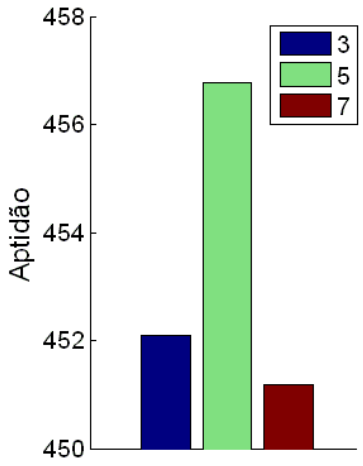


Figura 43: Avaliação para cada caso de teste da Iris

O desvio padrão da avaliação de todos os resultados é dado por $\delta = 0.4952$, o que demonstra não haver uma variação significativa nos resultados obtidos.

Pela Figura 43 pode-se concluir que os testes com a quantidade de funções igual a cinco, obtiveram resultados melhores do que os testes com as outras configurações.

Para executar a validação pelo sistema *fuzzy* resultado, foram utilizadas 45 instâncias, quinze para cada classe da flor Iris, que são três. A saída destes testes possui então a forma de um vetor e sua configuração é mostrada nas Tabelas 17 e 18, que exemplificam com alguns resultados encontrados. A saída original é mostrada na primeira coluna, e nos outros campos

são apresentados o resultado e o erro para cada tipo de teste, com três, cinco e sete funções de pertinência (μf). A Tabela 17 possui os primeiros quinze elementos dos vetores e a Tabela 18 os trinta restantes.

Tabela 17: Primeira parte de exemplos de resultado para cada caso de teste da Iris

Saída	3 μf		5 μf		7 μf	
Original	Resultado	Erros	Resultado	Erros	Resultado	Erros
1	1.088	0.088	1.185	0.185	1.034	0.035
1	1.080	0.080	1.001	0.001	1.153	0.154
1	1.070	0.070	1.048	0.048	1.063	0.063
1	1.080	0.080	1.028	0.028	1.083	0.083
1	1.070	0.070	1.145	0.145	1.138	0.138
1	1.080	0.080	1.105	0.105	1.052	0.052
1	1.080	0.080	1.087	0.087	1.056	0.056
1	1.080	0.080	1.171	0.171	1.101	0.101
1	1.068	0.068	1.002	0.002	1.154	0.154
1	1.068	0.068	1.245	0.245	1.107	0.107
1	1.074	0.074	1.109	0.109	1.142	0.142
1	1.068	0.068	1.091	0.091	1.150	0.150
1	1.074	0.074	1.095	0.095	1.146	0.146
1	1.070	0.070	1.120	0.120	1.022	0.022
1	1.074	0.074	1.134	0.134	1.043	0.043

Os testes exibidos nas Tabelas 17 e 18 foram escolhidos entre aqueles de melhor expressão, apenas para exemplo. Por esta tabela pode-se perceber que dificilmente o erro chega a 0.25, ou seja, 25%.

A Figura 44 mostra o padrão das funções de pertinência das variáveis de entrada e saída de um sistema *fuzzy* que gera saída para um caso de teste com três funções de pertinência.

Na Seção A.3 do Apêndice A é apresentado o conjunto de regras gerado para o sistema *fuzzy* com a configuração de variáveis indicada na Figura 44.

A Tabela 19 e a Figura 45 mostram uma comparação da média de quantidade de regras encontradas nos sistemas *fuzzy* resultantes dos experimentos com o PSO. O campo Total na Tabela 19 indica o total de regras possíveis de acordo com o trabalho citado.

A Tabela 20 e a Figura 46 informam a taxa de classificação para cada experimento, o desvio padrão desta taxa e a média de erro na classificação.

A média de erro indica o quanto os sistemas *fuzzy* erram ao definirem a que classe pertence um conjunto de informações. Como exemplo, supõe-se que uma determinada configuração de valores de entrada deva ser classificada como classe um, com um erro de 20% nesta classificação o valor poderia ser de 1.20 ou 0.80 ao invés de exatamente um.

Tabela 18: Segunda parte de exemplos de resultado para cada caso de teste da Iris

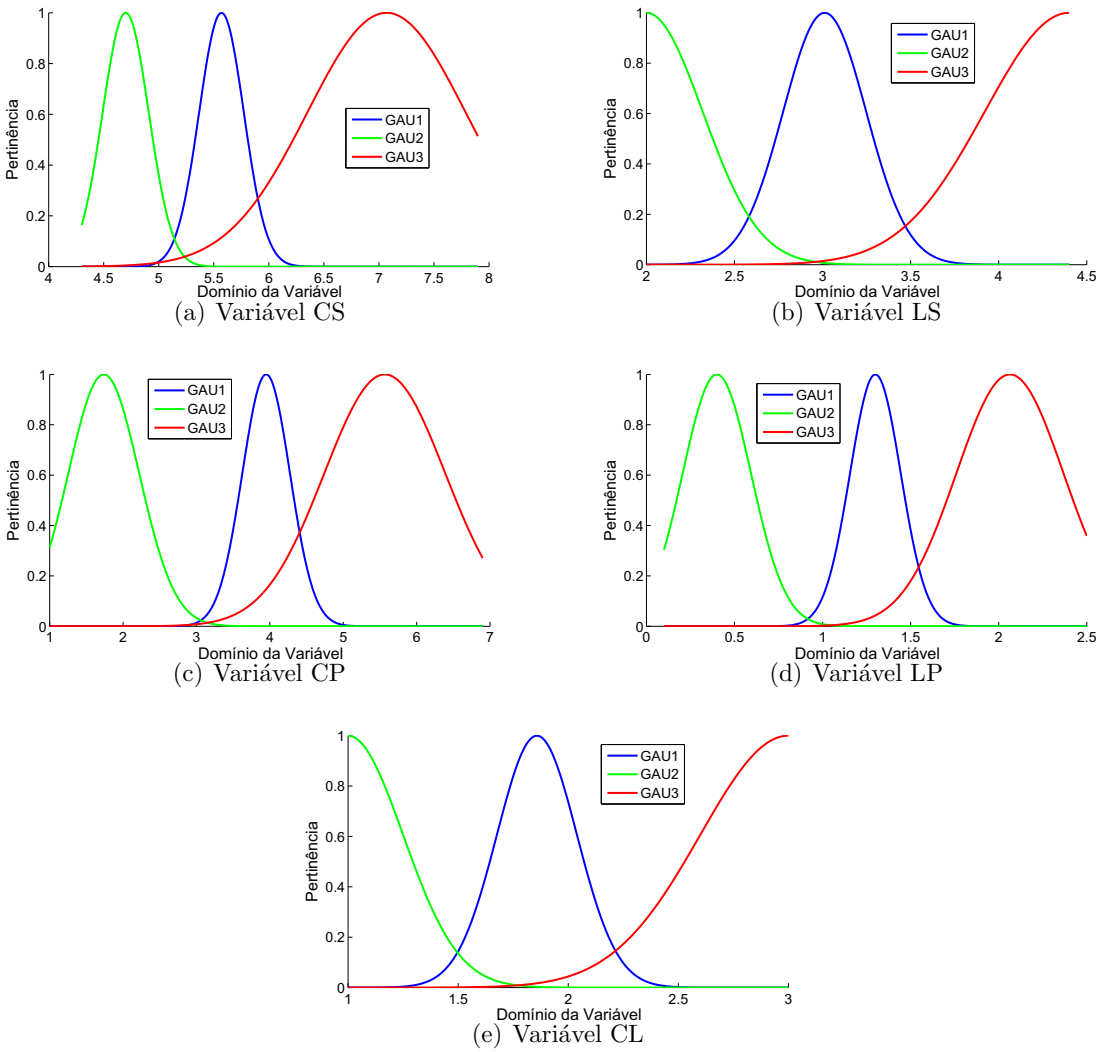
Saída	3 μf		5 μf		7 μf	
Original	Resultado	Erros	Resultado	Erros	Resultado	Erros
2	2.186	0.186	2.000	0.000	1.980	0.020
2	2.148	0.148	2.000	0.000	1.886	0.114
2	2.034	0.034	2.000	0.000	2.000	0.000
2	2.018	0.018	2.000	0.000	2.000	0.000
2	2.018	0.018	2.000	0.000	2.000	0.000
2	2.016	0.016	2.000	0.000	2.000	0.000
2	2.182	0.182	2.000	0.000	2.000	0.000
2	2.006	0.006	2.000	0.000	2.000	0.000
2	1.998	0.002	2.006	0.006	2.000	0.000
2	2.024	0.024	2.000	0.000	2.000	0.000
2	2.008	0.008	2.000	0.000	2.000	0.000
2	2.022	0.022	2.000	0.000	2.000	0.000
2	2.028	0.028	2.000	0.000	2.000	0.000
2	1.862	0.138	2.198	0.198	2.000	0.000
2	2.020	0.020	2.000	0.000	2.000	0.000
3	2.896	0.104	3.010	0.010	3.081	0.081
3	2.854	0.146	3.176	0.176	3.019	0.019
3	2.884	0.116	3.150	0.150	3.218	0.218
3	2.868	0.132	3.000	0.000	3.068	0.068
3	2.928	0.072	3.126	0.126	3.230	0.230
3	2.854	0.146	3.176	0.176	3.215	0.215
3	2.894	0.106	3.052	0.052	3.015	0.015
3	2.916	0.084	3.054	0.054	3.085	0.085
3	2.896	0.104	3.050	0.050	3.006	0.006
3	2.806	0.194	3.200	0.200	3.211	0.211
3	2.894	0.106	3.076	0.076	3.119	0.119
3	2.914	0.086	3.028	0.028	3.061	0.061
3	2.928	0.072	3.076	0.076	3.032	0.032
3	2.896	0.104	3.126	0.126	3.091	0.091
3	2.882	0.118	3.054	0.054	3.170	0.170

