Airton Motoki Tamakoshi

Abordagem Coevolutiva com Processamento Paralelo para a Obtenção de Sistemas Fuzzy

Itajubá
2016

Airton Motoki Tamakoshi

Abordagem Coevolutiva com Processamento Paralelo para a Obtenção de Sistemas Fuzzy

Dissertação de Mestrado apresentada à Universidade Federal de Itajubá como parte dos requisitos para obtenção do título de Mestre em Ciência e Tecnologia da Computação.

Universidade Federal de Itajubá – UNIFEI
Ciência e Tecnologia da Computação
Programa de Pós-Graduação

Orientador: Jeremias Barbosa Machado
Coorientador: Edmilson Marmo Moreira

Itajubá
2016

AGRADECIMENTOS

Agradeço aos meus pais que forneceram condições e estrutura para que eu pudesse realizar meu sonho. Agradeço aos demais familiares e amigos por todo apoio recebido.

Agradeço ao meu orientador Prof. Dr. Jeremias Barbosa Machado e ao coorientador Prof. Dr. Edmilson Marmo Moreira, que me orientaram e guiaram durante a realização deste trabalho.

Agradeço aos professores do Instituto de Engenharia de Sistemas e Tecnologias da Informação (IESTI).

Agradeço a CAPES pela bolsa de estudos concedida.

RESUMO

O presente trabalho tem como objetivo apresentar uma proposta capaz de melhorar o desempenho do processo de obtenção de modelos *fuzzy* por metodologia evolutiva hierárquica conhecida como CoevolGFS. O CoevolGFS é utilizado na obtenção de modelos lineares e não-lineares *fuzzy* a partir de dados de entrada e saída do sistema ou função sob análise. A determinação dos parâmetros dos modelos ocorre por meio da utilização de algoritmos genéticos (AG) aplicados em uma representação hierárquica. Esta representação facilita a modelagem e identificação uma vez que simplifica o mapeamento dos parâmetros em cromossomos para a utilização de AG. No entanto, processos com um grande número de parâmetros e/ou variáveis demandam um alto custo computacional para a determinação de seu modelo. Este elevado custo computacional pode impedir a utilização da abordagem original. A metodologia proposta neste trabalho consiste em aplicar técnicas de processamento paralelo em conjunto com o CoevolGFS visando melhorar a eficiência do método. Desta forma, pretende-se realizar execuções e a obtenção de modelos complexos em um menor tempo de processamento frente a metodologia original. Para ilustrar a eficiência do método, apresenta-se a aplicação deste na obtenção de modelos *fuzzy* para funções não-lineares com diferentes números de variáveis.

Palavras-chaves: Modelagem não linear, Sistemas *fuzzy*, Algoritmo genético, processamento paralelo.

ABSTRACT

The main proposal of this work is to present an approach to improve the performance of a process used for the obtaining of fuzzy models known as CoevolGFS. CoevolGFS is an hierarchical evolutionary approach developed for the obtaining of linear and nonlinear fuzzy models from input and output data. The determination of the model parameters occurs through using of genetic algorithms (GA) applied in a hierarchical representation. This representation facilitates the modeling and identification process since it simplifies the parameters mapping in chromosomes. However, processes with a large number of parameters and/or variables may require a high computational cost. This computational cost can prevent the use of the original methodology due to the time required to obtain those models. The approach proposed in this dissertation consists of applying parallel processing techniques in conjunction with the CoevolGFS. Thus, it is intended to carry out the execution and obtaining complex models in a shorter time when compared to the original method. To illustrate the efficiency of the proposed methodology,it is presented the application of this one in the obtaining of nonlinear fuzzy models for functions with different numbers of variables.

Key-words: Nonlinear modeling, Fuzzy systems, Genetic algorithm, Parallel processing.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

A modelagem de sistemas reais é importante para praticamente todas as áreas do conhecimento. Modelos matemáticos são aplicados para realizar análises de sistemas, permitindo um melhor entendimento sobre seu funcionamento. Na Engenharia, os modelos são utilizados de diversas formas: desenvolvimento de novas formas de processo, análise de processos já existentes, otimização, supervisão, detecção e diagnóstico de falhas. Modelos lineares são utilizados para solucionar diversos tipos de problemas, mas atualmente existe uma demanda cada vez maior na solução de modelos para sistemas não-lineares (NELLES, 2013).

Os sistemas que requerem a utilização de uma modelagem não-linear são normalmente mais complexos. Suas variáveis e seu funcionamento dificilmente são modelados por uma pessoa sem o auxílio de uma ferramenta adequada. Sendo assim, em geral, tem-se a necessidade da utilização de recursos como sistemas computacionais para auxiliar na modelagem do sistema.

A modelagem matemática de sistemas reais pode ser realizada de diversas formas. Uma das alternativas aplicadas é a utilização da teoria dos conjuntos nebulosos (*fuzzy*) (NELLES, 2013). A teoria dos conjuntos nebulosos permite realizar modelagens lineares e não-lineares, apresentando soluções eficientes para diversos tipos de problemas e situações.

A teoria dos conjuntos nebulosos é, de forma simplificada, uma extensão da teoria dos conjuntos clássicos. Realizar a construção de um modelo *fuzzy*, em geral, toma como base o conhecimento de um especialista. A necessidade de um especialista para auxiliar na construção de um sistema *fuzzy* pode apresentar alguns problemas, como por exemplo: o conhecimento do especialista limita a construção do sistema e a necessidade de um ou mais especialistas para o desenvolvimento do modelo (LIMA; PINHEIRO; OLIVEIRA, 2014). Outra forma de construção dos sistemas nebulosos é baseado em métodos de otimização ou aprendizado de máquina. Tais formas de construção permitem obter um modelo *fuzzy* sem necessitar do auxílio de um especialista de forma direta, extraindo as informações do problema de forma automática (NELLES, 2013). Mesmo que a extração automática de informações demande maior recurso computacional sua aplicação se torna fundamental quando é utilizada para solucionar problemas de dimensões elevadas.

Dentre as diversas técnicas existentes para a obtenção de um sistema *fuzzy*, o Algoritmo Genético (AG) recebe um grande destaque. A utilização dos AGs para solucionar problemas foi popularizada por ser uma estrutura simples e de fácil entendimento. O AG toma como inspiração a evolução e seleção natural da biologia. O algoritmo é aplicado em diversos tipos de problemas, com destaque aos problemas de explosão combinatória (HOLLAND, 1975).

A estrutura do algoritmo genético é baseada em um população de cromossomos, possíveis respostas para um problema. Para cada cromossomo é atribuído um valor de aptidão (*fitness*), que representa a qualidade da possível solução para o problema. O algoritmo genético manipula os cromossomos através de operações genéticas (seleção, cruzamento e mutação) buscando por respostas melhores. Em muitos dos casos o algoritmo genético é executado de forma paralela.

A execução do algoritmo genético de forma paralela é uma tarefa consideravelmente simples de ser realizada, já que grande parte das operações podem ser executadas de forma simultânea. Os indivíduos representados por seus cromossomos são estruturas que independem uma das outras, permitindo realizar o cálculo do *fitness* de forma paralela. O mesmo ocorre com as operações genéticas (seleção, cruzamento e mutação). Para estruturas de algoritmos genéticos clássicos encontra-se na literatura várias soluções implementadas de forma paralela (ALBA; TROYA, 1999; KONFRST, 2004).

A utilização dos algoritmos genéticos para a obtenção automática de sistemas *fuzzy* é um tipo de solução híbrida denominada *Genetic Fuzzy System - GFS*. A ideia de tal combinação é utilizar diversos sistemas nebulosos verificando o desempenho que cada um deles possui, realizando operações genéticas entre eles e buscando obter sistemas *fuzzy* cada vez melhores. O desempenho de cada indivíduo é avaliado a partir de uma base de dados de entrada e saída desejada, verificando o erro apresentado por cada sistema (NELLES, 2013).

A aplicação do AG para determinar modelos *fuzzy* é uma solução que permite buscar, dentre diversas possibilidade de sistema, o mais adequado. No entanto, a utilização deste tipo de abordagem pode apresentar problema tornando-o inviável quando a codificação do sistema *fuzzy* em um único cromossomo se torna extensa. Para contornar este problema, foi proposto por Delgado (2002) uma metodologia baseada em níveis hierárquicos para realizar a representação de modelos *fuzzy* de forma mais amigável e facilitar a obtenção de modelos *fuzzy* via AG. Esta metodologia foi denominada CoevolGFS.

A principal diferença do CoevolGFS dos demais sistemas encontrados na literatura é a forma de codificação do sistema *fuzzy* e a aplicação da teoria de coevolução (MORIARTY; MIIKKULAINEN, 1998). O sistema *fuzzy* apresentado por Delgado (2002) é caracterizado por utilizar o modelo de Takagi-Sugeno com mapeamento do consequente realizado por funções não-lineares. As principais vantagens na utilização do modelo de Takagi-Sugeno, quando comparado ao modelo de Mamdani, é a capacidade de representar sistemas complexos utilizando um menor número de regras. Já o uso das funções do tipo não-linear permite realizar uma melhor representação de cada regra. Mais informações sobre como é codificado e funcionamento do CoevolGFS serão apresentadas no Capítulo 4. Para melhor compreensão do trabalho é apresentado uma breve descrição sobre a computação evolutiva, com foco no algoritmo genético, no Capítulo 2 e sistemas nebulosos, com foco no modelo de Takagi-Sugeno, no Capítulo 3.

No entanto, Delgado (2002) apresenta um problema relacionado à execução do CoevolGFS, que ocorre também em outros GFS. Este problema é o custo computacional elevado quando existe um grande número de variáveis no sistema. O presente trabalho têm como objetivo propor uma solução para este problema utilizando metologias de computação paralela nos pontos naturalmente paralelizáveis e priorizando as etapas que mais demandam tempo de execução.

De forma geral, computação paralela consiste em solucionar um problema dividindo a execução em vários processadores ou máquinas em *cluster* (JAVED et al., 2015). A motivação para o desenvolvimento em paralelo do CoevolGFS está relacionada ao cenário atual da Eletrônica e da Computação tendo em vista que cada vez mais se apresentam computadores com múltiplos núcleos. Estes computadores possibilitam a execução de diversos programas de forma paralela dentro de um mesmo processador, além de permitirem que um programa utilize diversos núcleos do processador para garantir um bom desempenho. Neste sentido, o objetivo deste trabalho é apresentar uma metodologia que possibilita realizar a execução do trabalho apresentado por Delgado (2002) em um processador com múltiplos núcleos e em vários computadores conectados via rede em uma estrutura de *cluster*. A proposta é avaliar a diferença no desempenho do algoritmo utilizando diferentes recursos computacionais e avaliar a metodologia proposta de paralelização em problemas com diferente números de dimensões.

O presente trabalho possui a seguinte estrutura, no Capítulo 2 é apresentada uma introdução e uma breve explicação sobre a computação evolutiva com foco no algoritmo genético. No Capítulo 3 é introduzido o conceito de sistemas nebulosos, apresentando o modelo de Takagi-Sugeno e a forma de otimização de seus consequentes. Já no Capítulo 4 é apresentado o trabalho desenvolvido por Delgado (2002), com a apresentação da proposta deste trabalho na Seção 4.8. No Capítulo 5 apresentam-se os testes realizados e seus respectivos resultados. Finalizando, apresenta-se no Capítulo 6 um resumo dos resultados com as conclusões sobre a proposta original apresentada, além de apresentar as possibilidades de futuros estudos.

2 COMPUTAÇÃO EVOLUTIVA

2.1 INTRODUÇÃO À COMPUTAÇÃO EVOLUTIVA

Nas décadas de 50 e 60 diversos estudos sobre computação evolutiva foram desenvolvidos. A motivação para tais estudos é relacionada às ferramentas de otimização utilizadas em problemas de engenharia. Os estudos da computação evolutiva têm como objetivo evoluir candidatos a solução de um dado problema através de técnicas baseadas na Biologia, utilizando a variação genética e a seleção natural na busca de uma melhor solução para um determinado problema (MITCHELL, 1998).

A computação evolutiva permite buscar soluções para problemas que requerem características como adaptação, inovação e paralelismo. Alguns problemas computacionais sofrem variações e neste sentido a proposta de solução deve conseguir se adaptar a tais variações sem prejudicar os resultados. A computação evolutiva permite elaborar propostas de soluções que mantenham um bom desempenho mesmo em ambientes que sofrem constantes alterações. Sob o ponto de vista da inovação, esta é a característica que permite o sistema evolutivo encontrar novas soluções, como um novo algoritmo ou novos parâmetros para um determinado resultado. A possibilidade de apresentar um novo tipo de solução para um problema é de grande valor para o desenvolvimento tecnológico. A terceira característica que favorece a aplicação dos algoritmos evolutivos é a facilidade em empregar o paralelismo ou computação paralela. O processo evolutivo da computação é análogo ao da Biologia, podendo ocorrer em diversos indivíduos de forma simultânea.

2.1.1 TERMINOLOGIA

A computação evolutiva (CE) é inspirada na Biologia e tem muitas das terminologias empregadas em tal metodologia baseadas na origem biológica. A estrutura dos elementos da computação evolutiva é relativamente mais simples que seus análogos da biologia. Para a melhor compreensão é necessário apresentar alguns dos termos empregados e suas definições (GRIFFITHS et al., 2013):

- Cromossomo: estrutura nucleoprotéica formada por uma cadeia de DNA, base física dos genes da célula. Em CE o cromossomo é o conjunto de valores que codificam os parâmetros da possível solução do problema;
- Indivíduo: organismo que interage com o ambiente, possui seu próprio conjunto de cromossomos no interior de suas células. Composto por um ou mais cromossomos, possuindo um valor de aptidão, resultado da associação dos parâmetros do cromossomo no ambiente;

- Gene: bloco funcional de DNA que codifica uma proteína específica. Em CE é a codificação de um único parâmetro do cromossomo;
- Alelo: Segmento homólogo de DNA, são genes que se encontram no mesmo locus e afetam as mesmas características do indivíduo. Conjunto de valores que o gene pode assumir;
- *locus*: local fixo em um cromossomo onde se encontra um determinado gene;
- *fitness*: Relação entre o indivíduo e o ambiente. Expressa a aptidão de um determinado indivíduo à solução do problema através da quantificação da qualidade de sua solução;
- ambiente: Espaço onde diversos indivíduos de diversas espécies se encontram e interagem. Em CE é representado por uma função ou relação entre os parâmetros dos indivíduos e o dado problema;
- população: Conjunto de indivíduos de uma mesma espécie;
- Seleção: método responsável por selecionar quais indivíduos participam da reprodução;
- cruzamento (*crossover*): Operador responsável por realizar a troca de material genético entre dois indivíduos. Responsável em definir como ocorre a troca;
- mutação: Operador responsável em realizar pequenas alterações em um cromossomo. A chance de ocorrência da mutação é normalmente baixa.

Na CE, normalmente um indivíduo possui apenas um cromossomo, que é um conjunto de valores da solução candidata do problema. Com base nessa abstração é possível realizar a implementação computacional de todos os demais elementos.

Além das terminologias inspiradas na Biologia, é possível encontrar outros elementos importantes para a CE. Exemplo: os métodos de seleção e avaliação de aptidão. Os métodos de seleção são os responsáveis em definir critérios para selecionar um indivíduo da população para realizar as operações de evolução.

A CE consiste de um conjunto de técnicas que utilizam como base a biologia. Os principais componentes da computação evolutiva são: população de indivíduos candidatos a solução, função de aptidão, métodos de seleção e operadores genéticos. As principais técnicas da computação evolutiva encontradas na literatura são: estratégias evolutivas, programação evolutiva e algoritmos genéticos (JONES, 1998).

A técnica de estratégias evolutivas foi apresentada por Rechenberg (1965) e desenvolvidas por Schwefel (1975). O objetivo inicial era solucionar problemas de otimização de parâmetros reais simulando o princípio da variação genética da evolução natural, através de operadores de mutação provocando pequenas mudanças nos parâmetros.

A programação evolutiva foi desenvolvida por Fogel (1966). O objetivo era criar uma técnica de inteligência artificial através da evolução de máquinas de estado finito codificando um alfabeto finito. Posteriormente foi estendida codificando valores reais. As máquinas de estado finito são evoluídas através do operador de mutação e seleção dos candidatos mais adaptados.