Tabela 19: Quantidade de regras para cada experimento da Iris

Qtd. Funções	Total	Média
3	243	10.01
5	1875	16.36
7	7203	41.16

Tabela 20: Taxa de classificação, desvio e erro para cada experimento da Iris

Qtd. Funções	Taxa de Classificação (%)	Desvio da Classificação	Erro (%)
3	99.85	0.0058	12.50
5	100.00	0.0000	11.93
7	97.81	0.0497	13.11



(a) Variável CS (b) Variável LS

(c) Variável CP (d) Variável LP

(e) Variável CL

Figura 44: Gráfico das variáveis fuzzy para o teste do problema da Iris

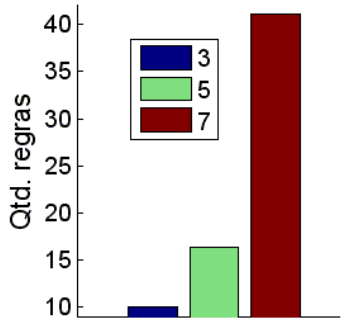
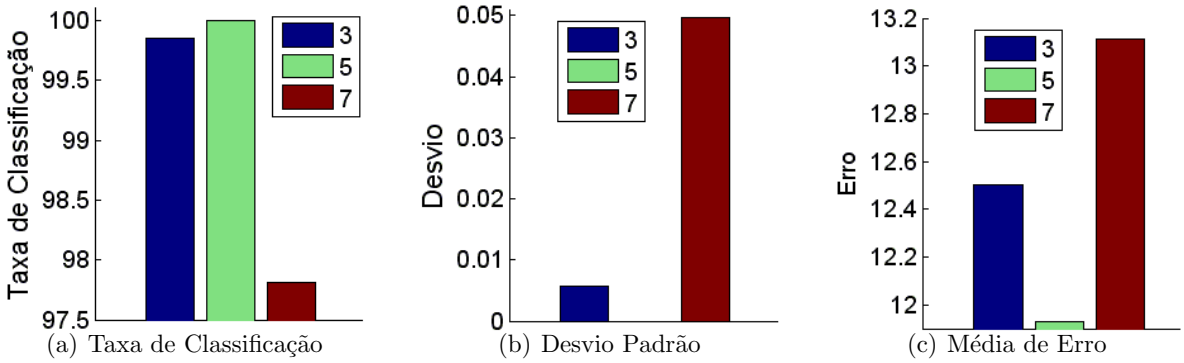


Figura 45: Quantidade de regras para cada experimento da Iris

De forma errônea pode ser considerado que com um erro menor do que 50% é possível classificar um determinado elemento, ou seja, se o erro do exemplo acima fosse de 49%, um possível valor de saída seria 1.49, e a instância seria classificada como pertencente à classe um.



(a) Taxa de Classificação (b) Desvio Padrão (c) Média de Erro

Figura 46: Taxa de classificação, desvio e erro para cada experimento da Iris

Mas é fácil perceber que um erro muito próximo de 50% torna essa classificação duvidosa. A média de erro dos experimentos, mostra que os sistemas *fuzzy* gerados não deixam dúvidas na classificação das instâncias utilizadas, pois o erro apresentado é bem inferior a 50%. Os exemplos de resultados expostos nas Tabelas 17 e 18 ilustram bem este fato, pois nenhum erro chegou a 25%.

Comparando os resultado entre as três quantidades de função de pertinência, percebe-se que foram encontrados sistemas *fuzzy* adequados para tal problema em todos os casos, mas com cinco funções chega-se a sistemas *fuzzy* mais adequados ao problema com mais frequência.

5.2.1.2 Comparação

Os resultados informados no artigo (CINTRA; CAMARGO, 2007b), descritos na Seção 3.2, são apresentados nas Figuras 47 e 48, junto com os resultados deste trabalho, o PSO. Além disso, mantendo a íntegra das informações declaradas pelos autores do artigo citado, a tabela contém informações sobre resultados referentes ao método WM.

A Tabela 21 e a Figura 47 mostram a comparação do tamanho do conjunto de regras gerado para os sistemas *fuzzy* em cada algoritmo.

Tabela 21: Comparação do conjunto de regras gerado pelos diversos algoritmos

Qtd. Funções	PSO	GA I	GA II	WM
3	10.01	7.60	13.80	15
5	16.36	15	41.2	44.8
7	41.16	54.4	61.4	67.2

Pela Figura 47 pode ser verificado que, para o experimento com três funções, o PSO apresentou um desempenho um pouco inferior ao GA I na definição do conjunto de regras, mas foi melhor que o GA II e o WM.

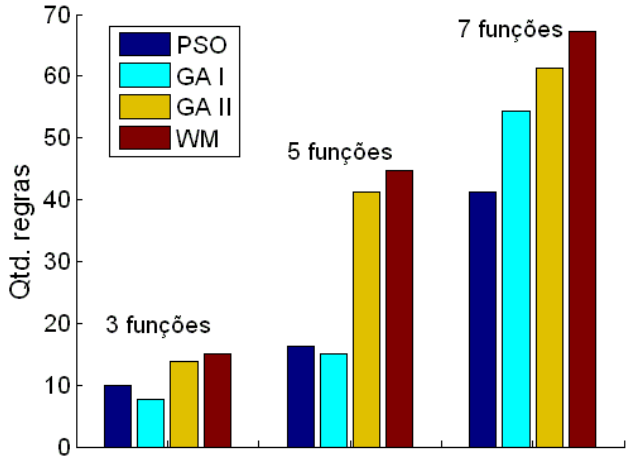


Figura 47: Comparação do conjunto de regras gerado pelos diversos algoritmos

Para o experimento com cinco funções, o PSO se saiu ainda melhor na definição do conjunto de regras, pois apesar de ainda ter gerado um pouco mais de regras do que o GA I, gerou um conjunto bem menor do que o do GA II e do WM.