O algoritmo genético foi desenvolvido por Holland (1975). O modelo clássico codificava os indivíduos de forma binária. O objetivo inicial não era solucionar um problema em específico, como ocorreu com as estratégias evolutivas e a programação evolutiva, mas simular o fenômeno de adaptação natural. O desenvolvimento do algoritmo genético formaliza os fundamentos teóricos do processo de adaptação do indivíduo na computação evolutivas.

Mesmo as três técnicas sendo desenvolvidas de forma independente, suas estruturas assim como o funcionamento básico dos operadores são similares. Os termos utilizados em computação evolutiva são empregados para todos os métodos relacionados aos algoritmos que tomam a seleção e evolução natural como base.

Muitos dos métodos da computação evolutiva são utilizados para otimização de parâmetros dentro de um universo de valores. Realizar tal atividade de forma convencional pode ser uma atividade inviável quando o número de parâmetros é grande ou o espaço de busca é extenso, sendo necessário fazer uso de técnicas mais inteligentes.

2.1.2 COMPUTAÇÃO EVOLUTIVA COMO MÉTODO DE BUSCA

A obtenção de uma solução desejada em uma coleção de candidatos é um procedimento comum em ciência da computação e recebe o nome de busca em um espaço de soluções ou espaço de busca. Espaço de busca se refere a dimensão de valores possíveis para o parâmetro.

Os métodos de busca tradicionais avaliam apenas uma solução candidata em cada instante, a expansão da busca por melhores soluções é realizada através de pequenas alterações na solução candidata gerando uma nova solução. Normalmente, se a nova solução for superior a solução anterior ela é adotada pelo sistema. As formas de busca tradicionais são métodos muito restritos, quando aplicada a espaços de busca extensos ou com uma quantidade de parâmetros elevada. Além disso, trata-se de uma execução custosa do ponto de vista computacional (DELGADO, 2002).

Quando comparado aos métodos de busca tradicionais, o método de busca com técnicas da computação evolutiva apresenta um bom desempenho. Este desempenho se justifica principalmente quando aplicado a problemas que possuem um espaço de busca extenso ou um grande número de parâmetros. No método de busca com técnicas da computação evolutiva, diversos candidatos a solução são avaliados em cada momento, permitindo realizar a busca de forma paralela. Outra característica é o fator não-determinístico resultante de etapas aleatórias da computação evolutiva. A busca estocástica aliada aos

métodos da computação evolutiva permite uma exploração ampla e uma busca mais localizada (explotação), refinando soluções candidatas já identificadas. Essa combinação de exploração ampla e explotação garante um grande potencial para a obtenção de uma solução em uma espaço de busca (MITCHELL, 1998).

2.1.3 ALGORITMO EVOLUTIVO

Algoritmos evolutivos são utilizados para solucionar problemas complexos copiando de forma simplificada o processo evolutivo de Charles Darwin (DARWIN, 1872). Para solucionar um dado problema, indivíduos (cromossomos) são criados para realizar a busca no espaço do problema apresentado. O conjunto de valores atribuído ao cromossomo representa os parâmetros para solucionar o problema. Os indivíduos competem entre si em cada momento para identificar as melhores áreas no espaço de busca descobrindo dessa forma as melhores soluções (JONES, 1998).

Segundo Jones (1998) os indivíduos que compõem o algoritmo evolutivo são representados por um conjunto de tamanho fixo de valores. Cada indivíduo codifica uma possível solução para o problema.

O funcionamento geral do algoritmo evolutivo é dado por duas etapas principais, a primeira é a de inicialização e a segunda é a iteração. Na primeira etapa ocorre a inicialização da população e dos indivíduos que a compõem. A segunda etapa é considerada o núcleo do algoritmo, onde novos indivíduos são gerados e uma nova população é definida a cada geração. No início de cada geração é realizado o cálculo da aptidão (*fitness*) de cada um dos indivíduos da população, o valor resultante do cálculo representa a qualidade da possível solução, os indivíduos que possuem um maior valor de *fitness* representam soluções melhores que os com menor valor.

Concluído o cálculo da aptidão é verificado o critério de parada, normalmente os critérios empregados são: número máximo de gerações, obtenção do *fitness* desejado ou algum critério de erro. Caso não se enquadre em nenhum dos critérios é iniciado a reprodução dos indivíduos, que é realizada pelos operadores genéticos correspondentes a cada técnica do algoritmo evolutivo. A reprodução é o momento no qual os indivíduos pertencentes a população geram indivíduos filhos para compor a nova população para a próxima geração. A Figura 1 ilustra o funcionamento geral do algoritmo evolutivo.

Segundo (JONES, 1998) algumas situações são favoráveis para a aplicação dos algoritmos evolutivos:

- Complexidade em desenvolver uma solução para o problema;
- Quando não há conhecimento suficiente para solucionar o problema;
- Se o problema esta em constante alteração;
- Quando uma solução suficientemente boa é aceita.

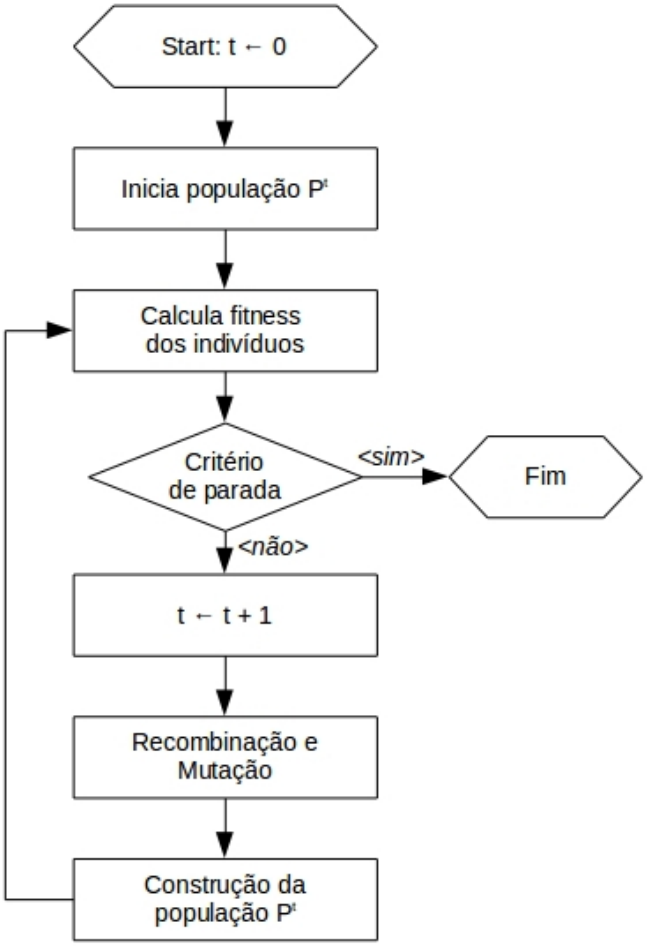

Figura 1 – Diagrama Geral do Algoritmo Evolutivo

Os principais tipos de algoritmos evolutivos encontrados na literatura são: algoritmos genéticos (AG), estratégias evolutivas (EE) e programação evolutiva (PE). Os três tipos de algoritmos citados são bem similares possuindo poucas diferenças. Todos realizam operações em estruturas genéticas de tamanho fixo. Os três possuem operadores de mutação e seleção. As EE e PE utilizam representação real para os valores, enquanto o AG, proposto por Holland, originalmente utiliza representação binária. Os AGs e as EEs possuem o operador de *crossover*. A seleção no AG e PE ocorre por probabilidade, enquanto na EE a seleção ocorre de forma determinística. Mesmo com as características definidas para cada tipo de algoritmo evolutivo é possível realizar modificações durante o processo de implementação. Como exemplo é possível citar que: o algoritmo genético pode ser implementado com representação real em seus cromossomos (JONES, 1998).

O algoritmo genético, de todos os algoritmos evolutivos, é o mais reconhecido. Seu reconhecimento é decorrente de sua extensa área de aplicação, pois o algoritmo foi desenvolvido para solucionar problemas gerais, diferente da estratégia evolutiva e da programação evolutiva. O algoritmo genético é aplicável a um grande número de problemas, o que é caracterizado como um método fraco.

2.2 ALGORITMO GENÉTICO

O algoritmo genético (AG) canônico foi desenvolvido por Holland em 1975 (HOLLAND, 1975) sendo tomado como base para outros algoritmos genéticos que vieram posteriormente. O algoritmo genético permite desenvolver novas gerações de indivíduos a partir de uma população inicial. A população inicial é gerada de forma aleatória e as novas gerações de indivíduos são produzidas através dos métodos de seleção, operações de cruzamento (crossover) e mutação nos indivíduos já existentes (MITCHELL, 1998).

Um indivíduo é representado por uma lista de valores. Cada componente da lista possui uma posição específica e é codificado com valores binários. A analogia da representação do algoritmo genético com a Biologia é: a lista de valores de um indivíduo representa o cromossomo, cada componente do cromossomo representa um gene, a posição desse gene é indicada pelo locus e a codificação dos valores representa os alelos.

Em inteligência artificial as estratégias utilizadas para solucionar os problemas podem ser divididas em fracas e fortes. Os métodos fracos são aplicados em um grande número de problemas e são recomendados em situações nas quais se tem pouco conhecimento específico sobre o problema. Quando aplicado a problemas de larga escala com uma combinação de parâmetros é alta, o desempenho é reduzido de forma significativa. Os métodos fortes são elaborados para tratar de um único problema e utilizam conhecimentos específicos do problema para auxiliar na busca por uma solução, permitindo manter um bom desempenho mesmo quando aplicado a problemas de larga escala (MICHALEWICZ; SCHOENAUER, 1996).

O algoritmo genético canônico proposto por Holland é classificado como um método fraco. Este algoritmo não utiliza conhecimentos específicos de um determinado problema sendo proposto para tratar de problemas em geral. É possível solucionar diversos problemas sem realizar alteração em sua estrutura básica. No entanto não existe uma limitação que proíba uma adaptação do algoritmo genético, possibilitando obter um melhor desempenho ou melhoria nos resultados. Sendo assim, é possível alterar sua estrutura aplicando técnicas alternativas para tratar um problema em específico.

O uso algoritmo genético é favorável a determinadas situações (MITCHELL, 1998; DELGADO, 2002), tais como:

- O espaço de busca grande;
- A superfície de desempenho mal comportada;
- Há pouca informação sobre a superfície de desempenho.

A implementação do algoritmo genético em situações contrárias às apresentadas não é indicado pois é possível solucionar tais problemas utilizando outros métodos. No caso do espaço de busca pequeno a dificuldade e o custo computacional para analisar todo o espaço

de busca procurando o ótimo global é baixo. A aplicação do algoritmo genético neste caso é pouco indicado já que existe a possibilidade do algoritmo resultar em um ótimo local.

A aplicação do algoritmo genético em superfícies de desempenho bem comportada é pouco indicada por existirem outros métodos de busca que podem ser mais apropriados. O método de subida da encosta (*hill-climbing*) e têmpera simulada (*simulated annealing*) são mais indicados nesse caso. Os dois métodos são caracterizados por apresentar uma busca direcionada e determinística em cada geração do processo.

Além dos critérios do espaço de busca, outras definições devem ser consideradas para aplicar o algoritmo genético em um problema (IYODA, 2000), tais como:

- Codificação: forma de representação do indivíduo/solução candidata;
- População inicial: forma de inicializar a população inicial;
- Função de aptidão: capacidade de um indivíduo resolver o problema;
- Operadores genéticos e método de seleção: cruzamento, mutação e seleção;
- Parâmetros utilizados pelo algoritmo genético: tamanho da população, taxa de ocorrência dos operadores.

O algoritmo genético é um algoritmo evolutivo mais específico. Ambos podem ser divididos em duas etapas: a inicialização e a iteração. O fluxograma presente na Figura 2 demonstra o funcionamento básico do algoritmo genético.

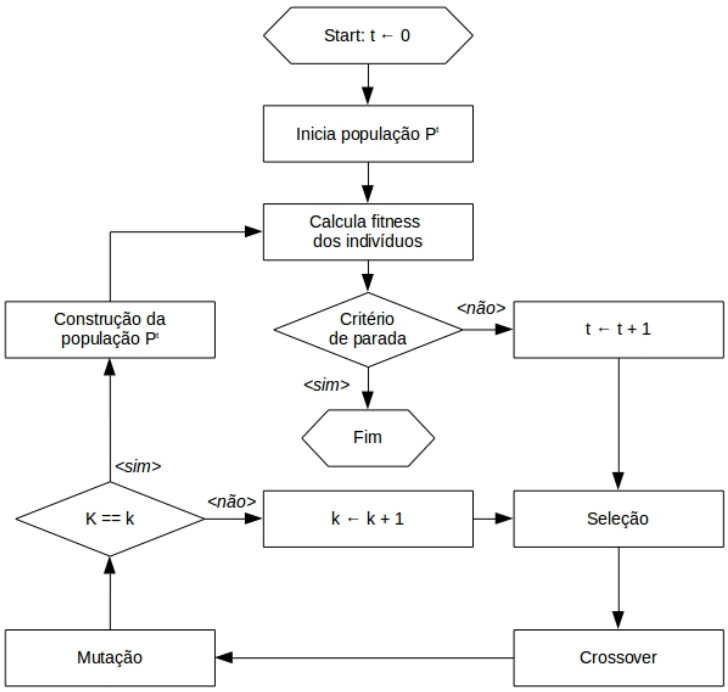

Figura 2 – Diagrama Geral do Algoritmo Genético

Conforme pode-se observar pela Figura 2 a primeira etapa consiste na construção da população inicial como é feito em todos os algoritmos evolutivos. A segunda etapa é o momento em que ocorre as iterações do algoritmo com cada iteração representando uma geração. Esta etapa é responsável pela reprodução e evolução da população. Cada geração possui as etapas de: análise do critério de parada, reprodução e obtenção da nova população. O critério de parada é igual ao do algoritmo evolutivo. A reprodução é a principal característica que diferencia o algoritmo genético dos demais algoritmos evolutivos.

A reprodução é o processo em que os N indivíduos da população geram K indivíduos. Para realizar a reprodução dois indivíduos são selecionados na população. Neste instante ocorre a troca de material genético entre os dois indivíduos pais através de algum operador de cruzamento (*crossover*) gerando dois indivíduos filhos. Para cada novo filho gerado existe uma probabilidade pequena de ocorrer a mutação, o processo de reprodução é executado até se obter K filhos. A construção da nova população é realizada pelos filhos resultantes da geração atual e caso necessário sua composição pode ser feita por indivíduos que compõem a população atual.

As etapas apresentadas do algoritmo genético são apenas a base do seu funcionamento, sendo possível adicionar outros mecanismos para melhorar seu desempenho ou adquirir alguma característica desejada. Um exemplo clássico é a adição do elitismo o que garante a preservação dos melhores indivíduos de uma geração para outra.

Para um melhor entendimento do funcionamento do algoritmo genético é necessário apresentar aspectos sobre a codificação, detalhamento dos métodos de seleção, operações de cruzamento e mutação.

2.2.1 CODIFICAÇÃO

Assim como nos algoritmos evolutivos, no algoritmo genético cada indivíduo representa um potencial candidato a solução do problema. No modelo canônico do algoritmo genético, proposto por (HOLLAND, 1975), os indivíduos são estruturas binárias de tamanho fixo.

O que justificou o uso da codificação binária por Holland está relacionado ao paralelismo implícito, no qual o uso de uma codificação binária maximiza o paralelismo. Michalewicz e Schoenauer (1996) demonstraram que o uso de um alfabeto binário possui um desempenho ruim quando as dimensões do número são grandes ou este possui muitas casas decimais. Quando se refere a dimensões grandes a representação binária exige o uso de uma estrutura de tamanho maior, enquanto que com a representação real o tamanho da estrutura é menor.

Outro aspecto tratado na literatura é o espaço de busca. Michalewicz e Schoenauer (1996) argumentam que a codificação binária aliada a um espaço de busca de dimensão elevada é desfavorável. Por outro lado, Fogel (1994) argumenta que a dimensão do espaço de busca não determina a eficiência do algoritmo, já que espaços de busca extensos podem

ser explorados de forma eficiente e o espaço de busca de dimensão reduzida pode apresentar grandes dificuldades para ser explorado.

Michalewicz e Schoenauer (1996) apresentam que a codificação binária aplicada a problemas envolvendo indivíduos com diversos valores apresenta uma estrutura extensa, dificultando a interpretação de tais valores. Por outro lado, Fogel (1994) apresenta que a codificação binário, mesmo apresentando uma estrutura maior, mantêm o funcionamento dos operadores genéticos, não necessitando de nenhuma forma de adaptação.

A forma como um indivíduo é codificado é um dos fatores fundamentais para um bom funcionamento do algoritmo genético. As principais formas de codificação de um algoritmo genético são: binária, real e em árvore (MITCHELL, 1998).

A codificação binária é a forma de codificação mais comum encontrada para o algoritmo genético por diversos motivos. O principal motivo é a proposta apresentada por Holland, no qual o algoritmo genético era proposto com codificação binária. Outro motivo é que as teorias dos algoritmos genéticos são normalmente apresentadas para estruturas de tamanho fixo e para codificação binária. Apesar desta teoria ser apresentada para a codificação binária ela pode ser aplicada à codificação real. A justificativa apresentada por Holland é relacionada ao paralelismo implícito. Mesmo que a codificação binária tenha vantagens, em alguns casos o uso da codificação binária não é natural para tratar o problema, sendo mais conveniente utilizar outra forma de codificação (HOLLAND, 1975).

A codificação real, ou com um alfabeto extenso, pode ser mais natural para codificar um indivíduo. Diversas comparações são realizadas entre a codificação binária e a real. O desempenho do algoritmo genético depende do problema que é tratado e de detalhes do algoritmo utilizado, não existindo assim, muito rigor sobre qual codificação é melhor (JANIKOW; MICHALEWICZ, 1991).

A quarta forma de codificação encontrada na literatura é a permutação. Existem problemas em que a permutação de valores se torna um fator mais importante para uma situação do que os valores atribuídos. Exemplo: no problema clássico do caixeiro viajante o objetivo é encontrar o menor trajeto que passe por todos os vilarejos. A codificação básica pode ser: um gene representa um vilarejo e o cromossomo a rota. Nesse exemplo a sequência em que são apresentados os vilarejos é fundamental para a representação da solução. Sendo assim, muitas vezes faz-se necessário realizar uma permutação de vilarejos de forma a garantir que todos sejam visitados de acordo com a representação cromossômica.

A forma de codificar um indivíduo/cromossomo no algoritmo genético é um momento crítico, uma codificação mal elaborada pode acarretar em um sistema com indivíduos que divergem dos resultados bons ou convergem de forma prematura. A codificação de um indivíduo deve ser a mais simples possível com o menor número de restrições possível (IYODA, 2000).

2.2.2 MÉTODOS DE SELEÇÃO

Após definida a forma de codificação dos indivíduos é necessário definir o método de seleção. O objetivo do método de seleção é definir a forma como os indivíduos da população são selecionados. Os indivíduos selecionados são utilizados para gerar novos indivíduos através dos operadores genéticos de cruzamento (*crossover*) e mutação. Normalmente o método de seleção favorece os indivíduos com melhor *fitness*, mas não necessariamente são selecionado apenas os melhores indivíduos. O método deve manter equilibrada a variedade genética com a pressão seletiva (MITCHELL, 1998).

A pressão seletiva é a diferença na probabilidade de seleção dos indivíduos com *fitness* alto e baixo. A pressão pode ser dividida em dois tipos básicos: pressão seletiva alta e pressão seletiva baixa. A pressão seletiva alta é a situação na qual um indivíduo com *fitness* alto possui uma probabilidade de ser selecionado muito superior a do indivíduo com *fitness* baixo. Já pressão seletiva baixa é quando um indivíduo com *fitness* alto possui uma probabilidade de ser selecionado pouco superior a do indivíduo com *fitness* baixo.

Métodos de seleção com pressão seletiva alta são aqueles que tendem a gerar indivíduos muito fortes quando comparados a média dos indivíduos da população. No momento em que um indivíduo possui um valor de aptidão muito superior ao da média da população, a probabilidade deste ser selecionado é superior. A seleção recorrente de um mesmo indivíduo provoca a perda da diversidade genética, dificultando a evolução e alteração posterior da população (MILLER; GOLDBERG, 1995).

Métodos de seleção com pressão seletiva baixa são métodos evoluem a população como um todo de forma lenta. Sua vantagem é a variedade genética, que ao contrário da pressão alta, permite manter uma evolução nas gerações futuras.

Além dos métodos tradicionais de seleção é possível encontrar métodos adicionais na literatura. Estes métodos são aqueles que adicionam alguma característica nova para evolução da população. As características podem ser a seleção por diversidade ou mesmo a adição dos melhores indivíduos da população atual na nova população.

A seguir são apresentados os principais métodos de seleção e os métodos adicionais.

2.2.2.1 SELEÇÃO PROPORCIONAL AO DESEMPENHO

No algoritmo proposto por Holland (1975), o método de seleção utilizado era o *Roulette Wheel*, conhecido também como método da roleta. A principal característica desse método é que a probabilidade de escolha de um indivíduo é proporcional ao seu *fitness*. O "valor esperado" para a seleção de um determinado indivíduo é determinado pelo seu valor de *fitness* dividido pelo somatório de todos os *fitness* da população. A Equação 2.1 apresenta como é calculado o "valor esperado". Mitchell (1998) apresenta que no método da roleta cada indivíduo é representado por uma fatia na roleta. O tamanho da fatia que um

determinado indivíduo representa é proporcional ao seu *fitness*.

$$ValorEsperado(i) = \frac{f(i)}{\sum_{n=1}^{N} f(n)} \tag{2.1}$$

A Figura 3 ilustra um exemplo com quatro indivíduos (k1, k2, k3 e k4) que possuem seus valores de *fitness*. Cada um dos indivíduos é representado na roleta por uma fatia com um tamanho equivalente ao seu valor de *fitness*. A probabilidade de escolha de um indivíduo é igual a sua proporção na roleta.

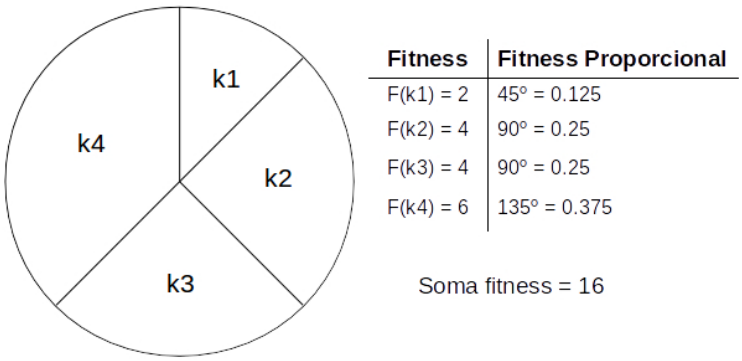

Figura 3 – Seleção Proporcional ao Desempenho (Roleta/*Roulett Wheel*)

Nos métodos de seleção, em sua grande maioria, cada indivíduo possui uma probabilidade P de ser selecionado. A chance de um determinado indivíduo ser selecionado é apresentado pela equação 2.2, com i representando o indivíduo e N o tamanho da população.

$$P_i = \frac{ValorEsperado(i)}{\sum_{n=1}^{N} ValorEsperado(n)} \tag{2.2}$$

O funcionamento básico da seleção possui as seguintes etapas: calcular T que é a soma dos valores esperados de todos os indivíduo da população; para cada seleção é necessário escolher um número aleatório r entre zero e T; percorrer todos os indivíduos da população somando os "valores esperados". No momento em que essa soma for maior ou igual a r, o indivíduo que realizou a soma deve ser selecionado.

A codificação básica dos métodos que utilizam o "valor esperado" é:

```
1 T = 0
2 for(i = 0; i < N; i++)
3         T = T + valorEsperado[i]; //Soma dos valor esperado
4 r = Random()*T; //Escolher um valor aleatório entre zero e T
5 soma = 0;
6 for(i = 0; i < N; i++){
7         soma = soma + valorEsperado[i];
```

```
8           if(somam > r)
9                   Seleciona(i);
10  }
```

Sendo N o tamanho da população, T a soma dos valores esperado, valorEsperado[i] é o "valor esperado" do indivíduo i e sum o somatório para escolha.

O método da roleta pode apresentar problemas vinculados a convergência prematura. A convergência prematura ocorre devido a pressão seletiva alta do algoritmo, esse fenômeno ocorre pois a taxa de seleção de um determinado indivíduo esta vinculado diretamente ao seu *fitness*. A existência de *fitness* muito alto quando comparado com a média da população faz com que o algoritmo da roleta reduza a variedade genética apresentando problemas na evolução futura da população.

2.2.2.2 SELEÇÃO BASEADA EM SIGMA SCALING

O método de seleção por *sigma scaling*, assim como no método da roleta, utiliza a equação do valor esperado (Equação 2.2) juntamente com o algoritmo de seleção por valor esperado. A seleção por *sigma scaling* possui a característica de fazer o algoritmo genético ser menos suscetível a convergência prematura.

Para evitar a convergência prematura, o método de seleção por *sigma scaling*, tenta manter pressão seletiva relativamente constante. A equação de seleção do método utiliza não só o *fitness* do indivíduo mas utiliza também a média e o desvio padrão do *fitness* da população. Um exemplo do cálculo do *sigma scaling* é a Equação 2.3, com i representando o indivíduo, *f(i)* o *fitness* do indivíduo i, $\bar{f}$ a média do *fitness* da população, σ é o desvio padrão da população e x é uma constante definida de forma arbitrária.

$$ValorEsperado(i) = \begin{cases} 1 + \frac{f(i)-\bar{f}}{x\sigma}, & \text{se } \sigma \neq 0 \\ 1, & \text{se } \sigma = 0 \end{cases} \tag{2.3}$$

Observa-se que o valor definido na constante x influencia na pressão seletiva do método. O aumento no valor da constante resulta em uma redução na pressão seletiva, já a redução no valor da constante resulta em um aumento na pressão seletiva.

Após definido o "valor esperado" de todos os indivíduos da população é necessário realizar o cálculo da probabilidade de seleção de cada indivíduo da população. Assim como no método da roleta, o cálculo da probabilidade é realizado conforme a Equação 2.2, sendo que para cada indivíduo da população é realizado o cálculo de P.

2.2.2.3 SELEÇÃO BASEADA NA CLASSIFICAÇÃO - *RANK*

A seleção por *rank*, assim como na seleção por *sigma scaling*, é outra forma de evitar a convergência prematura do algoritmo. A seleção por *rank* pode ser dividida em duas etapas principais. A primeira consiste em atribuir *rank* aos indivíduos da população e a segunda em definir o "valor esperado" para os indivíduos.

Na primeira etapa cada indivíduo recebe uma pontuação (*rank*). O *rank* é um valor que vai de 1 até N, onde N é o número de indivíduos da população. A atribuição do *rank* é realizada de forma crescente do pior *fitness* ao melhor *fitness*, ou seja, o indivíduo com o pior valor recebe o *rank* 1, e o melhor recebe N.

Na segunda etapa define-se o "valor esperado" para os indivíduos. O cálculo do valor esperado é estabelecido pela equação (2.4), com *min* e *max* representando os limites inferior e superior de valores da equação, sendo definidos de forma arbitrária, $rank(i)$ é o valor do *rank* atribuído ao indivíduo i na etapa anterior e N é o número de indivíduos na população.

$$ValorEsperado(i) = min + (max - min)\frac{rank(i) - 1}{N - 1} \tag{2.4}$$

Após definido o "valor esperado" de todos os indivíduos da população é necessário calcular a probabilidade de seleção do indivíduo. O cálculo é realizado da mesma forma como é feito com os métodos de seleção apresentados até o momento. O cálculo se baseia na equação (2.2), sendo que para cada indivíduo é realizado o cálculo de P.

A Tabela 1 apresenta um exemplo do cálculo da seleção por *rank* de seis indivíduos, com os parâmetros $min = 1$ e $max = 11$. O aumento no valor do parâmetro *min* diminui a pressão seletiva dos indivíduos e quando diminui seu valor a pressão seletiva aumenta. No caso do parâmetros *max*, o aumento no valor provoca um aumento na pressão seletiva, e a redução no valor resulta em uma redução na pressão seletiva.

Tabela 1 – Exemplo: Seleção *Rank*

Rank	ExpVal(i)	% seleção
1	1	3,03%
2	3	8,48%
3	5	13,94%
4	7	19,39%
5	9	24,85%
6	11	30,30%

2.2.2.4 SELEÇÃO POR TORNEIO

O método do torneio é diferenciado dos métodos que utilizam o "valor esperado" por não utilizar o *fitness* para realizar o cálculo de probabilidade para a seleção e nem realizar o cálculo da probabilidade com todos os indivíduos para dar continuidade ao algoritmo. O método do torneio utiliza o *fitness* dos indivíduos apenas para verificar qual indivíduo é melhor e não é caracterizado como um método de seleção proporcional ao *fitness*, como os métodos da roleta, *sigma* e *rank* (MITCHELL, 1998).

Para realizar a seleção por torneio é necessário definir um valor entre zero e um, esse valor é a taxa "k" de seleção do torneio. Após definido a taxa do torneio são selecionados

dois indivíduos da população de forma aleatória. É escolhido um número aleatório "r" entre zero e um, se "r < k" o indivíduo com melhor *fitness* é selecionado, caso contrário outro indivíduo é selecionado. Os dois indivíduos são devolvidos a população original podendo ser selecionados novamente..

O número de indivíduos separados para realizar o torneio foi dois, mas é possível separar mais de dois indivíduos para realizar o torneio. Com mais de dois indivíduos o algoritmo funciona da mesma forma até o momento em que é escolhido um número aleatório "r", se "r < k" o indivíduo com melhor *fitness* é escolhido, caso contrário o indivíduo com melhor *fitness* é removido do torneio e devolvido à população original. O procedimento é realizado até um indivíduo ser selecionado ou restar um indivíduo no torneio. Neste caso, restando apenas um indivíduo, este é selecionado (MILLER; GOLDBERG, 1995).

Observa-se que o valor da taxa de seleção k influencia na pressão seletiva do método, uma taxa de seleção alta resulta em uma pressão seletiva elevada e uma taxa baixa resulta em uma pressão seletiva baixa.

2.2.2.5 ELITISMO

O elitismo foi um método proposto por (JONG, 1975) tendo como propósito manter os melhores indivíduos de uma geração para a geração seguinte. A motivação no uso do elitismo no algoritmo genético se deve ao fato da possível perda dos melhores indivíduo da população após a reprodução de indivíduos em uma nova geração. Aplicar o elitismo no algoritmo genético melhora o desempenho do algoritmo genético de forma significativa (MITCHELL, 1998).

2.2.2.6 SELEÇÃO POR DIVERSIDADE

A seleção por diversidade tem como objetivo selecionar os indivíduos mais diferentes da população. Uma das formas de identificar os indivíduos mais diferentes é verificando a distância do indivíduo em relação ao indivíduo mais bem adaptado da população. O cálculo da distância pode ser realizado com a distância de Hamming, para valores binários, e distância Euclidiana, para valores reais (MITCHELL, 1998).

A forma como a distância é aplicada pode ser baseado em qualquer outro mecanismo de seleção, trocando apenas a aptidão do indivíduo pela distância entre este e o melhor indivíduo da população. A forma mais comum em que é aplicado a seleção por diversidade é similar ao elitismo, sendo que os indivíduos mais diferentes são adicionados na próxima geração.

2.2.3 OPERADORES DE CRUZAMENTO/*CROSSOVER*

Após definido como é realizada a seleção é necessário definir os operadores genéticos responsáveis pela reprodução dos indivíduos da população. Tais operadores definem o

cruzamento e a mutação, gerando novos indivíduos para compor a população da próxima geração.

A operação de cruzamento (*crossover*) é responsável em realizar a troca de material genético entre dois ou mais indivíduos previamente selecionados de uma população. O cruzamento apresenta como resultado novos indivíduos filhos distintos, sendo que cada indivíduo possui características dos cromossomos pais (MITCHELL, 1998).

Os tipos de cruzamento encontrados na literatura são o cruzamento simples, múltiplos pontos e uniforme. Eles têm como principal diferença os pontos onde são realizados as trocas de material genético.