Para o experimento com sete funções, o PSO desenvolveu menos regras que todos os outros algoritmos, em compensação quando analisada a taxa de classificação na Figura 48, ele ficou bem abaixo do GA I, apesar de ter sido melhor do que os outros. A Tabela 22 e a Figura 48 mostram a comparação entre as taxas de classificação.

Tabela 22: Comparação da taxa de classificação entre os diversos algoritmos

Qtd. Funções	PSO	GA I	GA II	WM
3	99.85	99.71	98.70	100.00
5	100.00	100.00	100.00	100.00
7	97.81	98.31	97.30	94.70
Média	99.22	99.34	98.67	98.23

Pela Figura 48 pode ser verificado que, para o experimento com três funções, novamente o PSO obteve um bom desempenho, com um desempenho acima dos outros algoritmos.

Para o experimento com cinco funções, todos os algoritmos, inclusive o PSO obtiveram taxa de 100%.

Em compensação para o experimento com sete funções, o PSO obteve um rendimento consideravelmente menor do que o GA I, apesar de manter-se melhor do que o GA II e o WM.

A Figura 48 mostra que, na média, o PSO se saiu melhor do que o GA II e o WM, perdendo apenas para o GA I, ainda assim por uma diferença não muito significativa.

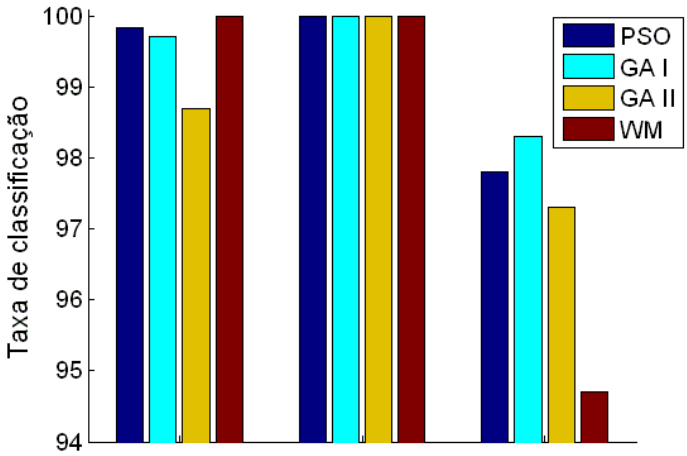


Figura 48: Comparação da taxa de classificação entre os diversos algoritmos

5.2.2 Avaliação de Carros

Um outro problema encontrado no repositório *UCI Machine Learning* (MERZ; MURPHY, 1998) foi o de avaliação de carros, no qual o objetivo é avaliar se um veículo é bom ou não para ser adquirido. Este é também um problema de classificação.

O repositório conta com 1728 instâncias para o problema que possui sete atributos, sendo seis de entrada e um de saída. A Tabela 23 mostra os atributos (variáveis) do problema e seus possíveis valores.

Tabela 23: Variáveis do problema de avaliação de carros

Variável	Valores
Valor de Compra (VC)	muito alto, alto, médio, baixo
Manutenção (MN)	muito alto, alto, médio, baixo
Qtd. de Portas (PO)	2, 3, 4, 5 ou mais
Qtd. de Pessoas (PE)	2, 4, mais
Porta Malas (PM)	pequeno, médio, grande
Nível de Segurança (NS)	baixo, médio, alto
Classe (CL)	inaceitável, aceitável, bom, muito bom

Para o algoritmo poder trabalhar com este problema, faz-se necessário que os valores das variáveis sejam alterados para valores reais, por isso a Tabela 24 mostra as variáveis, com seus domínios reais. Assim pode-se definir o domínio das variáveis. Cada valor de variável na Tabela 23 corresponde a um valor inteiro, dentro do domínio definido, na Tabela 24.

Para um conjunto de dados em que à variável *Valor de compra*, por exemplo, for associado o valor *muito alto* será utilizado o valor um no algoritmo, para *alto* será utilizado valor dois, para *médio* o valor três e para *baixo* o valor quatro. Estes valores são apenas para o algo-

ritmo poder executar de forma padronizada, ou seja, sempre recebendo como entrada valores reais. Em termos de função de pertinência, a cada valor de variável na Tabela 23 corresponde uma função de pertinência que deverá ser definida pelo PSO.

Tabela 24: Domínio das variáveis do problema de avaliação de carros

Variável	Mínimo Domínio	Máximo Domínio
Valor de compra (VC)	1	4
Manutenção (MN)	1	4
Qtd. de portas (PO)	2	5
Qtd. de pessoas (PE)	2	5
Porta malas (PM)	1	3
Nível de segurança (NS)	1	3
Classe (CL)	1	4

Neste caso a variável de saída é a Classe, e sua classificação dá-se pelo valor numérico definido na Tabela 24, de um a quatro, que simboliza as classes definidas na Tabela 23, similar ao caso da flor Iris.

5.2.2.1 Testes com funções Gaussianas

Como no problema da avaliação de veículo as variáveis possuem um conjunto de valores possíveis pré-definidos, este teste possui características um pouco diferente do problema da flor Iris, que são quantidades fixas de funções de pertinência para cada variável, como mostra a Tabela 25.

Tabela 25: Quantidade de funções de pertinência por variável

Variável	Qtd. Funções
Valor de compra (VC)	4
Manutenção (MN)	4
Qtd. de portas (PO)	4
Qtd. de pessoas (PE)	3
Porta malas (PM)	3
Nível de segurança (NS)	3
Classe (CL)	4

O conjunto de testes selecionado possui 200 instâncias, sendo 140 utilizadas no processamento do algoritmo e 60 no final para testar o melhor sistema *fuzzy* encontrado.

Os parâmetros do PSO utilizados aqui, foram os mesmos definidos para as funções contínuas, e também utilizados no problema da flor Iris.

A maior avaliação conseguida nestes testes foi 450.2399, sendo a média das avaliações igual a 427.4825.

O desvio padrão da avaliação de todos os resultados é dado por $\delta = 0.2107$, o que demonstra também não haver uma variação muito significativa no valor de avaliação dos resultados obtidos neste problema.

Para executar a validação pelo sistema *fuzzy* resultado, foram utilizadas 60 instâncias, como informado anteriormente, quinze para cada classe de avaliação de carro, que são quatro. A saída destes testes também possui a forma de um vetor e sua configuração é mostrada na Tabela 26, que exemplifica com um dos resultados encontrados. A saída original é mostrada na primeira coluna, seguida do resultado e do erro.