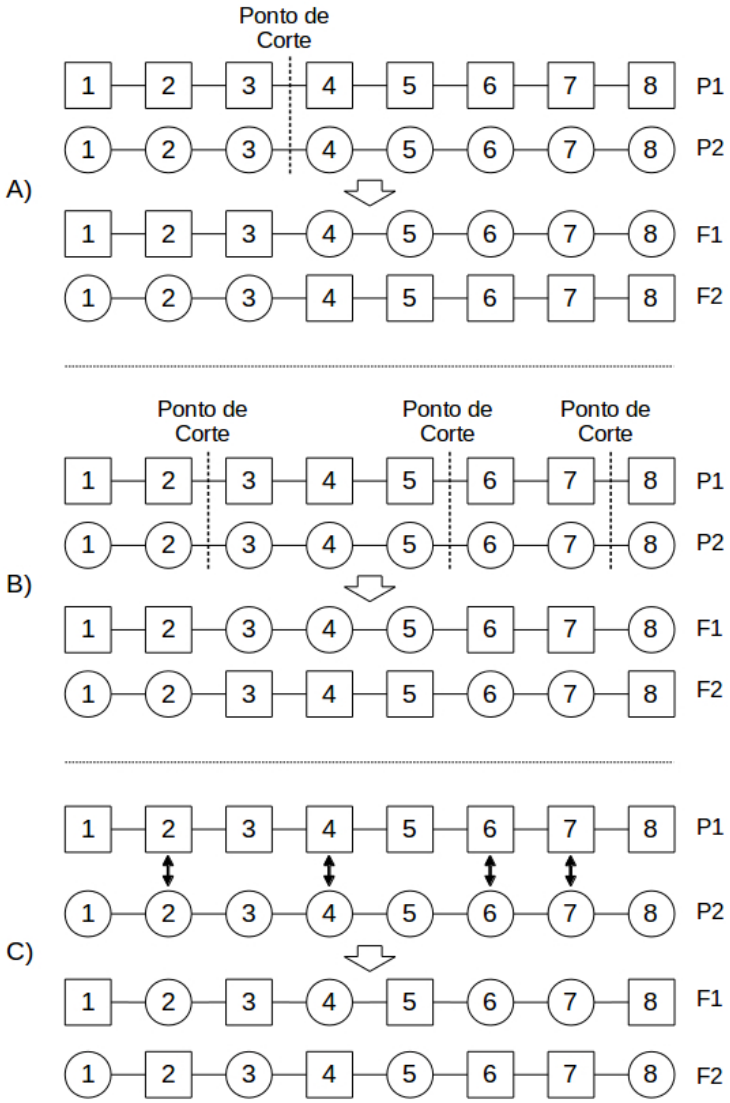

Figura 4 – Tipo de cruzamento: A) Cruzamento Simples; B) Cruzamento Múltiplo; C) Cruzamento Uniforme 50%

No cruzamento simples é definido uma posição aleatória no cromossomo para realizar tal operação. A posição definida, denominada ponto de troca ou de corte, se encontra entre dois genes e é o local em que é realizado a troca de material genético entre os cromossomos

pais. A Figura 4 ilustra o caso, temos dois filhos resultados do cruzamento. A primeira parte do cromossomo do filho F1 é herdada do pai P1 até o ponto de corte e do ponto de troca até o final do cromossomo as informações genéticas do cromossomo são do pai P2. O mesmo ocorre com o filho F2, as informações do início ao ponto de troca são do pai P2 e do ponto de troca até o final do cromossomo as informações genéticas são do pai P1.

A outra forma de cruzamento presente na literatura é a de múltiplos pontos. A diferença entre o simples e o múltiplo pontos é que no cruzamento de múltiplos pontos é definido mais de um ponto de forma aleatória para a quebra do cromossomo e a troca de material genético entre os pais. A Figura 4 ilustra como ocorre a operação apresentando três pontos de corte para realizar o cruzamento.

O terceiro operador é conhecido como cruzamento uniforme. Seu funcionamento tem como base uma porcentagem de troca. Esta determina a proporção de alelos que são trocados. A Figura 4 ilustra como é realizada a operação. O cromossomo filho F1 apresenta 50% do pai P2 e o cromossomo filho F2 apresenta 50% do pai P2. A proporção de troca no exemplo é de 50% mas é possível utilizar outros valores, nota-se que se um indivíduo possui, por exemplo, 40% das características do pai P1 o restante 60% são herdadas do pai P2.

A apresentação inicial dos três tipos de operadores de cruzamento é para codificação binária, mas não existe uma regra clara proibindo a aplicação destes em outras formas de codificação.

2.2.4 OPERADORES MUTAÇÃO

O operador de mutação assim como o operador de cruzamento é responsável pela reprodução dos indivíduos. A mutação tem como objetivo alterar um gene do cromossomo de forma aleatória, mas com uma probabilidade de ocorrência baixa. Os indivíduos que podem sofrer a mutação são os indivíduos filhos do cruzamento (MITCHELL, 1998).

O que motiva a aplicação do operador de mutação é que após a evolução de uma população por diversas gerações, os indivíduos podem estagnar em um ótimo local. A mutação permite que o algoritmo genético realize a busca em espaços não explorados através de pequenas alterações aleatórias nos genes dos novos indivíduos.

Para algumas aplicações, realizar uma mutação alterando de forma aleatória o valor do cromossomo não é a melhor alternativa para expandir o espaço de busca. Outros métodos de mutação podem ser empregados nesses casos, como por exemplo a mutação por troca de posição. A operação consistem em escolher dois pontos aleatórios em um mesmo cromossomo para realizar a troca. Essa forma de mutação é recomendada na codificação por permutação (JACOBSON; KANBER, 2015).

Mesmo que o operador de mutação tenha baixa taxa de ocorrência sua importância é fundamental para que o algoritmo genético tenha a possibilidade de evoluir sem estagnar em um ponto do espaço de busca. Mitchell (1998) apresenta que não é apenas necessária uma

escolha adequada entre cruzamento ou mutação, mas equilíbrio entre seleção, cruzamento e mutação, que são três etapas importantes para a execução do algoritmo genético.

3 SISTEMAS NEBULOSOS

Introduzidas por Zadeh (1965), a teoria dos conjuntos nebulosos ou conjuntos *fuzzy* teve uma notável evolução desde sua criação. Sua evolução é motivada pelo fato de ser aplicada em diversos tipos de problemas. Dentre suas aplicações as que mais se destacam são: modelos de controle, reconhecimento de padrão, processamento de sinal e informação, desenvolvimento de máquinas inteligentes, tomada de decisão, gerenciamento, análise de finanças, medicina, controle e automação (FENG, 2006).

A ideia de conjuntos nebulosos permite o desenvolvimento de sistemas com noções linguísticas. Sua elaboração e execução envolvem a captura de dados, representação e processamento de informação. A formalização da teoria dos conjuntos *fuzzy* possibilitou a utilização de técnicas para tratar informações imprecisas (PEDRYCZ; GOMIDE, 1998). A aplicação dos sistemas nebulosos permite tratar as informações na forma de raciocínio aproximado, permitindo expressar informações com ambiguidade e subjetividade presentes no raciocínio humano.

A elaboração de um sistema nebuloso depende de alguns elementos: quantidade e tipo de regras, parâmetros da função de pertinência, semântica das regras, operadores do mecanismo de inferência.

O objetivo deste capítulo é apresentar uma breve descrição dos sistemas nebulosos juntamente com seus elementos. Para isso é necessário apresentar, também, alguns conceitos básicos sobre os conjuntos, relações, regras e mecanismos de inferência nebulosa.

3.1 INTRODUÇÃO AOS SISTEMAS NEBULOSOS

O conceito primitivo de conjunto é um agrupamento de elementos com características em comum com fronteiras bem definidas do que pertence ou não ao conjunto. Além disso, a pertinência de um determinado elemento com relação a um conjunto pode ser quantificada como “verdadeira” ou “falsa”. Na representação da álgebra booleana as informações são classificadas como “1” ou “0” (LIMA; PINHEIRO; OLIVEIRA, 2014).

Seja X o universo de discurso da variável denominada genericamente por x. Além disso, seja um conjunto A representado conforme apresentado em (3.1) e definido por “é número ímpar”. Conhecendo o conjunto e o universo dos elementos é possível fazer dois tipos de afirmações “x pertence a A” ou “x não pertence a A”. No caso de $x = 3$, “3 pertence a A” e para $x = 4$ temos que “4 não pertence a A”

$$A = \{\text{x} \mid \text{x é ímpar e x} \in \text{X}\} \tag{3.1}$$

Em conjuntos nebulosos as relações dos elementos como “não pertence” e “pertence” a

um conjunto é, em geral, gradual e varia de zero a um. Desta forma, um conjunto nebuloso pode ser definido da seguinte forma: seja Y o universo de discurso da variável denominada genericamente por y; então um conjunto B é definido por uma coleção de pares ordenados dados pela Equação (3.2).

$$B = \{(\mathrm{y},\, \mu_B(\mathrm{y})) \mid \mathrm{y} \in \mathrm{Y}\} \tag{3.2}$$

O par ordenado apresentado em (3.2) apresenta dois elementos, o primeiro (y) é o objeto em análise e o segundo ($\mu_B(\mathrm{y})$) a função de pertinência que determina qual é o grau que o objeto em análise pertence ao conjunto B.

Todo conjunto nebuloso possui uma função de pertinência, que define o grau de participação de um elemento no conjunto. Para compreender melhor o funcionamento da teoria dos conjuntos nebulosos é necessário compreender aspectos que envolvem as funções de pertinência.

3.1.1 FUNÇÃO DE PERTINÊNCIA

Função de pertinência é a base da teoria dos conjuntos nebulosos. Um conjunto nebuloso A é definido por uma função de pertinência $\mu_A(x)$, responsável em indicar o grau de pertinência de x no conjunto A. Tem-se que $\mu_A(x) \in [0, 1]$ é o valor da função de pertinência definido como um valor entre 0 e 1, inclusive.

A teoria dos conjuntos clássicos, booleanos, pode ser visto como um caso particular da teoria dos conjuntos nebulosos. Os valores atribuídos para a função de pertinência nos conjuntos clássicos podem ser definidos como $\mu_A(x) \in \{0, 1\}$.

Seja um conjunto nebuloso A definido por "número próximo a 5" e um universo contínuo onde $x \in \Re$. A representação da função de pertinência pode ocorrer de diferentes formas variando conforme a concepção adotada. A Equação 3.3 apresenta uma função de pertinência utilizando a concepção de conjuntos clássicos.

$$\mu_A(x) \begin{cases} 0, & \text{se x} \leq 4{,}5 \\ 1, & \text{se } 4{,}5 \leq \text{x} \leq 5{,}5 \\ 0, & \text{se x} \geq 5{,}5 \end{cases} \tag{3.3}$$

A Equação 3.4 apresenta uma função de pertinência utilizando a concepção de conjuntos nebulosos.

$$\mu_A(x) \begin{cases} 0, & \text{se x} \leq 4{,}5 \\ \frac{x-4{,}5}{0{,}5}, & \text{se } 4{,}5 \leq \text{x} \leq 5 \\ \frac{5{,}5-x}{0{,}5}, & \text{se } 5 \leq \text{x} \leq 5{,}5 \\ 0, & \text{se x} \geq 5{,}5 \end{cases} \tag{3.4}$$

A Figura 5 ilustra a diferença no resultado do grau de pertinência de ambas as concepções. A ilustração da concepção clássica o valor da função de pertinência é 0 ou

1, já na concepção nebulosa o valor cresce ou decresce de forma gradual conforme x se aproxime de 5.

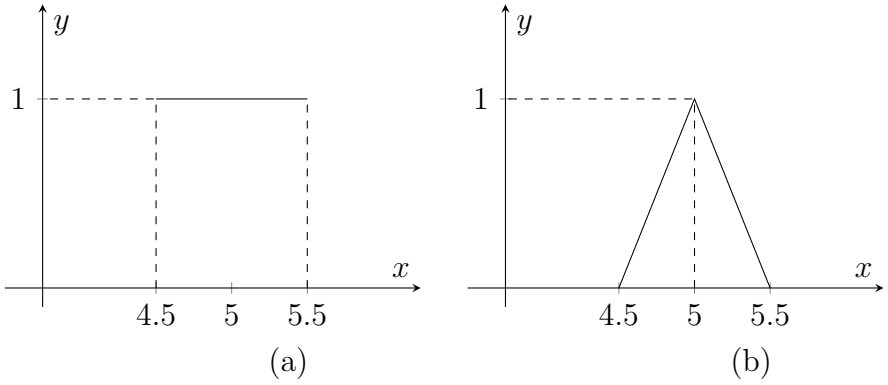

Figura 5 – Função de pertinência: (a) Concepção clássica, (b) Concepção Nebulosa

Os conjuntos nebulosos são representados por funções de pertinência. Considerando a Figura 6, tem-se $x \in \Re$ e os conjuntos A, B e C representando respectivamente os estados: frio, normal e quente. Cada estado representa um conjunto, onde cada um possui uma função de pertinência própria ($\mu_A(x)$, $\mu_B(x)$ e $\mu_C(x)$ respectivamente). A pertinência de um valor no eixo x pode assumir valores de duas funções simultaneamente.

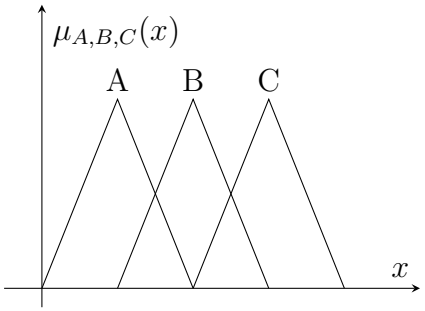

Figura 6 – Exemplo: Conjuntos nebulosos com três estados possíveis

As funções de pertinência ilustradas na Figura 6 apresentam um formato triangular. O formato que uma função de pertinência possui é definido como sua classe. Na literatura a classe de funções mais comuns são: triangular, trapezoidal, Gaussiana, crisp e conjunto unitário (*singleton*) (LIMA; PINHEIRO; OLIVEIRA, 2014).

A Figura 7.a representa a construção de uma função da classe triangular com seus parâmetros. A expressão geral para sua construção é dada por $\mu_{triangular}(x)$ indicado pela Equação (3.5). Sua construção é dada com base nos parâmetros a, b e m, com $a \leq m \leq b$.

$$\mu_{triangular}(x) = \begin{cases} 0 & \text{se } x < a \\ \frac{x-a}{m-a} & \text{se } x \in [a, m] \\ \frac{b-x}{b-m} & \text{se } x \in [m, b] \\ 0 & \text{se } x > b \end{cases} \tag{3.5}$$

A Figura 7.b representa uma função da classe trapezoidal justamente com seus parâmetros. A expressão geral de sua construção é dada por $\mu_{trapezoidal}(x)$ indicado pela expressão (3.6). Sua construção é dada com base nos parâmetros a, b, m e n, com $a \leq m \leq n \leq b$. Um caso particular das funções com formato trapezoidal é quando $m = n$, resultando em uma função com o formato triangular. Pode-se dizer que a classe da função triangular é um caso específico da trapezoidal (JANG; SUN; MIZUTANI, 1997).

$$\mu_{trapezoidal}(x) = \begin{cases} 0 & \text{se } x < a \\ \frac{x-a}{m-a} & \text{se } x \in [a, m] \\ 1 & \text{se } x \in [m, n] \\ \frac{b-x}{b-n} & \text{se } x \in [n, b] \\ 0 & \text{se } x > b \end{cases} \tag{3.6}$$

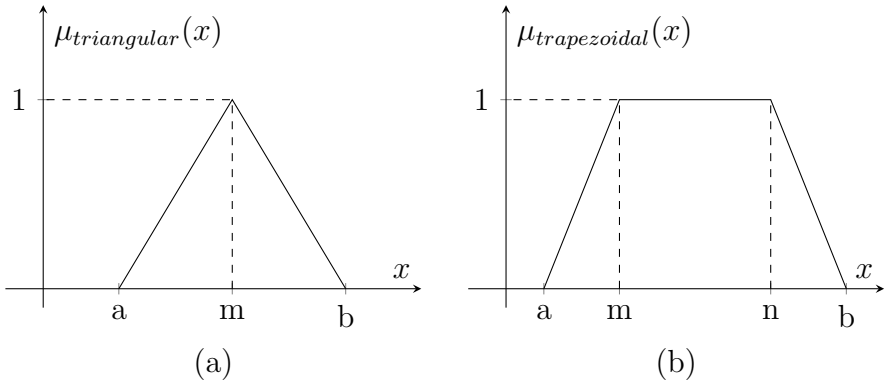

Figura 7 – Função de pertinência triangular

A função ilustrada na Figura 8 representa uma função da classe Gaussiana juntamente com seus parâmetros. A expressão geral para sua construção é dada por $\mu_{gaussiana}(x)$ apresentado pela expressão 3.7. Sua construção é dada com base nos parâmetros m e σ_k, com $\sigma_k \leq 0$.

$$\mu_{gaussiana}(x) = \exp^{-\sigma_k(x-m)^2} \tag{3.7}$$

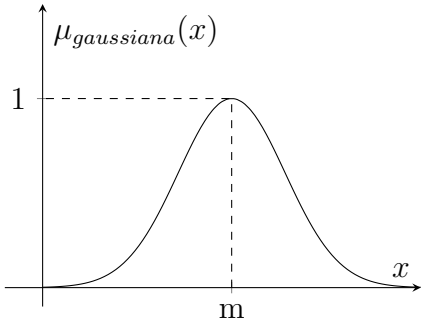

Figura 8 – Função de pertinência Gaussiana

Os conjuntos unitários (*singleton*) e *crisp*, ilustrados na Figura 9, são as funções de pertinência mais simples. A construção do *singleton* é realizado com base na expressão $\mu_{singleton}$ (3.8), com $k > 0$. A função do tipo *singleton* se caracteriza por possuir apenas um único ponto onde o valor da função é diferente de zero.