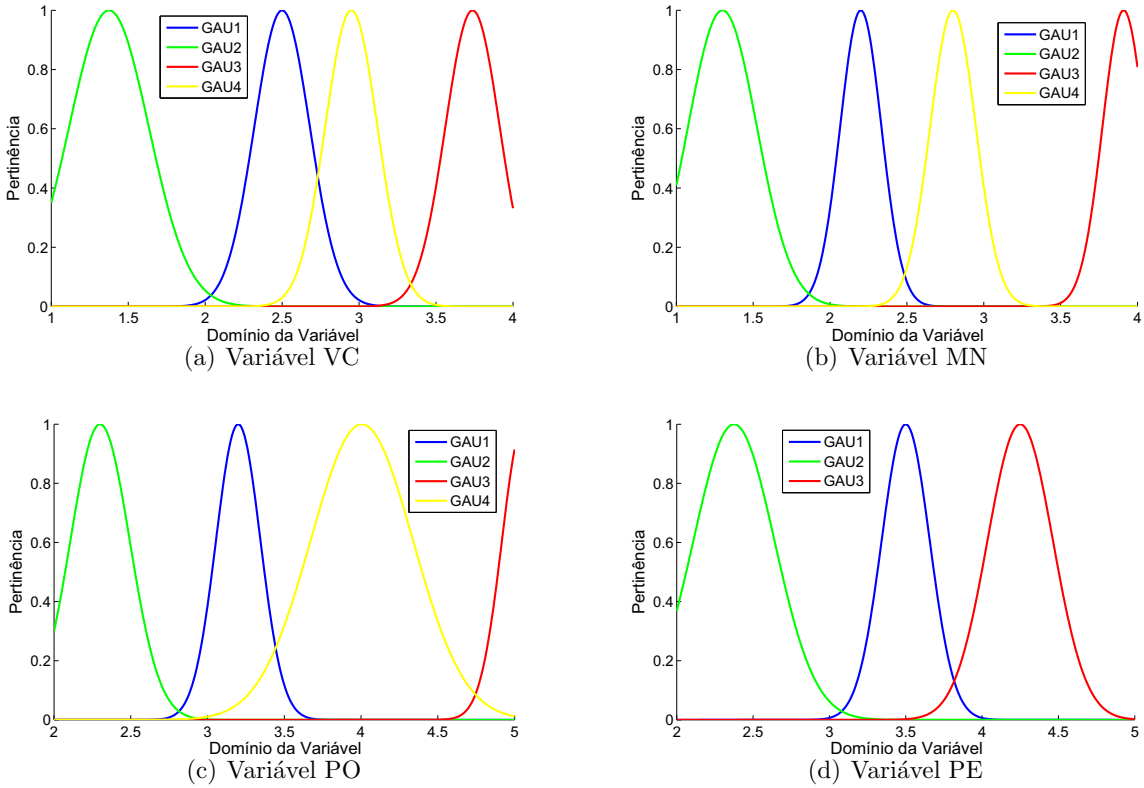
Tabela 26: Exemplo de resultado para o problema de avaliação de carro

Saída Original	Resultado	Erro	Saída Original Continuação	Resultado	Erro
1	1.1037	0.1037	3	2.8080	0.1920
1	1.0437	0.0437	3	3.0000	0.0000
1	1.1185	0.1185	3	3.0000	0.0000
1	1.1935	0.1935	3	3.0000	0.0000
1	1.1174	0.1174	3	3.0000	0.0000
1	1.0087	0.0087	3	3.0000	0.0000
1	1.0175	0.0175	3	2.9670	0.0330
1	1.1771	0.1771	3	3.0000	0.0000
1	1.1875	0.1875	3	2.8080	0.1920
1	1.0476	0.0476	3	3.0000	0.0000
1	1.2154	0.2154	3	3.0000	0.0000
1	1.2024	0.2024	3	3.0000	0.0000
1	1.0672	0.0672	3	3.0000	0.0000
1	1.1041	0.1041	3	3.0000	0.0000
1	1.0216	0.0216	3	3.0000	0.0000
2	2.0000	0.0000	4	4.1693	0.1693
2	2.0000	0.0000	4	4.0724	0.0724
2	2.0000	0.0000	4	4.1837	0.1837
2	2.0000	0.0000	4	4.0602	0.0602
2	2.0330	0.0330	4	4.0336	0.0336
2	2.0000	0.0000	4	4.1011	0.1011
2	2.0000	0.0000	4	4.0450	0.0450
2	2.0000	0.0000	4	4.1959	0.1959
2	2.0330	0.0330	4	4.0297	0.0297
2	2.0000	0.0000	4	4.2159	0.2159
2	2.0000	0.0000	4	4.2040	0.2040
2	2.0000	0.0000	4	4.0219	0.0219
2	2.0000	0.0000	4	4.2057	0.2057
2	2.0000	0.0000	4	4.0380	0.0380
2	2.0000	0.0000	4	4.2497	0.2497

O teste exibido na Tabela 26 foi escolhido entre aqueles de melhor expressão, apenas para exemplo. Por esta tabela pode-se perceber, que assim como o caso da flor Iris, dificilmente

o erro chega a 0.25, ou seja, 25%, garantindo uma classificação precisa.

As Figuras 49 e 50 mostram a configuração das variáveis de entrada e saída de um sistema *fuzzy* encontrado pelo algoritmo.



(a) Variável VC (b) Variável MN

(c) Variável PO (d) Variável PE

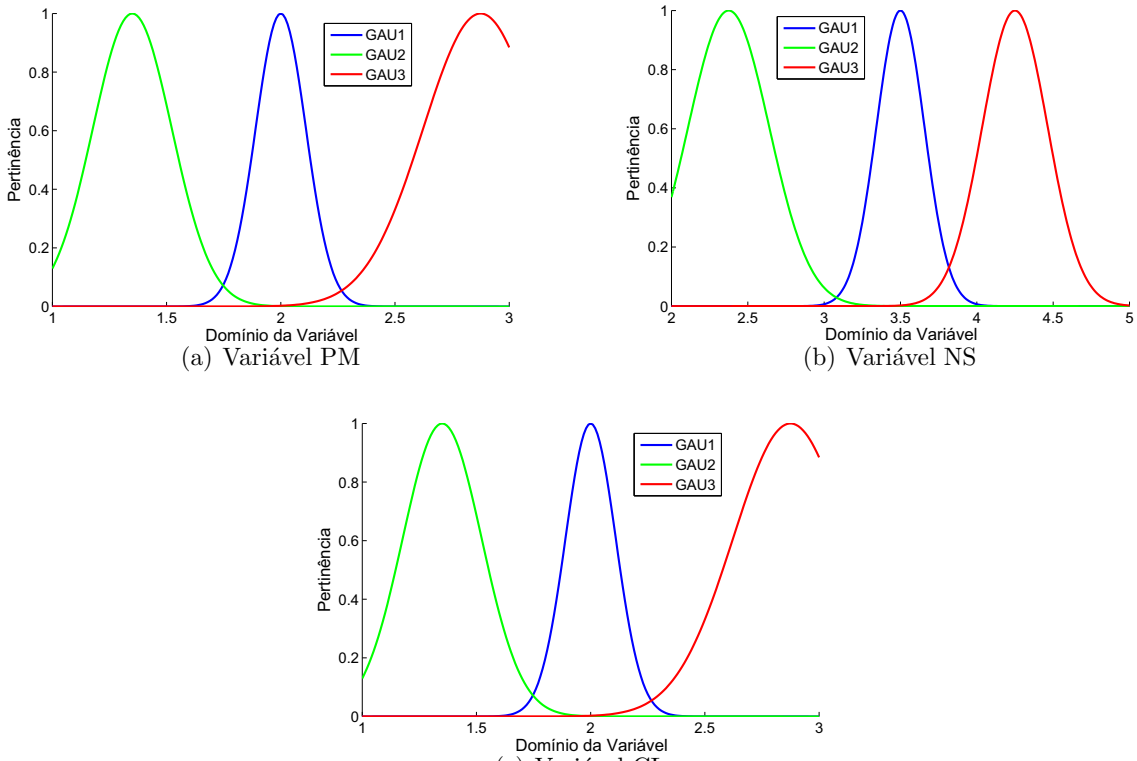
Figura 49: Pertinência das variáveis fuzzy para o problema de avaliação de veículo A

Na Seção A.4 do Apêndice A é apresentado o conjunto de regras gerado para o sistema *fuzzy* com a configuração de variáveis indicada nas Figuras 49 e 50.

O número de regras que podem fazer parte do conjunto de regras para a configuração do problema é de 7999, calculado conforme explicado na Seção 4.2.4. A média de regras geradas nos sistemas *fuzzy* nos diversos testes foi de 80.20.

A média da taxa de classificação encontrada nos testes foi de 98.76, sendo o desvio de 0.0272 e a média de erro da classificação igual a 12.97.

A média de erro, conforme foi explicado na Seção 5.2.1.1, indica o quanto os sistemas *fuzzy* erram ao definirem a que classe pertence um conjunto de informações. Da mesma forma que os testes com o problema da flor Iris, a média de erro dos experimentos do problema do carro, mostra que os sistemas *fuzzy* gerados não deixam dúvidas na classificação das instâncias utilizadas, pois o erro apresentado é bem inferior a 50%.



(a) Variável PM

(b) Variável NS

(c) Variável CL

Figura 50: Pertinência das variáveis fuzzy para o problema de avaliação de veículo B

5.3 Considerações Finais do Capítulo

Neste capítulo foram descritos os testes realizados e os resultados encontrados para o algoritmo desenvolvido. Os problemas foram apresentados, efetuando-se comparações nos casos em que eles foram obtidos de outros trabalhos. Estes testes foram desmembrados em dois grupos, funções contínuas e funções discretas, a fim de apresentar resultados para os dois casos. Como função contínua foram utilizadas superfícies tridimensionais, e para funções discretas problemas de classificação com diversas variáveis.

Capítulo 6

CONCLUSÕES E TRABALHOS FUTUROS

PELOS resultados descritos no Capítulo 5 pode-se verificar que para os problemas discretos, de classificação, o algoritmo conseguiu encontrar sistemas *fuzzy* satisfatórios, mas que para problemas contínuos, sistemas com três variáveis, houve dificuldades em se chegar a bons resultados. Mesmo nos resultados discretos, foi observado o fato dos sistemas *fuzzy* gerados apresentarem um erro na classificação dos elementos, que em geral ficaram abaixo de 25%, o que permite classificar os elementos sem problemas.

O erro relatado, no caso de funções discretas, não possui uma influência tão grande no resultado desejado, enquanto o erro nas funções contínuas transforma completamente a função em outra, o que não é desejável. Destes relatos conclui-se que neste momento o algoritmo poderia ser utilizado para funções discretas em problemas de classificação, mas ainda não poderia ser empregado em caso de funções contínuas.

Testes iniciais mostraram que deixar o algoritmo muito livre, ou seja, com o número de funções membro variando até sete funções para cada variável e o conjunto de regras de uma a cinquenta regras, não era uma boa prática, pois o espaço de busca era tão grande que o algoritmo não conseguiu chegar a resultados muito satisfatórios (COSTA; NEDJAH; MOURELLE, 2009). Tentar encontrar o conjunto de regras e as funções de um sistema fuzzy é uma tarefa muito custosa (KRONE; SLAWINSKI, 1998). Por isso, os testes realizados neste trabalho fixaram determinada quantidade de funções e regras, o que mostrou um desempenho melhor. Além disso, no caso dos problemas testados, principalmente os contínuos, mostrou-se mais producente utilizar apenas um tipo de função membro, que foi a Gaussiana.

A dificuldade em se chegar a um modelo "perfeito", tanto no caso contínuo, no qual se mostra mais acentuado o problema, quanto no caso discreto, deve-se, além de outras coisas, pela dimensão do espaço de busca de cada objetivo, neste que é um problema a multiobje-

tivo: encontrar o conjunto de regras que melhor se adeque, e as funções membro com melhor dimensionamento ao problema e que apresente uma acurácia aceitável, além de considerar a interpretabilidade. E também pela quantidade de parâmetros do problema tanto os do PSO, quanto os do *fuzzy* (limitação do conjunto de regras e funções), e os parâmetros internos como: mínimo e máximo de regras e funções a serem alteradas, total de regras do WM, inicialização aleatória ou por WM, ou utilizando partições etc.

O PSO é baseado em um determinismo que é o fato de uma partícula saber se movimentar na direção daquela que está mais próxima da melhor solução, mas no caso em questão nem sempre ao se criar, remover ou alterar nova regra, ou função de pertinência, o sistema *fuzzy* gerado irá necessariamente apresentar resultado mais próximo do encontrado pela partícula com maior aptidão. Desta forma, o algoritmo também sofre com o não determinismo destas alterações.

Outra conclusão que se pode obter do que foi descrito é que a dificuldade no modelo deve-se à grande quantidade de parâmetros e ao fato do não determinismo da alteração de um sistema *fuzzy* (COSTA; NEDJAH; MOURELLE, 2010a) (COSTA; NEDJAH; MOURELLE, 2010b), pois no PSO o movimento da partícula é em direção ao indivíduo mais apto. Por isso, e por conta das comparações com AGs, talvez um AG seja mais adequado para este tipo de tarefa, pelo fato das regras estarem no genótipo do indivíduo. Apesar de, nas comparações realizadas com outros trabalhos, o fato dos autores terem limitado a um tipo de problema, ter reduzido o espaço de busca.

Algumas modificações poderiam ser realizadas no algoritmo na tentativa de melhorar seu desempenho. Uma delas é utilizar o WM, no caso de funções contínuas, para gerar as regras durante o processamento do algoritmo, não apenas na inicialização, desta forma talvez o algoritmo consiga encontrar melhores conjuntos de regras em um tempo mais curto, mas necessitaria continuar possibilitando a definição de regras que não são alcançadas pelo WM, como aquelas que não utilizam todas as variáveis do problema no antecedente.

Neste projeto foi utilizado o modelo global do PSO, a utilização do modelo local pode vir a apresentar resultados diferentes dos que foram expostos no Capítulo 5.

Na inicialização das funções de pertinência foi definido que o valor mínimo do domínio do suporte de uma função de pertinência deveria estar entre o mínimo e o máximo da função de pertinência anterior. Mas para não tolher o algoritmo essa restrição não se mantém durante o processamento, sendo este controle obtido pelos critérios de distinguibilidade e completude. A utilização desta restrição também durante o processamento do algoritmo pode ser

que traga mais benefícios. O efeito não desejado poderia ser o de a alteração acabar impedindo o algoritmo de alcançar alguma solução diferente que poderia vir a ser mais eficaz.

REFERÊNCIAS

ABDELHALIM, M.; SALAMA, A.; HABIB, S. Hardware software partitioning using particle swarm optimization technique. *The 6th international workshop on system-on-chip for real-time applications*, IEEE Computer society press, Cairo, p. 189–194, 12 2006.

ARAUJO, E. Lógica difusa (fuzzy) e raciocínio aproximado: Conceitos e aplicações. Synergismus Scyentifica, v. 4, n. 2, 2009.

ATKIN, H.; ALTIN, V. Rule-based fuzzy logic controller for a pwr-type nuclear power plant. *IEEE Transactions on nuclear science*, IEEE Computer society press, Istanbul, Turkey, v. 38, p. 883–890, 04 1991.

BABUSKA, R. *Fuzzy modeling for control*. Boston: Klumer academic publishers, 1998.

BENI, G.; WANG, J. Swarm intelligence in cellular robotic systems. In: *NATO advanced workshop on robots and biological systems*. [S.l.: s.n.], 1989.

BLANCHE, R.; DUBUCS, J.-P. *História da lógica*. [S.l.]: EDIÇOES 70, 1899.

CHANUSSOT, J.; MAURIS, G.; LAMBERT, P. Fuzzy fusion techniques for linear features detection in multitemporal sar images. *IEEE Transactions on geoscience and remote sensing*, IEEE Computer society press, Annecy, France, v. 37, p. 1292–1305, 05 1999.

CHEN, L.; CHEN, C. Pre-shaped fuzzy c-means algorithm (pfcm) for transparent membership function generation. *Man and cybernetics, 2007. ISIC. IEEE International conference on systems*, IEEE Computer society press, Montreal, Que., n. 789–794, p. 789–794, 10 2007.

CHEN, Y.; PENG, L.; ABRAHAM, A. Programming hierarchical ts fuzzy systems. *Evolving fuzzy systems, 2006 international symposium*, IEEE Computer society press, Ambleside, p. 157–162, 9 2006.

CINGOLANI, P. *Pacote de sistemas fuzzy para Java, jFuzzyLogic*. 2008. Disponível em: <http://jfuzzylogic.sourceforge.net/html/index.html>.

CINTRA, M.; CAMARGO, H. Fuzzy rules generation with pre-selection of candidate rules. *submited for publication in Portuguese*, 2007.

CINTRA, M. E.; CAMARGO, H. de A. Fuzzy rules generation using genetic algorithms with self-adaptive selection. *IEEE International conference on information reuse and integration*, IEEE Computer society press, São Carlos, Brasil, p. 261–266, 08 2007.