A construção do *crisp* é realizado com base na expressão μ_{crisp} (3.8), em que $m \leq n$ e $k > 0$. A função se caracteriza por possuir apenas um único segmento constante em k dentro do intervalo $[m, n]$. Um caso particular das funções *crips* se dá na situação em que $m = n$, o resultado é uma função *singleton*. Pode-se dizer que a função da classe singleton é um caso particular da *crips*.

$$\mu_{singleton}(x) = \begin{cases} 0 & \text{se } x < m \\ k & \text{se } x = m \\ 0 & \text{se } x > m \end{cases} \quad \mu_{crisp}(x) = \begin{cases} 0 & \text{se } x < m \\ k & \text{se } x \in [m, n] \\ 0 & \text{se } x > n \end{cases} \tag{3.8}$$

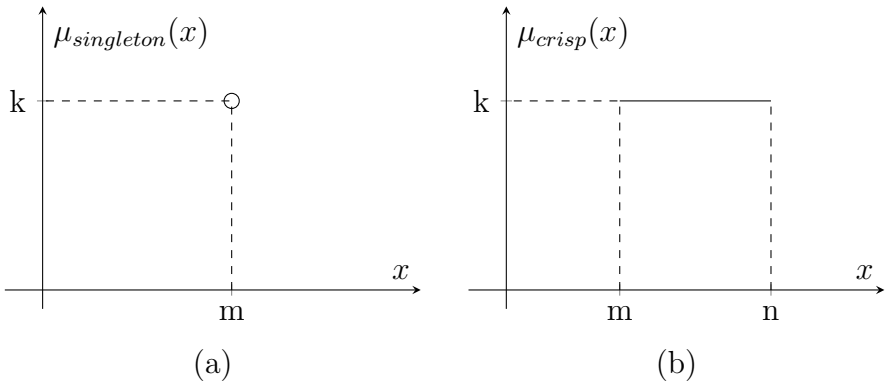

Figura 9 – Função de pertinência: (a) conjuntos unitários, (b) crisp

Observa-se a existência dos diversos tipos de função de pertinência sendo as mais utilizadas as triangulares e as trapezoidais. A escolha do seu tipo é um dos elementos fundamentais para solucionar um dado problema. Cada tipo de função, juntamente com seus respectivos parâmetros possuem características próprias.

No desenvolvimento de um sistema nebuloso os tipos de função e os parâmetros utilizados são definidos por um especialista. Em aplicações muito complexas e abrangentes, a escolha nem sempre é obvia. Outra dificuldade é o caso onde o problema exige mais conhecimento que o especialista possui.

Devido as aplicações mais complexas e a necessidade de um especialista para a construção do sistema, sistemas nebulosos adaptativos foram surgindo, permitindo solucionar problemas que antes não eram viáveis ou possíveis. Nos sistemas nebulosos adaptativos as funções de pertinência são ajustadas com base em valores de entrada e saída.

3.2 REGRAS NEBULOSAS: RACIOCÍNIO APROXIMADO

Nesta Sessão são apresentados outros conceitos necessários para o entendimento sobre os sistemas nebulosos. Os conceitos apresentados são: variáveis linguísticas, regras nebulosas, regra composicional de inferência e raciocínio aproximado.

3.2.1 VARIÁVEIS LINGUÍSTICAS

O conceito de variáveis linguísticas foi apresentado por (ZADEH, 1965) como uma proposta para modelar o pensamento humano. A abordagem apresenta como organizar e estruturar as informações resultando em termos de conjuntos nebulosos.

A introdução de variáveis linguísticas permite definir as informações, que anteriormente eram números, em palavras. A teoria dos conjuntos nebulosos permite caracterizar as informações em diversos conjuntos, sendo atribuída para cada conjunto existente uma definição formal para uma variável linguística (JANG; SUN; MIZUTANI, 1997).

Conforme apresentado por Jang, Sun e Mizutani (1997), uma variável linguística é caracterizada pela quíntupla:

$$(x, T(x), X, G, M), \text{ onde} \tag{3.9}$$

x: indica o nome da variável linguística;

$T(x)$: conjunto de termos linguísticos de x;

X: universo de discurso da variável x;

G: regra sintática que gera os termos de $T(x)$;

M: regra semântica que associa para cada valor linguístico A o seu significado $M(A)$, onde $M(A)$ é um conjunto *fuzzy* em X.

Por exemplo, considerando uma variável linguística $x = temperatura$ no universo $X = [0, 50]$, o conjunto de termos para a variável *temperatura* poderia ser:

$$T(temperatura) = \{\text{muito-baixa, baixa, pouco-baixa, media, pouco-alta, alta, muito-alta}\} \tag{3.10}$$

A gramática G define como os termos primários da variável *temperatura* {baixa, média, alta} são associados ao modificadores {muito, pouco, maior, menor, ou, não}. A regra semântica M associa cada rótulo do termo $T(temperatura)$ com um conjunto nebuloso no universo de discurso X. O conjunto nebuloso no qual os termos são associados são normalmente nos formatos: triangular, trapezoidal e Gaussiana.

A Figura 10 apresenta como é realizada a construção das funções de pertinência dos conjuntos nebulosos com suas respectivas variáveis linguísticas.

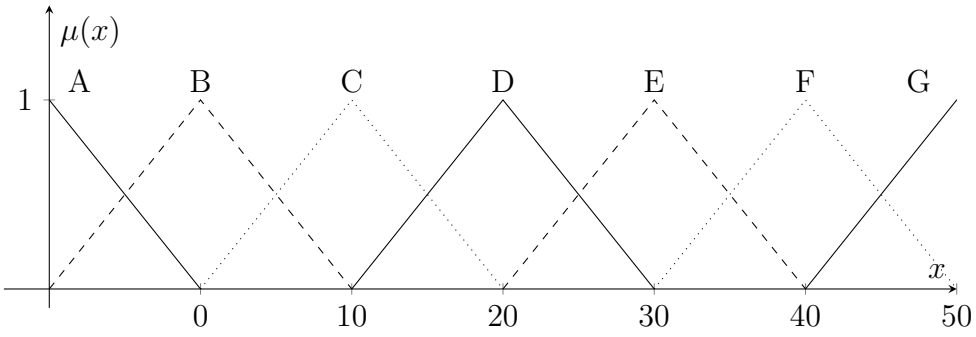

Figura 10 – Exemplo: Variáveis linguísticas de temperatura: (A) muito baixa, (B) baixa, (C) pouco baixa, (D) média, (E) pouco alta, (F) alta, (G) muito alta

3.2.2 REGRAS NEBULOSAS

As regras nebulosas, conhecidas como regras nebulosas "se-então", implicações nebulosas, declaração condicional nebulosa, dentre diversas nomenclaturas, são a apresentação formal de diretivas e estratégias. A construção de regras nebulosas são apropriadas para inserir conhecimento de um especialista humano em um sistema para solucionar ou otimizar um dado problema.

A formação geral de uma regra nebulosa é dividida em antecedente (premissa) e consequente (conclusão), sua construção geral é apresentado conforme a Equação 3.11, com o antecedente o e consequente representados na forma de proposições.

$$\text{se } <\text{antecedente}> \text{ então } <\text{consequente}> \tag{3.11}$$

Conforme apresentado por Jang, Sun e Mizutani (1997), a construção de expressões no formato das regras nebulosas ocorre frequentemente como, por exemplo:

- Se o tomate é vermelho, então ele está maduro
- Se o dia está ensolarado, então abrir a janela

Assumindo como antecedente o termo "x é A" e como consequente "y é B" a regra nebulosa resulta na Equação 3.12, sendo A e B valores linguísticos definidos pelos conjuntos nebulosos no universo de discurso X e Y, respectivamente, e $x \in X$ e $y \in Y$ (JANG; SUN; MIZUTANI, 1997). Outra forma de representação para a Equação 3.12 é o formato "$R : A \rightarrow B$", sendo R é uma relação nebulosa binária no produto cartesiano $X \times Y$.

$$\text{se } x \text{ é } A \text{ então } y \text{ é } B \tag{3.12}$$

A função de pertinência da relação nebulosa R, bi-dimensional, pode ser escrita conforme a Equação 3.13, com $a = \mu_A(x)$, $b = \mu_B(y)$ e f uma função na forma $f : [0, 1]^2 \rightarrow [0, 1]$

que especifica a semântica da regra $R : A \rightarrow B$. Observa-se que o número de termos empregados para a função μ_R é equivalente ao número de dimensões da relação nebulosa.

$$\mu_R(x, y) = f(\mu_A(x), \mu_B(x)) = f(a, b) \tag{3.13}$$

A representação das regras nebulosas "se <antecedente> então <consequente>", por motivos de simplicidade, foi realizada utilizando apenas uma variável no antecedente "X é A" e uma no consequente "Y é B". Nos diversos problemas onde se aplica a teoria de conjuntos nebulosos, a composição das regras se torna mais complexa, podendo conter m e n variáveis no antecedente e no conseguente, respectivamente. Então, a construção da regra pode ser apresentada conforme a expressão (3.14).

$$\text{se } \alpha \text{ então } \beta \tag{3.14}$$

onde α e β podem ser, por exemplo, conjunções:

$\alpha = X_1$ é A_1 E X_2 é A_2 E ... E X_m é A_m;

$\beta = Y_1$ é B_1 E Y_2 é B_2 E ... E Y_n é B_n.

A construção do antecedente e consequente não precisa ser necessariamente da forma conjuntiva, elas podem ser da forma disjuntiva ("OU"). Outro aspecto que deve ser mencionado é a forma como as regras são obtidas. Conforme apresentado por Lima, Pinheiro e Oliveira (2014), as principais formas de obtenção das regras são:

- Informação de especialista;
- Análise qualitativa de dados;
- Técnica de regressão, método do mínimos quadrados;
- Métodos de otimização;
- Técnicas de agrupamento;
- Métodos de treinamento, aprendizado de máquina;
- Associação com outras técnicas de IA.

3.2.3 RACIOCÍNIO APROXIMADO

A inferência da base de regras na lógica tradicional de dois valores é o *modus ponens*, que utiliza fatos e regras para obter uma conclusão. A interpretação da informação é realizada de forma binária (verdadeiro ou falso). Por exemplo, uma implicação ($A \rightarrow B$) "se o tomate está vermelho, então ele está maduro", se o antecedente "o tomate está vermelho"

(A) for verdade pode-se assumir que o consequente (B) "ele está maduro" também é verdade, conforme apresentado pela expressão (3.15):

Modus Ponens:

(Fato)	X é A	(tomate está vermelho)
(Regra)	$A \rightarrow B$	(se o tomate está vermelho, então ele está maduro)
(Conclusão)	Y é B'	(o tomate está maduro)

(3.15)

Conforme apresentado por Jang, Sun e Mizutani (1997), no raciocínio humano o *modus ponens* é empregado com aproximação, conhecido como *modus ponens* generalizado. A base para seu funcionamento é o mesmo, utilizando fatos e regras para obter uma conclusão. A interpretação da informação ocorre com aproximações, atribuindo valores no intervalo de [0, 1].

Por exemplo, para a regra "se o tomate está vermelho, então ele está maduro" os fatos pode-se ter a informação expressa de diversas formas, como: "o tomate não está vermelho", "o tomate está pouco vermelho" e "o tomate está muito vermelho", podendo concluir que "ele não está maduro", "ele está pouco maduro" e "ele está muito maduro", receptivamente. Considerando a mesma implicação ($A \rightarrow B$) "se o tomate está vermelho, ele está maduro" e um fato (A') "o tomate está pouco vermelho", é possível realizar a inferência do *modus ponens* generalizado da seguinte forma:

Modus Ponens:

(Fato)	X é A'	(tomate está pouco vermelho)
(Regra)	$A \rightarrow B$	(se o tomate está vermelho, então ele está maduro)
(Conclusão)	Y é B'	(o tomate está pouco maduro)

(3.16)

3.3 SISTEMAS NEBULOSOS

Conforme apresentado por Wang (1999), sistemas nebulosos utilizam conhecimento juntamente com informações imprecisas para obter conclusões. O conhecimento é representado por meio de regras nebulosas "se-então" (*if-then*) e as imprecisões das informações são representadas por palavras como "muito" e "pouco", ou expressas por valores numéricos no intervalo de [0, 1].

Sistema nebuloso, conhecido também como sistema de inferência nebulosa, é uma ferramenta que toma como base a teoria dos conjuntos nebulosos, regras nebulosas e raciocínio nebuloso. Sua área de aplicação é extensa, como sistemas de automação, classificação de dados, análise de decisões, sistemas especialistas e reconhecimento de padrões (JANG; SUN; MIZUTANI, 1997).

Um sistema nebuloso (sistema *fuzzy*) consiste em três componentes básicos:

- Base de regras: composto pelas regras *fuzzy*;
- Base de dados: define os elementos das funções utilizadas nas regras nebulosas;
- Sistema de inferência: objetivo de obter conclusões a partir de fatos e regras, utilizando o processo de inferência para informações nebulosas.

3.3.1 SISTEMA DE INFERÊNCIA

Como foi apresentado na Seção 3.2.3, o raciocínio nebuloso, em uma abordagem mais simples, é a obtenção de uma conclusão a partir de um fato e uma regra. A construção de forma geral de um raciocínio nebuloso é dado por múltiplos fatos, regras e a obtenção de múltiplas conclusões. Como apresentado por Delgado (2002), sua representação é apresentada conforme a Expressão 3.17, sendo que n é o número de antecedentes, s é o número de consequentes e m é o número de regras.

$$\begin{array}{ll} (\text{Fato } F_i) & X_1 \text{ é } A_1 \text{ E } X_2 \text{ é } A_2 \text{ E } \ldots \text{ E } X_n \text{ é } A_n \\ (\text{Regra } R_1) & \text{Se } X_1 \text{ é } A_1^1 \text{ E } \ldots \text{ E } X_n \text{ é } A_n^1 \text{ então } Y_1 \text{ é } B_1^1 \text{ E } \ldots \text{E } Y_s \text{ é } B_s^1 \\ & \vdots \\ (\text{Regra } R_m) & \text{Se } X_1 \text{ é } A_1^m \text{ E } \ldots \text{ E } X_n \text{ é } A_n^m \text{ então } Y_1 \text{ é } B_1^m \text{ E } \ldots \text{E } Y_s \text{ é } B_s^m \\ \hline (\text{Conclusão } C_0) & Y_1 \text{ é } B_1 \text{ E } Y_2 \text{ é } B_2 \text{ E } \ldots \text{ E } Y_s \text{ é } B_s \end{array} \tag{3.17}$$

Para realizar a extração da conclusão em um raciocínio nebuloso é utilizado um sistema de inferência nebulosa.

Modelos de sistemas nebulosos, ou sistemas de inferência nebulosa, são técnicas que utilizam os mecanismos de inferência para a obtenção de conclusões. Dentre os modelos existentes os mais conhecidos são os modelos de Mamdani (MAMDANI; ASSILIAN, 1975) e Takagi-Sugeno (TAKAGI; SUGENO, 1985).

O que diferencia os modelos, além da forma de obtenção da saída é a forma em que os resultados são apresentados. Por exemplo: no modelo de Takagi-Sugeno a saída obtida pelo sistema é uma média ponderada das regras nebulosas, que possuem como consequente polinômios, já no modelo de Mamdani a saída obtida é uma saída nebulosa, onde é necessário realizar a transformação da saída nebulosa em uma saída não nebulosa.