CORDóN, O.; HERRERA, F. A hybrid genetic algorithm-evolution strategy process for learning fuzzy logic controller knowledge bases. In: *Genetic algorithms and soft computing*. [S.l.]: Physica-Verlag, 1996. p. 251–278.

COSTA, S.; NEDJAH, N.; MOURELLE, L. Modelagem automática de sistemas fuzzy utilizando o método de otimização baseada em enxame de partículas. In: *Congresso ibero-latino-americano de métodos computacionais em engenharia*. Búzios, RJ, Brasil: [s.n.], 2009.

COSTA, S.; NEDJAH, N.; MOURELLE, L. Automatic modeling of fuzzy systems using particle swarm optimization. In: *10th International conference on artificial intelligence and soft computing*. Zakopane, Polônia: Springer-Verlag, 2010. (LNAI, v. 6113), p. 35–42.

COSTA, S.; NEDJAH, N.; MOURELLE, L. Automatic modeling of fuzzy systems using particle swarm optimization. *transaction on computer science VIII*, Springer-Verlag, v. 6260, 2010.

COX, E. *The fuzzy systems handbook: A practitioner's guide to building, using, and maintaining fuzzy systems*. Oval Road, London: Academic Press Limited, 1994.

DETYNIECKI, M.; YAGER, R.; BOUCHON-MEUNIER, B. Specifying t-norms based on the value of t(1/2, 1/2). *Mathware & soft computing*, Mathware & soft computing, Granada, Espanha, v. 07, n. 1, p. 77–87, 2000.

ENGELBRECHT, A. P. *Fundamentals of computational swarm intelligence*. England: John wiley e sons ltd, 2005.

ESMIN, A.; AOKI, A.; LAMBERT-TORRES, G. Particle swarm optimization for fuzzy membership functions optimization. *IEEE International Conference on Systems, Man and Cybernetics*, IEEE Computer society press, Itajuba, Brasil, v. 3, p. 6, 10 2002.

FIGUEIREDO, K.; VELLASCO, M. M. B. R.; PACHECO, M. A. C. Controle de robô usando agentes inteligentes. *Congresso catarinense de computação*, I Congresso catarinense de computação, Santa Catarina, Brasil, v. 01, 2000.

GROBLER, J.; ENGELBRECHT, A.; YADAVALLI, V. Multi-objective de and pso strategies for production scheduling. *IEEE Congress on evolutionary computation, 2008. CEC 2008. (IEEE world congress on computational intelligence).*, IEEE Computer society press, Hong Kong, p. 1154–1161, 06 2008.

GUIMARãES, A. C. F. Um sistema difuso inteligente para avaliar informações de usuários na internet. *Ciência da informação*, IEEE Computer society press, Brasília, Brasil, v. 31, n. 3, p. 37–41, 11 2002.

GUO, B. et al. Sigmoid surface control for mini underwater vehicles by improved particle swarm optimization. *International conference on robotics and biomimetics*, IEEE Computer society press, Sanya, China, 12 2007.

HO, S. et al. Design of accurate classifiers with a compact fuzzy-rule base using an evolutionary scatter partition of feature space. *IEEE Transactions on systems, man, and cybernetics B*, IEEE Systems, man, and cybernetics society, v. 34, n. 2, p. 1031–1044, 04 2004.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. *IEEE International conference on neural networks*, IEEE Computer society press, Perth, Australia, v. 4, p. 1942–1948, 11 1995.

KHOSLA, A. et al. *Swarm intelligent systems*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2006.

KIM, M.; KIM, C.; LEE, J. Building a fuzzy model with transparent membership functions through constrained evolutionary optimization. *International journal of control, automation, and systems*, International journal of control, automation, and systems, v. 02, n. 3, p. 289–309, 09 2004.

KIM, M.-S.; KIM, C.-H.; LEE, J. jang. Evolving compact and interpretable takagi-sugeno fuzzy models with a new encoding scheme. *Man, and cybernetics, part B, IEEE transactions on systems*, IEEE Computer society press, Korea, v. 36, p. 1006–1023, 10 2006.

KRONE, A.; SLAWINSKI, T. Data-based extraction of unidimensional fuzzy sets for fuzzy rulegeneration. *IEEE World congress on computational intelligence. IEEE International conference on fuzzy systems.*, Anchorage, AK, USA, v. 02, p. 1032–1037, 05 1998.

LOPES, H. S.; COELHO, L. S. Particle swarm optimization with fast local search for the blind travelling salesman problem. *Fifth international conference on hybrid intelligent systems*, IEEE Computer society press, Rio de Janeiro, Brasil, v. 5, p. 245–250, 11 2005.

MAMDANI, E. Applications of fuzzy algorithms for simple dynamic plant. *IEEE Transactions on computers*, IEEE Computer society press, London, England, v. 121, n. 12, p. 1585–1588, 1974.

MAMDANI, E. Application of fuzzy logic to approximate reasoning using linguistic synthesis. *IEEE Transactions on computers*, IEEE computer society press, London, England, v. 26, n. 1, p. 1182–1191, 12 1977.

MARINKE, R. et al. Particle swarm optimization (pso) applied to fuzzy modeling in a thermal-vacuum system. *Fifth International Conference on Hybrid Intelligent Systems*, IEEE Computer society press, São Paulo, Brasil, p. 6, 11 2005.

MERZ, C.; MURPHY, P. *UCI Repository of machine learning databases*. 1998. Disponível em: <http://www.ics.uci.edu>.

NEDJAH, N. et al. *Fuzzy systems engineering. Theory and practice.* [S.l.]: Springer-Verlag Berlin Heidelberg, 2005.

PENA, R. et al. Vector control of a diesel-driven doubly fed induction machine for a stand-alone variable speed energy system. *IECON 02 [IEEE 28th annual conference of the industrial electronics society]*, IEEE Computer society press, Punta Arenas, Chile, v. 02, p. 985–990, 11 2002.

RIVAS, V. et al. Evolving two-dimensional fuzzy systems. *Fuzzy sets syst.*, Elsevier North-Holland, Inc., v. 138, n. 2, p. 381–398, 09 2003.

SANKAR, G.; KUMAR, S. S. Fuzzy logic based automatic braking system in trains. *TENCON 2006. 2006 IEEE region 10 conference*, IEEE Computer society press, Hong Kong, p. 1–4, 11 2006.

SETNES, M.; ROUBOS, H. Ga-fuzzy modeling and classification: complexity and performance. *IEEE Transactions fuzzy systems*, IEEE Computer society press, v. 08, n. 5, p. 509–522, 10 2000.

SHI, Y.; EBERHART, R. A modified particle swarm optimizer. In: *In proceedings of IEEE congress on evolutionary computation*. [S.l.]: IEEE Computer society press, 1998. p. 69–73.

TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modelling and control. *IEEE Transactions on systems, man, and cybernetics*, IEEE systems, man, and cybernetics society, v. 15, n. 1, p. 116–132, 02 1985.

TANSCHEIT, R. Sistemas fuzzy. In: SOCIEDADE BRASILEIRA DE AUTOMáTICA, 6., 2003, Bauru, SP. *VI simpósio brasileiro de automação inteligente*. [S.l.]: Sociedade Brasileira de Automática, 2003. p. 35.

WANG, L.-X. The wm method completed: A flexible fuzzy system approach to data mining. *IEEE transactions on fuzzy systems*, IEEE Computer society press, Hong Kong, China, v. 11, p. 768–782, 12 2003.

WANG, L.-X.; MENDEL, J. M. Generating fuzzy rules by learning from examples. *Man and cybernetics, IEEE transactions on systems*, IEEE Computer society press, California, Los Angeles, USA, v. 22, p. 1414–1427, 11-12 1992.