A seguir é descrito com mais detalhes o modelo de Takagi-Sugeno, que é utilizado no desenvolvimento do presente trabalho.

3.3.1.1 MODELO DE TAKAGI-SUGENO

A estrutura das regras em um modelo Takagi-Sugeno (TAKAGI; SUGENO, 1985) é baseada em antecedente composto por variáveis linguísticas e conseguente representado por uma função. A estrutura geral das regras no modelo de Takagi-Sugeno é apresentada na

expressão (3.18), com A_i representando os conjuntos nebulosos, x_i as variáveis de entrada do sistema, n indica o número componentes do antecedente e Y representa o polinômio de saída.

$$\text{se } x_1 \text{ é } A_1 \text{ e } x_2 \text{ é } A_2 \text{ e } \ldots \text{ e } x_n \text{ é } A_n \text{ então } Y \tag{3.18}$$

O antecedente é a composição de n elementos e o consequente é um polinômio, normalmente linear. Outra forma de representação é dada por meio de vetores, onde $\overrightarrow{x} = [x_1, x_2, \ldots, x_n]$ representa as variáveis de entrada do sistema e $\overrightarrow{w} = [w_0, w_1, \ldots, w_j]$ representa os pesos presentes no polinômio, onde n é o número de variáveis de entrada e j o número de pesos do polinômio.

É importante salientar que a quantidade de pesos presentes no polinômio, e consequentemente, o tamanho do vetor $\overrightarrow{w}$, varia conforme o ordem do polinômio empregado. Os tipos de polinômios mais comuns no modelo de Takagi-Sugeno são: função de ordem constante ($j = 1$) e linear ($j = n + 1$). A representação geral de polinômio de ordem linear para os consequentes das regras é dado pela equação (3.19) (CORDÓN, 2001).

$$Y = f(\overrightarrow{w}, \overrightarrow{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_j x_j \tag{3.19}$$

O funcionamento geral do modelo de Takagi-Sugeno ocorre conforme apresenta a Figura 11. A base de conhecimento é responsabilizada em calcular o valor de pertinência das regras para o padrão de entrada e o cálculo do consequente Y de cada regra. A etapa da média ponderada é responsável em realizar a média ponderada dos valores de Y conforme o valor de pertinência da regra. A resposta obtida da operação de média ponderada é o resultado do sistema para o padrão de entrada x

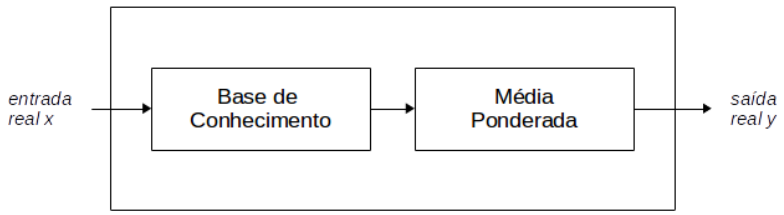

Figura 11 – Estrutura básica do modelo Takagi-Sugeno

Outra forma de representar o funcionamento é por meio da Equação 3.20, sendo que $\overrightarrow{x}_p$ é o p-ésimo padrão de entrada, $\mu_j(\overrightarrow{x}_p)$ é o valor da função de pertinência, $f_j(\overrightarrow{w}_j, \overrightarrow{x}_p)$ é o resultado do polinômio e j referencia a regra.

$$y(\overrightarrow{x}_p) = \frac{\sum_{j=1}^{n} \mu_j(\overrightarrow{x}_p) f_j(\overrightarrow{w}_j, \overrightarrow{x}_p)}{\sum_{j=1}^{n} \mu_j(\overrightarrow{x}_p)} \tag{3.20}$$

Além da representação da função nos consequentes das regras por meio de polinômios lineares é possível utilizar outras formas de representação como funções não lineares ou equação diferenciais. Considerando um polinômio não linear para a regra R_j apresentado como $f_j(\overrightarrow{w}_j, \overrightarrow{x}_p)$, com ordem $q = \frac{n(n-1)}{2} + 2n + 1$, a função no consequente da regra pode ser expressa conforme a Equação 3.21.

$$f_j(\overrightarrow{w}_j, \overrightarrow{x}_p) = w_{j0} + w_{j1}x_1 + ... + w_{jn}x_n + w_{j(n+1)}x_1x_1 + w_{j(n+2)}x_1x_2 + ...w_{jq}x_nx_n \quad (3.21)$$

Os valores da saída $\overrightarrow{y}$ estão diretamente relacionados ao valor das funções de pertinência e dos pesos dos polinômios de cada regra. O processo de obtenção dos pesos do polinômio é apresentado de diversas formas. O método utilizado no presente trabalho é o método local de otimização de consequente, que considera uma regra independente da outra (HOFFMANN; NELLES, 2000)

É possível obter w_j pelo método dos quadrados mínimos ponderado, conforme apresentado por Delgado (2002), a formulação resulta em

$$\min_{w_j} \frac{1}{2}||\overrightarrow{y} - \overrightarrow{y}_d||^2_{\Psi_j} = \min_{w_j} \frac{1}{2}[(\overrightarrow{y} - \overrightarrow{y}_d)^T \Psi_j (\overrightarrow{y} - \overrightarrow{y}_d)] \quad (3.22)$$

onde Ψ é a matriz diagonal definida como

$$\Psi_j = \begin{bmatrix} \beta_j^1 & 0 & 0 & \dots & 0 \\ 0 & \beta_j^2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \beta_j^N \end{bmatrix} \quad (3.23)$$

O resultado dos pesos do polinômio para a regra R_j é apresentado pela Equação 3.24, onde w_j é o vetor de pesos, $\overrightarrow{y}_d$ é o vetor de saída desejada para cada padrão de entrada

$$w_j = (\Lambda_j^T \Psi_j \Lambda_j)^{-1} \Lambda_j^T \Psi_j \overrightarrow{y}_d \quad (3.24)$$

4 PROJETO AUTOMÁTICO DE SISTEMAS NEBULOSOS CO-EVOLUTIVOS

Este capítulo descreve a proposta apresentada por Delgado (2002). Trata-se de um algoritmo genético co-evolutivo hierárquico (CoevolGFS).

O CoevolGFS utiliza com base três trabalhos presentes na literatura. O esquema de evolução hierárquica é baseado no trabalho de Delgado, Zuben e Gomide (2001), o processo co-evolutivo é inspirado no paradigma de hierarquia apresentado por Moriarty e Miikkulainen (1998) e do modelo de co-evolução cooperativa apresentado por Potter e Jong (2000).

Uma solução de um problema pode ser codificada e interpretada como diversas soluções parciais, cada solução parcial é representada por uma espécie distinta. As populações hierárquicas apresentam como proposta um modelo para organizar e padronizar a forma como ocorrem as relações entre os indivíduos das diferentes populações. A relação entre as populações podem ser tanto por dependência de valor, valores de um indivíduo de uma população dependam de outro indivíduo de outra população, quanto por dependência de *fitness*, onde o valor de *fitness* dos indivíduos de uma população depende dos valores de *fitness* dos indivíduos de outra população (DELGADO, 2002).

4.1 A ABORDAGEM CO-EVOLUTIVA

A proposta de Delgado (2002) apresenta uma abordagem co-evolutiva entre quatro espécies diferentes, organizado de forma hierárquica. Cada espécie codifica um parâmetro do sistema nebuloso, e são agrupadas por populações distintas. Os elementos codificados pelas populações são:

- População Nível I: Partição nebulosa
- População Nivel II: Regras individuais
- População Nivel III: Base de regras
- População Nivel IV: Sistema *Fuzzy*

A Figura 12 ilustra de forma superficial como ocorre a cooperação entre os indivíduos de cada nível e como cada população interage com o domínio da aplicação. A seguir é apresentado de forma mais detalhada como ocorre a cooperação entre os níveis hierárquicos.

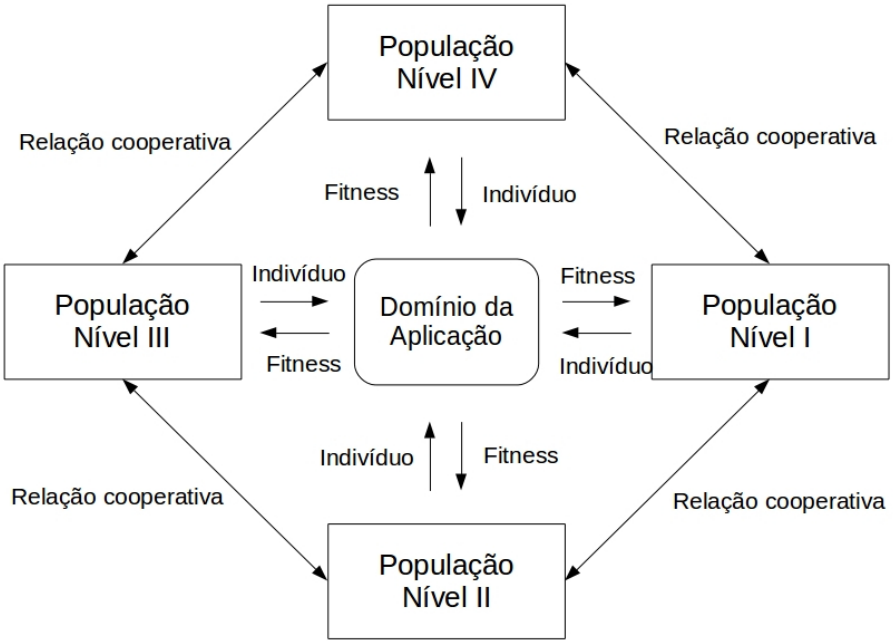

Figura 12 – Sistema hierárquico

4.2 HIERARQUIA E COOPERAÇÃO

A cooperação é a relação existente entre dois indivíduos de diferentes espécies, sendo que uma auxilia a outra a solucionar um determinado problema. No caso do CoevolGFS o sistema de cooperação ocorre diversas vezes entre diferentes níveis hierárquicos.

A população de partições nebulosas é composta por indivíduos que representam as funções de pertinência dentro do universo das variáveis envolvidas. A codificação da função é feita por valores inteiros e reais.

Os indivíduos da população de regras individuais são representações de regras nebulosas dadas a partir das funções de pertinência, indivíduos da população de partições nebulosas.

A regra referencia a função de pertinência por meio de índices. Os indivíduos da população de base de regras são formados por índices relacionados aos indivíduos da população de regras individuais.

A população de sistemas *fuzzy* é composta por indivíduos que representam o sistema *fuzzy* como um todo. Sua codificação é por meio de valores inteiros, reais e o índice de um indivíduo da população de base de regras.

A descrição mais detalhada sobre como é realizado a codificação de cada parâmetro é apresentado na Seção 4.3

4.3 CODIFICAÇÃO

No CoevolGFS são identificados quatro tipos de indivíduos pertencentes a quatro populações distintas. A forma como os indivíduos são codificados é diferenciado para cada

um dos níveis hierárquicos. Esta seção tem como objetivo apresentar como é a codificação de cada indivíduo e como é representado a relação entre os indivíduos.

Os indivíduos da população de sistemas nebulosos (indivíduos de nível IV) são representados por cromossomos de codificação mista, ou seja, possuem mais de um tipo de valor. O cromossomo dos indivíduos de nível IV é codificado por meio de nove alelos com codificação mista, seus alelos são:

1. Inferência (Escalonada)
2. Agregação dos antecedentes (t-norma)
3. Parâmetro p_t
4. Semântica de regra (norma)
5. Parâmetro associado
6. Agregação das regras
7. Defuzzyficação
8. Índice base de regras
9. Índice da partição nebulosa

A forma de codificação apresentada pela autora é abrangente, codificando um sistema nebuloso para os modelos de Takagi-Sugeno e para o modelo Mamdani. Para o modelo de Mamdani os alelos nos loci 1 a 7 são representações que codificam o processo de inferência, enquanto para o modelo de Takagi-Sugeno são utilizados apenas os alelos nos loci 1 a 3, os alelos localizados nos loci 4 a 7, sendo específicos para o tratamento do modelo de Mamdani.

O alelo localizado no locus 8 identifica a base de regras utilizada pelo sistema, sua codificação é um índice que referencia um indivíduo da população de nível III. O último alelo, localizado no locus 9, representa a partição nebulosa utilizada pelo sistema (DELGADO, 2002).

Os indivíduos da população de nível III possui tamanho fixo com codificação inteira. Cada alelo do cromossomo pode representam uma regra individual (indivíduo de nível II) ou pode ser vazia, permitindo reduzir o número de regras utilizadas pelo sistema. A quantidade de alelos é definido pelo número máximo de regras permitidas.

A codificação dos cromossomos de nível II é caracterizada por possuir tamanho fixo e codificação inteira. A quantidade de alelos é igual a quantidade de dimensões do problema tratado. Cada alelo do cromossomo representa o índice de uma função de pertinência (indivíduo de nível I) ou pode ser vazio, permitindo a construção de regras individuais mais genéricas e simplificadas.

Os indivíduos da população de nível I são representações das funções de pertinência. Os cromossomos dos indivíduos são caracterizados por possuírem uma sequência de cinco alelos de codificação real.

O parâmetro S_k indica o tipo da função de pertinência, podendo adotar os valores: 1 (trapezoidal), 2 (triangular) e 3 (Gaussiana). Os parâmetros C1, C2, L e R são aplicados para a construção da função de pertinência

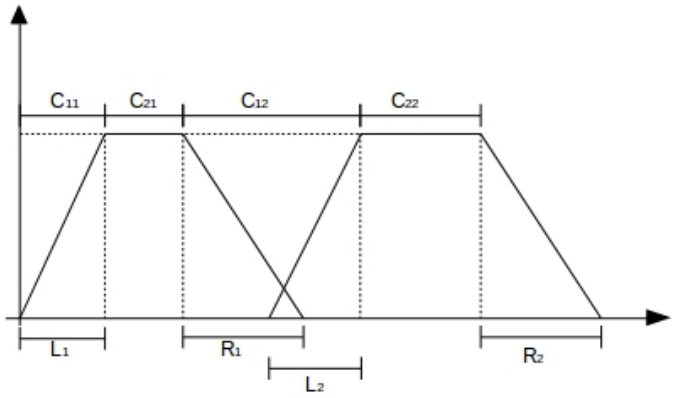

Figura 13 – Diagrama Geral do Algoritmo Genético

A construção da função de pertinência é feita com base nos parâmetros dos indivíduos de nível I. A Figura 13 ilustra como é realizado a construção da função. Os três tipos de funções são construídos através dos mesmos parâmetros, a função trapezoidal é construída de forma direta, a triangular e Gaussiana são construídas através de operações simples. A Figura 14 apresenta como é realizado a construção das funções triangulares e Gaussiana. Em ambas as construções são apresentados os pontos a, b e uma distância m. Os pontos a e b, são pontos não relativos, diferente da forma como é codificada no cromossomo, com $a = origem + C1$ e $b = a + C2$. A variável origem é o resultado da soma de todos os valores de $C1$ e $C2$ das funções de pertinência que antecedem a função. A distância m é a média dos pontos a e b, onde $m = \frac{a+b}{2}$. Na construção da função Gaussiana é necessário definir a dispersão $\sigma = D/3$, onde $D = \frac{L+R}{2}$.

O resultado da codificação dos quatro níveis hierárquicos no modelo de Takagi-Sugeno é apresentada na Figura 15.

4.4 ALGORITMO GENÉTICO CO-EVOLUTIVO

O algoritmo CoevolGFS pode ser dividido em cinco etapas principais, sendo elas:

- Inicialização da população;
- Otimização dos consequentes;
- Cálculo do *fitness*;

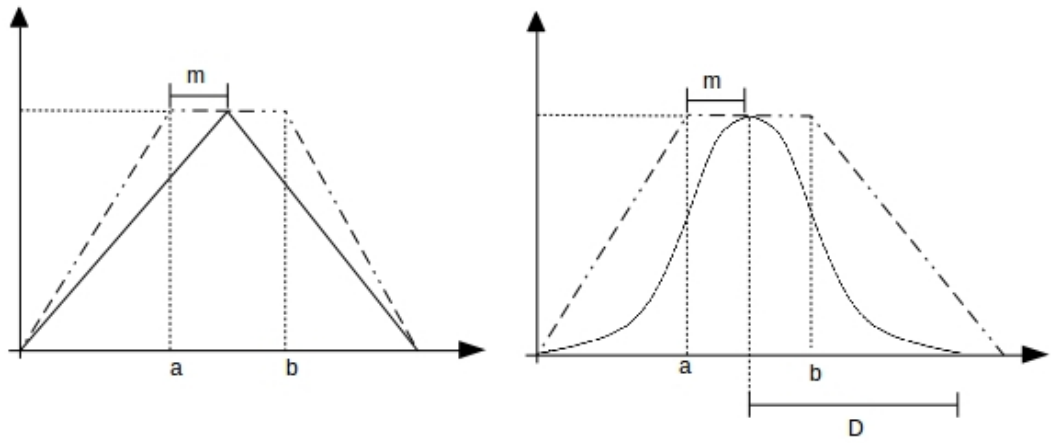

Figura 14 – Diagrama Geral do Algoritmo Genético

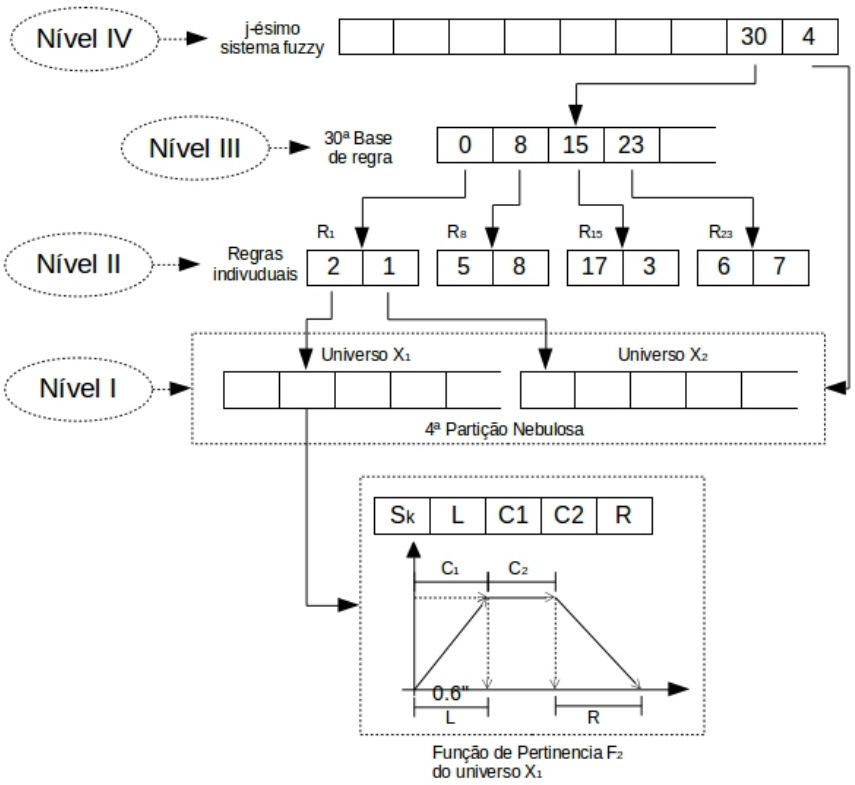

Figura 15 – Codificação Hierarquia

- Critério de parada;
- Reprodução.

O diagrama presente na Figura 16 ilustra como é o funcionamento geral do algoritmo genético.

A primeira etapa é a inicialização da população, é o momento onde as quatro populações são inicializadas. A criação de cada indivíduo deve ser feita respeitando os limites impostos

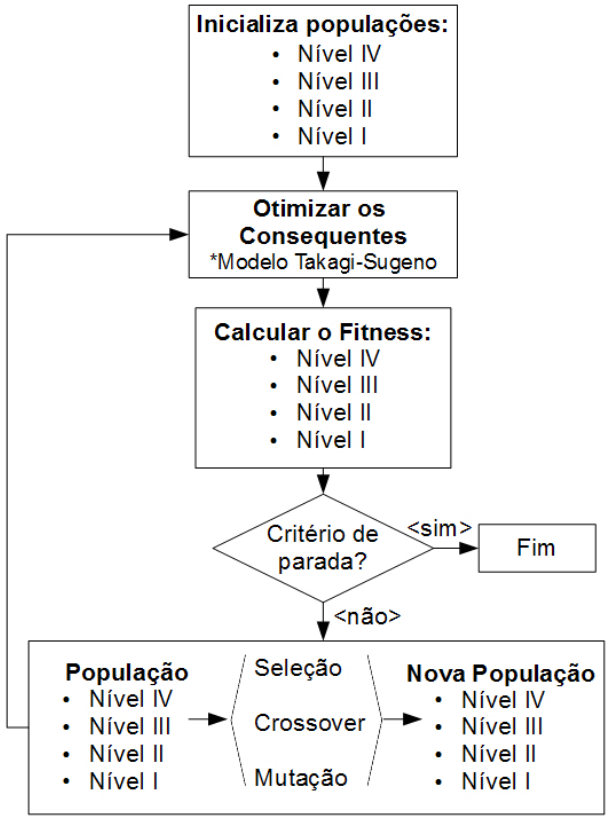

Figura 16 – Diagrama Geral do Algoritmo Genético

pela codificação de cada um, assim como deve ser respeitado a estrutura de dependência hierárquica existente entre cada um dos níveis.

A inicialização das populações é na ordem crescente. A seguir é descrito as peculiaridades de cada um dos níveis:

- Nível I: As partições nebulosas são inicializadas, juntamente com suas respectivas funções de pertinência. A inicialização de uma função de pertinência consiste em definir valores aleatórios para cada um de seus atributos: S_k, $C1$, $C2$, L, R. Para o atributo S_k é atribuído os valores referentes ao tipo de função de pertinência: 1(trapezoidal), 2(triangular) ou 3(Gaussiana). O valor escolhido para $C1$ é o mais adequado para distribuir de maneira equidistante as funções. Os demais valores ($C2$, L, R) são definidos de forma aleatória conforme a granularidade do universo em questão.

- Nível II: As regras individuais são codificações inteiras, cada alelo do cromossomo representa um índice que faz referência a um indivíduo da população de nível I. Os valores são gerados de forma aleatória respeitando o intervalo de valores permitido $[1, SP_1]$, onde SP_1 é o tamanho da população de nível I. Deve ser possível atribuir valores nulos aos parâmetros, permitindo gerar regras mais simplificadas.

- Nível III As bases de regra são codificações inteiras de tamanho fixo, cada alelo do cromossomo representa um índice que faz referência a um indivíduo de nível II. Os valores são gerados de forma aleatória respeitando o intervalo de valores permitido $[1, SP_2]$, onde SP_2 é o tamanho da população de nível II. Deve ser possível atribuir valores nulos aos parâmetros, permitindo gerar bases de regra reduzidas.

- Nível IV: O sistema *fuzzy* são codificações mistas de tamanho fixo. No caso do trabalho apresentado por Delgado (2002) é considerado apenas os alelos 2, 3, 8 e 9, já que é aplicado ao modelo de Takagi-Sugeno. A inicialização do do locus 2 é um índice que referencia as t-normas, o valor é um inteiro inicializado de forma aleatória respeitando o intervalo $[1, 9]$. O locus 3 é iniciado com o valor $p_t = 2$. Os alelos assim como os demais índices de referência, são codificações inteiras que fazem referência a base de regras e a partição nebulosa, respectivamente, cada um deles deve respeitar o intervalo de índices existentes.

A segunda etapa do algoritmo é a otimização dos parâmetros dos consequentes. A otimização é aplicada apenas para os sistemas *fuzzy* no modelo de Takagi-Sugeno. A operação de otimização é realizada para cada um dos indivíduos de nível IV. Mais detalhes sobre a otimização dos consequentes será apresentado na Seção 4.7

A terceira etapa do algoritmo consiste em calcular o desempenho de cada um dos sistemas nebulosos (indivíduos da população de nível IV), o desempenho é relacionado ao domínio da aplicação. Para cada indivíduo de cada população é atribuído o valor de *fitness*. A Seção 4.6 apresenta mais detalhes de como é realizado o cálculo do *fitness* para cada um dos diferentes níveis hierárquicos

A quarta etapa é o instante que é verificado as condições de parada do algoritmo, podendo ser quando o algoritmo atinge o número máximo de gerações ou critério de erro. Caso a condição de parada seja atendida o algoritmo é encerrado, caso contrário é dado continuidade.

A quinta e última etapa do algoritmo é relacionado aos operadores evolutivos. A evolução das populações ocorre do nível mais alto para o mais baixo, ou seja, os operadores de seleção, *crossover* e mutação são aplicado inicialmente em todos os indivíduos da população de sistemas *fuzzy*. Quando finalizado em um nível descem para o próximo nível na hierarquia, repetindo a operação até o nível I.

4.5 OPERADORES EVOLUTIVOS

A elaboração do CoevolGFS é a junção do algoritmo genético com sistema *fuzzy*, codificando níveis hierárquicos com evolução cooperativa. No desenvolvimento do algoritmo genético é necessário definir os operadores evolutivos que serão utilizados, são eles: seleção, *crossover* e mutação.

O operador evolutivo de seleção é aplicado no algoritmo de forma híbrida, ou seja, selecionando os indivíduos por meio de dois critérios. O primeiro critério de seleção é a forma clássica do algoritmo genético, a forma de avaliar os indivíduos é pelo *fitness*. O segundo critério de seleção é baseado na diversidade. Além dos dois mecanismos de seleção é aplicado o elitismo, no qual o indivíduo com maior *fitness* sempre sobrevive.

A seleção primária é responsável em selecionar 80% dos indivíduos por meio da seleção por torneio. O torneio é realizado da seguinte forma, 10% dos indivíduos da população são escolhidos aleatoriamente para participar do torneio e o melhor indivíduo dentre os escolhidos é selecionado. Para cada indivíduo que deve ser selecionado um novo torneio é realizado.

A seleção secundária consiste em selecionar 20% dos indivíduos por meio da seleção por diversidade, o objetivo é selecionar os indivíduos mais diferentes quando comparados com o melhor indivíduo da população. A relação de diferença é dada pela distância Euclidiana para as codificações reais, e distância de Hamming para as codificações inteiras.

O operador evolutivo de *crossover* realiza a operação utilizando a técnica de crossover simples. Consiste em selecionar um par de indivíduo (pais) e escolher um ponto no cromossomo de forma aleatória. O ponto definido é o local onde ocorre a troca de informação no cromossomo. Para cada nova operação de *crossover* é escolhido um novo ponto de troca.

O operador evolutivo de mutação apresenta uma pequena probabilidade de realizar trocas aleatórias em alguns indivíduos. No momento em que ocorre a mutação um alelo é alterado de forma aleatória, essa troca de valor deve respeitar o universo de valores do locus onde o alelo se encontra. Delgado (2002) define duas formas de mutação:

- Escolha aleatória do ponto de mutação (EA): todos os pontos de todos dos indivíduos têm a mesma probabilidade de sofrerem mutação.
- Escolha do ponto de mutação baseado no *fitness* (EF): no sistema hierárquico algumas espécies podem ser escolhidas com base em seus *fitness*, permitindo favorecer a mutação em indivíduos piores. Os indivíduos de nível inferior possuem o *fitness* associado aos indivíduos de nível superior (o cálculo do *fitness* é apresentado de forma mais detalhada na Seção 4.6).

A forma como é definida a mutação realizada em cada uma das gerações é feita de modo aleatório. No início de cada geração um valor de zero a um é sorteado, caso maior ou igual a 0,5 é utilizado a "Escolha aleatória do ponto de mutação", caso contrário, é utilizado a "Escolha do ponto de mutação baseado no *fitness*".

O resultado das operações genéticas podem influenciar o elitismo, o melhor indivíduo de uma população pode sofrer alteração mesmo que separado na nova população. O que provoca tal fenômeno é relacionado as operações genéticas dos níveis inferiores, alterando indivíduos que participem da hierarquia do melhor indivíduo, mesmo que ocorra de forma

direta ou indireta. Deve-se garantir que as operações realizadas nos níveis inferiores não interfiram nos resultados do melhor indivíduo selecionado pelo elitismo.

4.6 AVALIAÇÃO DO *FITNESS*

O valor de *fitness* de cada espécie é avaliado de forma diferente. No nível IV para realizar o cálculo do valor de *fitness* é decodificado o sistema *fuzzy* e é avaliado seu desempenho. O desempenho é dado com base nos valores obtidos pelo sistema e nos valores corretos, verificando o erro apresentado. O cálculo do *fitness* dos indivíduos de nível IV é o resultado da Equação 4.1, sendo que o EQM representa o erro quadrático médio.

$$F(SN_i) = \frac{1}{\sqrt{EQM}} \tag{4.1}$$

O valor dos indivíduos de nível inferior (I, II, III) é dado com base no *fitness* dos indivíduos de nível superior:

- Base de Regras (nível III): $F(BR_k) = max(F(SN_b), \ldots, F(SN_d))$, $b, \ldots, d$ representam os sistemas nebulosos que utilizam a base de regras k;
- Regras Individuais (nível II): $F(RI_j) = mdia(F(BR_m), \ldots, F(BR_n))$, $m, \ldots, n$ representam as bases de regras que utilizam a regra individual j;
- Partição Nebulosa (nível I): $F(PN_q) = max(F(SN_x), \ldots, F(SN_z))$, $x, \ldots, z$ representam os sistemas nebulosos que utilizam a partição nebulosa q.

Nota-se que a forma como é realizado o cálculo do *fitness* nos indivíduos de nível III e I é apresentado pelo operador max e os indivíduos de nível II é dado com o operador $mdia$. A justificativa apresentada por Delgado (2002) é que um sistema nebuloso (nível IV) que apresenta um bom resultado é devido a participação da base de regras (nível III) e da partição nebulosa (nível I). O valor de *fitness* de uma base de regras ou uma partição nebulosa não deve ser prejudicado devido a participações em sistemas nebulosos de baixo desempenho. Já no caso das regras individuais (nível II) a avaliação do *fitness* com base na média é mais indicado, já que a regra individual participa da base de regras juntamente com outras regras. A participação de uma regra individual em uma base de regras dificilmente corresponde pelo *fitness* do sistema nebuloso.

4.7 OTIMIZAÇÃO DO CONSEQUENTE

A solução de um problema utilizando sistemas nebulosos permite utilizar modelos para solucionar o problema. Dentre os modelos existentes, os mais utilizado são: Mamdani e

Takagi-Sugeno. No trabalho de Delgado (2002) é apresentado ênfase no modelo de Takagi-Sugeno, que apresenta uma interpretabilidade linguística é menor quando comparado com o modelo de Mamdani, mas apresenta diversas características que favorecem seu uso. A principal característica do modelo de Takagi-Sugeno é permitir a representação de sistemas complexos utilizando um menor número de regras, e a construção do consequente realizado por meio de algoritmo de estimação numérica.

Em cada uma das regras do sistema nebuloso, o consequente é calculado através de uma relação funcional paramétrica entre as variáveis de entrada. Considerando um sistema nebuloso com m regras, a representação da regra R_j é representado pela Equação 4.2, onde $x = [x_1, x_2, \ldots, x_n]$ representa a base de entrada e $w_j = [w_{j0}, w_{j1} \ldots w_{j(Q-1)})]^T$ contêm os parâmetros da função do consequente.

$$R_j : \text{ Se } X_1 \text{ é } A_1^j \text{ E } X_2 \text{ é } A_2^j \text{ E } \ldots \text{ E } X_n \text{ é } A_n^j \text{ então } Y \text{ é } g_j w_j, x \tag{4.2}$$

A ordem da função de g_j é apresentada conforme o tipo de função utilizada na solução no problema, os tipos mais comuns são: função constante ($Q = 1$), função linear ($Q = n+1$) e função não-linear ($Q = \frac{n(n-1)}{2} + 2n + 1$). No caso presente é apresentado um foco maior para os consequentes não-lineares que possui sua construção conforme apresenta a Equação 4.3.

$$\begin{gathered} g_j(w_j, x) = w_{j0} + w_{j1}x_1 + \cdots + w_{jn}x_n + w_{j(n+1)}x_1x_1 + \cdots + \\ w_{j(2n)}x_1x_n + w_{j(2n+1)}x_2x_2 + \cdots + w_{\frac{n(n-1)}{2}+2n+1}x_nx_n \end{gathered} \tag{4.3}$$

Os valores do vetor w_j são calculados por meio de métodos de otimização de consequente. O primeiro método a ser apresentado é o método global e o segundo e último é o método local.