WENDELKEN, S.; MCGRATH, S.; BLIKE, G. A medical assessment algorithm for automated remote triage. *Engineering in medicine and biology society, 2003. Proceedings of the 25th annual international conference of the IEEE*, IEEE Computer society press, Hanover, NH, USA, v. 04, p. 3630–3633, 09 2003.

XUE, H.; CHONG, N.; JAMSHIDI, M. Fuzzy associative memory optimization using genetic algorithms. *IEEE World congress on computational intelligence. Proceedings of the third IEEE conference on fuzzy systems.*, Albuquerque, NM, USA, v. 01, p. 509–513, 06 1994.

ZADEH, L. Fuzzy sets. *Information and control*, v. 08, p. 338–353, 06 1965.

ZHAO, J.; BOSE, B. K. Evaluation of membership functions for fuzzy logic controlled induction motor drive. *IECON 02 [IEEE 2002 28th annual conference of the industrial electronics society]*, IEEE Computer society press, Dept. of Control Sci. e Eng., HUST, China, v. 1, n. 4, p. 229–234, 11 2002.

APÊNDICE A – Regras dos Sistemas *Fuzzy* Gerados

A.1 Regras do resultado da função seno 3D

O conjunto de regras abaixo é a relação das regras geradas em um sistema *fuzzy* encontrado como resultado do processamento do PSO, conforme descrito na Seção 5.1.2.1.

R1:	se x é GAU_1 e y é GAU_1	então z é GAU_4
R2:	se x é GAU_2 e y é GAU_2	então z é GAU_5
R3:	se x é GAU_3 e y é GAU_3	então z é GAU_4
R4:	se x é GAU_4 e y é GAU_4	então z é GAU_5
R5:	se x é GAU_5 e y é GAU_5	então z é GAU_7
R6:	se x é GAU_3 e y é GAU_2	então z é GAU_2
R7:	se y é GAU_4	então z é GAU_1
R8:	se x é GAU_3 e y é GAU_5	então z é GAU_4
R9:	se x é GAU_4 e y é GAU_2	então z é GAU_3
R10:	se x é GAU_4 e y é GAU_5	então z é GAU_7
R11:	se y é GAU_5	então z é GAU_3
R12:	se x é GAU_1 e y é GAU_4	então z é GAU_3
R13:	se y é GAU_2	então z é GAU_6
R14:	se x é GAU_3 e y é GAU_1	então z é GAU_2
R15:	se x é GAU_3	então z é GAU_6
R16:	se x é GAU_1 e y é GAU_5	então z é GAU_6
R17:	se x é GAU_5	então z é GAU_3
R18:	se y é GAU_3	então z é GAU_4
R19:	se x é GAU_1 e y é GAU_3	então z é GAU_3
R20:	se x é GAU_2 e y é GAU_3	então z é GAU_1
R21:	se x é GAU_1	então z é GAU_2

A.2 Regras do resultado da função exponencial

O conjunto de regras abaixo é a relação das regras geradas em um sistema *fuzzy* encontrado como resultado do processamento do PSO, conforme descrito na Seção 5.1.3.1.

R1:	se x é GAU_1 e y é GAU_1	então z é GAU_4
R2:	se x é GAU_1 e y é GAU_2	então z é GAU_6
R3:	se x é GAU_2 e y é GAU_2	então z é GAU_6
R4:	se x é GAU_2 e y é GAU_3	então z é GAU_5
R5:	se x é GAU_3 e y é GAU_3	então z é GAU_5
R6:	se x é GAU_3 e y é GAU_4	então z é GAU_4
R7:	se x é GAU_4 e y é GAU_5	então z é GAU_3
R8:	se x é GAU_5 e y é GAU_6	então z é GAU_4
R9:	se x é GAU_6 e y é GAU_7	então z é GAU_5
R10:	se x é GAU_6 e y é GAU_8	então z é GAU_6
R11:	se x é GAU_7 e y é GAU_8	então z é GAU_6
R12:	se x é GAU_7 e y é GAU_9	então z é GAU_5
R13:	se x é GAU_3 e y é GAU_8	então z é GAU_4
R14:	se y é GAU_5	então z é GAU_2
R15:	se x é GAU_6	então z é GAU_2
R16:	se y é GAU_1	então z é GAU_5
R17:	se x é GAU_6 e y é GAU_5	então z é GAU_9
R18:	se x é GAU_4	então z é GAU_3
R19:	se x é GAU_3	então z é GAU_1
R20:	se y é GAU_4	então z é GAU_3
R21:	se x é GAU_7 e y é GAU_3	então z é GAU_6
R22:	se y é GAU_2	então z é GAU_2
R23:	se y é GAU_9	então z é GAU_2
R24:	se x é GAU_7 e y é GAU_2	então z é GAU_4
R25:	se x é GAU_5 e y é GAU_4	então z é GAU_8
R26:	se x é GAU_7	então z é GAU_5
R27:	se x é GAU_5 e y é GAU_9	então z é GAU_8
R28:	se x é GAU_6 e y é GAU_6	então z é GAU_4
R29:	se x é GAU_6 e y é GAU_1	então z é GAU_7

A.3 Regras do resultado do problema da flor iris

O conjunto de regras abaixo é a relação das regras geradas em um sistema *fuzzy* encontrado como resultado do processamento do PSO, conforme descrito na Seção 5.2.1.1.

R1:	se CS é GAU_2 e LS é GAU_3 e CP é GAU_1 e LP é GAU_1	então CL é GAU_1
R2:	se CS é GAU_1 e LS é GAU_3 e CP é GAU_1 e LP é GAU_1	então CL é GAU_1
R3:	se CP é GAU_2	então CL é GAU_2
R4:	se LP é GAU_3	então CL é GAU_3
R5:	se CS é GAU_1 e LS é GAU_2	então CL é GAU_1
R6:	se CS é GAU_1 e LS é GAU_2 e CP é GAU_3 e LP é GAU_1	então CL é GAU_2
R7:	se CS é GAU_3 e LS é GAU_1 e LP é GAU_1	então CL é GAU_2
R8:	se CS é GAU_2 e CP é GAU_1	então CL é GAU_1
R9:	se CP é GAU_1	então CL é GAU_1

A.4 Regras do resultado do problema da avaliação de veículo

O conjunto de regras abaixo é a relação das regras geradas em um sistema *fuzzy* encontrado como resultado do processamento do PSO, conforme descrito na Seção 5.2.2.1.

R1: se VC é GAU_1 e MN é GAU_2 e PO é GAU_3 e PE é GAU_3 e PM é GAU_1 e NS é GAU_2 então CL é GAU_1

R2: se VC é GAU_1 e MN é GAU_3 e PO é GAU_4 e PE é GAU_3 e PM é GAU_1 e NS é GAU_2 então CL é GAU_1

R3: se VC é GAU_1 e MN é GAU_3 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R4: se VC é GAU_1 e MN é GAU_4 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_1

R5: se VC é GAU_1 e MN é GAU_4 e PO é GAU_3 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_1

R6: se VC é GAU_2 e MN é GAU_4 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_1

R7: se VC é GAU_1 e MN é GAU_4 e PO é GAU_4 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_1

R8: se VC é GAU_1 e MN é GAU_1 e PO é GAU_2 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_1

R9: se VC é GAU_1 e MN é GAU_2 e PO é GAU_2 e PE é GAU_3 e PM é GAU_1 e NS é GAU_2 então CL é GAU_1

R10: se VC é GAU_1 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_1

R11: se VC é GAU_1 e MN é GAU_2 e PO é GAU_3 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_1

R12: se VC é GAU_1 e MN é GAU_3 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_1

R13: se VC é GAU_1 e MN é GAU_3 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R14: se VC é GAU_1 e MN é GAU_4 e PO é GAU_4 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_1

R15: se VC é GAU_3 e MN é GAU_1 e PO é GAU_1 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_1

R16: se VC é GAU_3 e MN é GAU_1 e PO é GAU_2 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_2

R17: se VC é GAU_2 e MN é GAU_1 e PO é GAU_1 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_1

R18: se VC é GAU_3 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_1

R19: se VC é GAU_3 e MN é GAU_2 e PO é GAU_1 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_1

R20: se VC é GAU_4 e MN é GAU_2 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_3

R21: se VC é GAU_4 e MN é GAU_2 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R22: se VC é GAU_3 e MN é GAU_3 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R23: se VC é GAU_3 e MN é GAU_3 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_1

R24: se VC é GAU_3 e MN é GAU_3 e PO é GAU_2 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_1

R25: se VC é GAU_3 e MN é GAU_3 e PO é GAU_2 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_1

R26: se VC é GAU_3 e MN é GAU_3 e PO é GAU_2 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_1

R27: se VC é GAU_3 e MN é GAU_1 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_3 então CL é GAU_1

R28: se VC é GAU_3 e MN é GAU_1 e PO é GAU_3 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_2

R29: se VC é GAU_3 e MN é GAU_1 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R30: se VC é GAU_3 e MN é GAU_1 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R31: se VC é GAU_3 e MN é GAU_1 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R32: se VC é GAU_3 e MN é GAU_2 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R33: se VC é GAU_3 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_1

R34: se VC é GAU_3 e MN é GAU_2 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R35: se VC é GAU_3 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_1

R36: se VC é GAU_3 e MN é GAU_2 e PO é GAU_3 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_1

R37: se VC é GAU_1 e MN é GAU_1 e PO é GAU_4 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_2

R38: se VC é GAU_1 e MN é GAU_1 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_2

R39: se VC é GAU_1 e MN é GAU_1 e PO é GAU_1 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_2

R40: se VC é GAU_1 e MN é GAU_1 e PO é GAU_4 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_2

R41: se VC é GAU_1 e MN é GAU_1 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_2

R42: se VC é GAU_1 e MN é GAU_2 e PO é GAU_1 e PE é GAU_3 e PM é GAU_2 e NS é GAU_3 então CL é GAU_2

R43: se VC é GAU_1 e MN é GAU_2 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_2

R44: se VC é GAU_1 e MN é GAU_2 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_2

R45: se VC é GAU_1 e MN é GAU_2 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_2

R46: se VC é GAU_4 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_2 então CL é GAU_2

R47: se VC é GAU_4 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_2

R48: se VC é GAU_1 e MN é GAU_2 e PO é GAU_1 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_2

R49: se VC é GAU_1 e MN é GAU_1 e PO é GAU_1 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_2

R50: se VC é GAU_1 e MN é GAU_1 e PO é GAU_2 e PE é GAU_3 e PM é GAU_1 e NS é GAU_3 então CL é GAU_2

R51: se VC é GAU_1 e MN é GAU_1 e PO é GAU_3 e PE é GAU_3 e PM é GAU_2 e NS é GAU_2 então CL é GAU_4

R52: se VC é GAU_1 e MN é GAU_1 e PO é GAU_1 e PE é GAU_1 e PM é GAU_3 e NS é GAU_3 então CL é GAU_3

R53: se VC é GAU_1 e MN é GAU_1 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R54: se VC é GAU_1 e MN é GAU_1 e PO é GAU_4 e PE é GAU_1 e PM é GAU_2 e NS é GAU_2 então CL é GAU_3

R55: se VC é GAU_1 e MN é GAU_2 e PO é GAU_1 e PE é GAU_1 e PM é GAU_2 e NS é GAU_1 então CL é GAU_3

R56: se VC é GAU_1 e MN é GAU_2 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R57: se VC é GAU_1 e MN é GAU_2 e PO é GAU_3 e PE é GAU_1 e PM é GAU_3 e NS é GAU_3 então CL é GAU_3

R58: se VC é GAU_1 e MN é GAU_2 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_1

R59: se VC é GAU_2 e MN é GAU_3 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R60: se VC é GAU_2 e MN é GAU_3 e PO é GAU_4 e PE é GAU_3 e PM é GAU_2 e NS é GAU_1 então CL é GAU_3

R61: se VC é GAU_1 e MN é GAU_3 e PO é GAU_4 e PE é GAU_1 e PM é GAU_1 e NS é GAU_1 então CL é GAU_3

R62: se VC é GAU_1 e MN é GAU_4 e PO é GAU_4 e PE é GAU_1 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R63: se VC é GAU_1 e MN é GAU_4 e PO é GAU_4 e PE é GAU_1 e PM é GAU_3 e NS é GAU_3 então CL é GAU_3

R64: se VC é GAU_1 e MN é GAU_4 e PO é GAU_2 e PE é GAU_1 e PM é GAU_1 e NS é GAU_3 então CL é GAU_2

R65: se VC é GAU_1 e MN é GAU_4 e PO é GAU_1 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R66: se VC é GAU_1 e MN é GAU_1 e PO é GAU_4 e PE é GAU_1 e PM é GAU_3 e NS é GAU_2 então CL é GAU_3

R67: se VC é GAU_1 e MN é GAU_1 e PO é GAU_1 e PE é GAU_3 e PM é GAU_2 e NS é GAU_1 então CL é GAU_3

R68: se VC é GAU_4 e MN é GAU_1 e PO é GAU_1 e PE é GAU_1 e PM é GAU_3 e NS é GAU_2 então CL é GAU_3

R69: se VC é GAU_1 e MN é GAU_1 e PO é GAU_2 e PE é GAU_1 e PM é GAU_2 e NS é GAU_2 então CL é GAU_3

R70: se VC é GAU_1 e MN é GAU_3 e PO é GAU_3 e PE é GAU_3 e PM é GAU_2 e NS é GAU_1 então CL é GAU_3

R71: se VC é GAU_1 e MN é GAU_3 e PO é GAU_2 e PE é GAU_1 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R72: se VC é GAU_1 e MN é GAU_3 e PO é GAU_2 e PE é GAU_1 e PM é GAU_1 e NS é GAU_1 então CL é GAU_3

R73: se VC é GAU_4 e MN é GAU_4 e PO é GAU_3 e PE é GAU_3 e PM é GAU_1 e NS é GAU_1 então CL é GAU_3

R74: se VC é GAU_1 e MN é GAU_4 e PO é GAU_3 e PE é GAU_1 e PM é GAU_2 e NS é GAU_3 então CL é GAU_3

R75: se VC é GAU_1 e MN é GAU_4 e PO é GAU_4 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R76: se VC é GAU_1 e MN é GAU_4 e PO é GAU_2 e PE é GAU_3 e PM é GAU_1 e NS é GAU_1 então CL é GAU_3

R77: se VC é GAU_1 e MN é GAU_4 e PO é GAU_3 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R78: se VC é GAU_3 e MN é GAU_2 e PO é GAU_2 e PE é GAU_3 e PM é GAU_3 e NS é GAU_1 então CL é GAU_3

R79: se VC é GAU_3 e MN é GAU_2 e PO é GAU_2 e PE é GAU_1 e PM é GAU_2 e NS é GAU_1 então CL é GAU_4

R80: se VC é GAU_3 e MN é GAU_2 e PO é GAU_3 e PE é GAU_1 e PM é GAU_1 e NS é GAU_2 então CL é GAU_4

R81: se VC é GAU_3 e MN é GAU_3 e PO é GAU_4 e PE é GAU_1 e PM é GAU_2 e NS é GAU_2 então CL é GAU_4

R82: se VC é GAU_3 e MN é GAU_3 e PO é GAU_1 e PE é GAU_3 e PM é GAU_2 e NS é GAU_1 então CL é GAU_3

R83: se VC é GAU_3 e MN é GAU_3 e PO é GAU_1 e PE é GAU_1 e PM é GAU_1 e NS é GAU_1 então CL é GAU_4

R84: se VC é GAU_3 e MN é GAU_4 e PO é GAU_1 e PE é GAU_3 e PM é GAU_1 e NS é GAU_1 então CL é GAU_4