4.7.1 MÉTODO GLOBAL

O primeiro método de otimização é o método global de estimação numérica dos consequentes, neste método todos as regras são consideradas simultaneamente para a definição dos parâmetros do consequente. O método dos quadrados mínimos é aplicado para cacular os melhores parâmetros de peso ($w^* = [(w_1^*)^T \ldots (w_m^*)^T]^T$), onde $w^* \in \Re^m Q$ e m define a quantidade de regras.

Considerando $\mu_j(x_p)$ o resultado da agregação da regra nebulosa j, $x_p = [x_{1p} \ldots x_{2p}]$ o padrão de entrada p, onde $p = 1, 2, \ldots, N$, com N definido como total de padrões de entrada. A saída $y(x_p)$ é dada pela Equação 4.4.

$$y(x_p) = \frac{\sum_{j=1}^{m} \mu_j(x_p) g_j(w_j, x_p)}{\sum_{j=1}^{m} \mu_j(x_p)} \tag{4.4}$$

O valor de $g_j(w_j, x_p)$ é definido pela Equação do consequente da regra nebulosa, no caso das funções não-lineares $g_j(w_j, x_p)$ é definido pela Equação 4.3. É possível definir β_j pela Equação 4.5 e $\lambda_j(x_p)$ pela Equação 4.6, onde $\lambda_j(x_p) \in \Re^Q$.

$$\beta_j(x_p) = \beta_j^p = \frac{\mu_j(x_p)}{\sum_{j=1}^{m} \mu_j(x_p)} \tag{4.5}$$

$$\lambda_j(x_p) = [\ \beta_j^p \quad \beta_j^p x_{1p} \quad \ldots \quad \beta_j^p x_{np} \quad \beta_j^p x_1 x_1 \quad \beta_j^p x_1 x_n \quad \beta_j^p x_2 x_2 \quad \ldots \quad \beta_j^p x_n x_n\ ]^T \tag{4.6}$$

Com a definição de $\lambda_j(x_p)$ é possível definir $\lambda(x_p) \in \Re mQ$ pela Equação 4.7.

$$\lambda(x_p) = [\ \lambda_1^T \quad \ldots \quad \lambda_m^T\ ]^T \tag{4.7}$$

Com a definição de $\lambda(x_p)$ é possível reescrever a Equação 4.4 conforme apresenta a Equação 4.8.

$$y(x_p) = (\lambda(x_p))^T w \tag{4.8}$$

Seja a matriz $\Lambda \in \Re^{N \times mQ}$ definida pela Equação 4.9.

$$\Lambda = [\ \lambda(x_1) \quad \lambda(x_2) \quad \ldots \quad \lambda(x_N)\ \ \lambda(x_p)]^T \tag{4.9}$$

A saída estimada $\hat{y} = [y(x_1) \ldots y(x_N)]^T \int \Re^N$ é dada pela Equação 4.10, onde y é a matriz de saída estimada para todos dos padrões de entrada.

$$\hat{y} = \Lambda w \tag{4.10}$$

Conforme apresentado por Delgado (2002), considerando y_d como saída desejada pelo sistema, através da otimização dos quadrados mínimos $\underset{w}{min} \frac{1}{2}\|y_d - \hat{y}\|_2^2$, onde $\|.\|$ representa a norma Euclidiana, é possível definir o vetor de parâmetros w^* conforme apresenta a Equação 4.11.

$$w^* = (\Lambda^T \Lambda)^{-1} \Lambda^T y_d \tag{4.11}$$

4.7.2 MÉTODO LOCAL

O segundo, e último, método de otimização é o método local de estimação numérica dos consequentes, neste método as regras são consideradas de forma independente para a otimização dos parâmetros do consequente.

Considerando um padrão de entrada de tamanho N e apenas uma regra j, a saída do sistema é dado conforme a Equação 4.12, onde $\Lambda_j \in \Re N \times Q$, definida pela Equação 4.13

$$\hat{y} = \hat{y}_j = \Lambda_j w_j \tag{4.12}$$

$$\Lambda_j = [\ \lambda_j(x_1) \quad \lambda_j(x_2) \quad \dots \quad \lambda(x_N)\ ]^T \tag{4.13}$$

O consequente $w_j^* = [w_{j0}^* \dots w_{Q-1}^*]$ é obtido pelo método dos quadrados mínimos ponderado expresso pela Equação 4.14, onde a matriz Ψ_j é definido pela Equação 4.15

$$\min_{w_j} \frac{1}{2}\|y_d - \hat{y}\|^2_{\Psi_j} = \min_{w_j} \frac{1}{2}[(y_d - \hat{y})^T \Psi_j (y_d - \hat{y})] \tag{4.14}$$

$$\Psi_j = \begin{bmatrix} \beta_j^1 & 0 & \dots & 0 \\ 0 & \beta_j^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \beta_j^n \end{bmatrix} \tag{4.15}$$

O conjunto de parâmetros ótimo para a regra j é dado pela Equação 4.16

$$w^* = (\Lambda_j^T \Psi_j \Lambda_j)^{-1} \Lambda_j^T \Psi_j y_d \tag{4.16}$$

4.8 PROPOSTA DE PARALELIZAÇÃO

A proposta do presente trabalho é apresentar uma solução para o problema relacionado ao tempo necessário para executar o algoritmo descrito por Delgado (2002). Os fatores que favorecem o aumento no tempo de execução são: treinamento extenso do sistema *fuzzy*, ordem da função utilizada no consequente da regra e quantidade de variáveis de entrada envolvidas no sistema.

As três características destacadas são codificadas no algoritmo em forma de parâmetros utilizados na otimização do consequente, que é a etapa que demanda um maior tempo para execução quando comparado com as outras etapas.

O problema da quantidade de valores aplicados ao treinamento é solucionado ou amenizado reduzindo a quantidade de valores utilizados no treinamento para um valor ideal. No caso da ordem da função do consequente utilizada nas regras, para muitos problemas, é necessário utilizar uma função do tipo não linear para a construção do consequente de cada regra. Utilizar funções constantes ou lineares não são soluções adequadas para sistemas que requerem um mapeamento mais complexo, prejudicando o resultado apresentado. Já o número de variáveis envolvidas dificilmente pode ser reduzido, ocorre apenas quando existe redundância entre elas.

A Figura 17 apresenta um comparativo do número de termos das funções lineares e não-lineares, o crescimento das funções é apresentado conforme o número de variáveis envolvidas é incrementado. Observa-se que a combinação da função não-linear juntamente com um grande número de variáveis envolvidas resulta em funções com vários elementos.

A função do tipo não linear é composta por vários elementos, a Equação 4.3 ilustra a situação. Cada elemento da função possui um valor w_j, que é calculado através de um dos

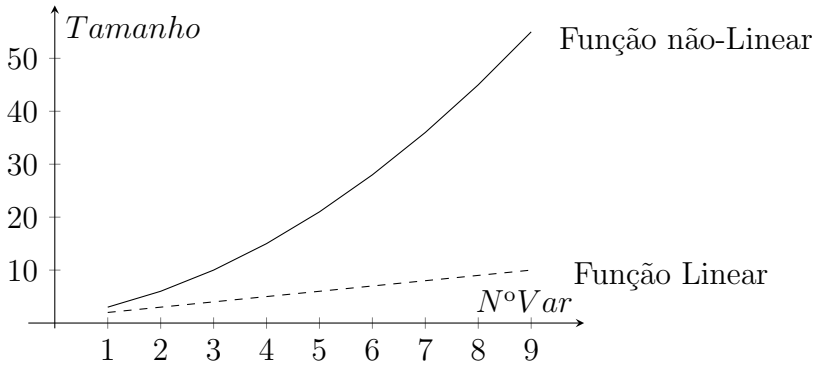

Figura 17 – Crescimento da função conforme o número de variáveis

métodos de estimação numérica (método global ou local), quanto mais w_j possuir uma função maior será a quantidade de cálculo necessário para a solução.

A execução do algoritmo CoevolGFS requer um tempo de execução elevado, principalmente quando executado com grande número de variáveis de entrada utilizando funções não-lineares no consequente das regras. O presente trabalho apresenta uma solução utilizando o paralelismo com a estratégia mestre e escravo.

A implementação do algoritmo CoevolGFS é dada em Java. Mesmo a linguagem escolhida exigindo mais recursos computacionais quando comparado com outras linguagens de programação. Entretanto a linguagem Java apresenta diversos recursos como bibliotecas extras para o desenvolvimento e característica como uma maior abstração e simplificação no desenvolvimento do algoritmo.

Para implementar o paralelismo no algoritmo CoevolGFS foi utilizada a biblioteca do MPJ Express, a qual é uma biblioteca de código aberto desenvolvida em Java e distribuída sob a licença MIT[1] (ROSEN, 2005). A biblioteca permite o desenvolvimento de aplicações com execução em paralelo, possibilitando a execução em multicore e *cluster* (http://mpj-express.org[2]).

A motivação na escolha do MPJ Express é relacionado a linguagem de programação escolhida e a facilidade no uso da biblioteca. A biblioteca permitir realizar a execução do programa em multicore e *cluster*, sem a necessidade de alterar o código desenvolvido, sendo necessário apenas modificar a forma de execução do programa. As linhas de comando a seguir apresentam a diferença entre os dois comandos

```
1 mpjrun.sh -np $1 -jar programa.jar
2
3 mpjrun.sh -np $1 -dev niodev -jar programa.jar
```

[1] licença de *software* criada pela Massachusetts Institute of Technology. Permite que qualquer pessoa que obtém uma cópia do software associado a licença possa utilizar sem restrição, o que inclui modificações e venda, sendo necessário apenas manter o aviso de *copyright* juntamente com a cópia da licença em todas as cópias do software

[2] Acessado em 24 de outubro de 2016

O primeiro comando é utilizado para a execução do programa em multicore e o segundo comando é utilizado par a execução em *cluster*, sendo que "$1" é o número de execuções do programa.

4.8.1 PARALELIZAÇÃO

Para realizar a paralelização do algoritmo é necessário definir em primeiro momento os pontos paralelizáveis do CoevolGFS. Para realizar a análise é necessário apresentar algumas características do algoritmo genético e do sistema *fuzzy*. O algoritmo genético canônico apresenta uma estrutura no qual é possível realizar a paralelização em toda sua execução, o que inclui o processo de inicialização, cálculo do *fitness* e operações genéticas. A facilidade na paralelização do algoritmo genético é favorecido por possuir uma estrutura simples permitindo executar as etapas de forma simultânea, já que um elemento independe do outro para sua construção (CANTU-PAZ, 2000). Como já foi apresentado, o sistema *fuzzy*, em algumas situações, apresenta uma execução que demanda um elevado tempo de processamento decorrente da otimização dos consequentes.

O ponto definido para realizar a paralelização do CoevolGFS é na otimização dos consequentes e cálculo do *fitness* dos indivíduos de nível IV. Ambas as etapas foram escolhidas porque os consequentes calculados são utilizados apenas no cálculo do *fitness* e por apresentarem as seguintes características:

- Maior tempo de processamento: otimizar os consequentes demanda tempo de execução;
- Comunicação mais simples: envio do sistema *fuzzy* e recebimento do *fitness*;
- Sem necessidade de atualizar as populações: o único valor atualizado é o valor de *fitness* dos indivíduos de nível IV.

A Figura 18 apresenta o resultado da paralelização do CoevolGFS. O cálculo do *fitness* dos indivíduos de nível I, II e III não é realizado em paralelo pelo simples fato de dependerem do *fitness* dos indivíduos de nível superior na hierarquia, conforme é apresentado na Seção 4.6.

4.8.2 MESTRE-ESCRAVO

A paralelização do algoritmo é feita utilizando o modelo mestre-escravo (*Master-Slave*), que consiste em um processo mestre e diversos escravos. A Figura 19 apresenta a esquemática de como ocorre a comunicação entre os processos, o processo mestre é responsável em realizar a execução principal do algoritmo e os processos escravos são utilizados para realizar o processamento das etapas paralelizadas (LUQUE; ALBA, 2011).

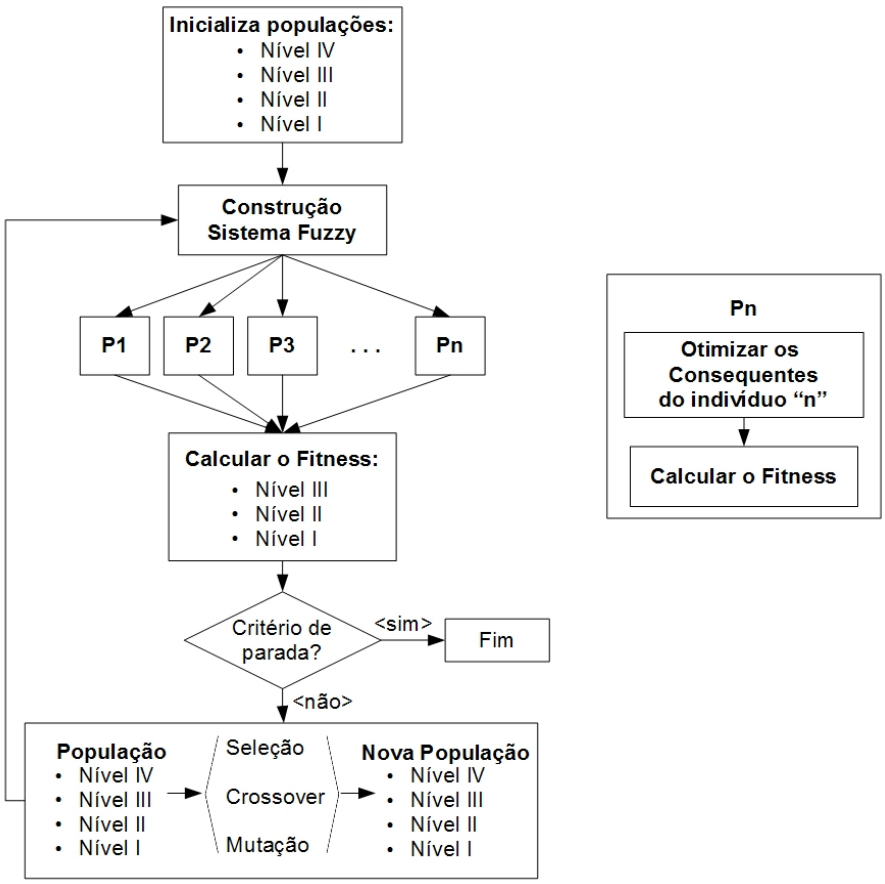

Figura 18 – Diagrama do CoevolGFS Paralelo

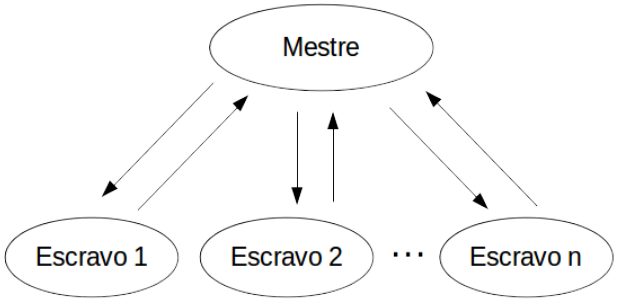

Figura 19 – Diagrama Geral da Estratégia Mestre-Escravo

No algoritmo CoevolGFS, o processo mestre é responsável em realizar a execução do algoritmo como um todo, já os processos escravos são responsáveis em realizar as etapas de otimização dos consequentes e calcular o *fitness* de cada um dos indivíduos de nível IV. A atribuição de tarefa para os processos escravos é responsabilidade do processo mestre. Com base no fluxograma da Figura 18 o processo escravo realiza a execução da etapa "P", nas demais etapas a execução é responsabilidade do processo mestre.

A atribuição das tarefas corre de forma simples, o processo mestre, no momento em que é necessário otimizar os consequentes, passa os indivíduos para os processos escravos de

forma sequencial até não haver mais processos escravos disponíveis. Após finalizar o envio, o processo mestre recolhe os resultados e envia novos indivíduos para serem caculados. Caso o processo escravo não tenha terminado o cálculo, o mestre espera sua finalização para dar continuidade. Tal procedimento é realizado até não existir mais indivíduos para serem calculados.

O resultado final do funcionamento do algoritmo na estratégia *master-slave* é apresentado na Figura 20. Na figura é apresentado apenas um processo mestre e um processo escravo por motivos de simplicidade, mas é aplicável com a existência de mais de um processo escravo.

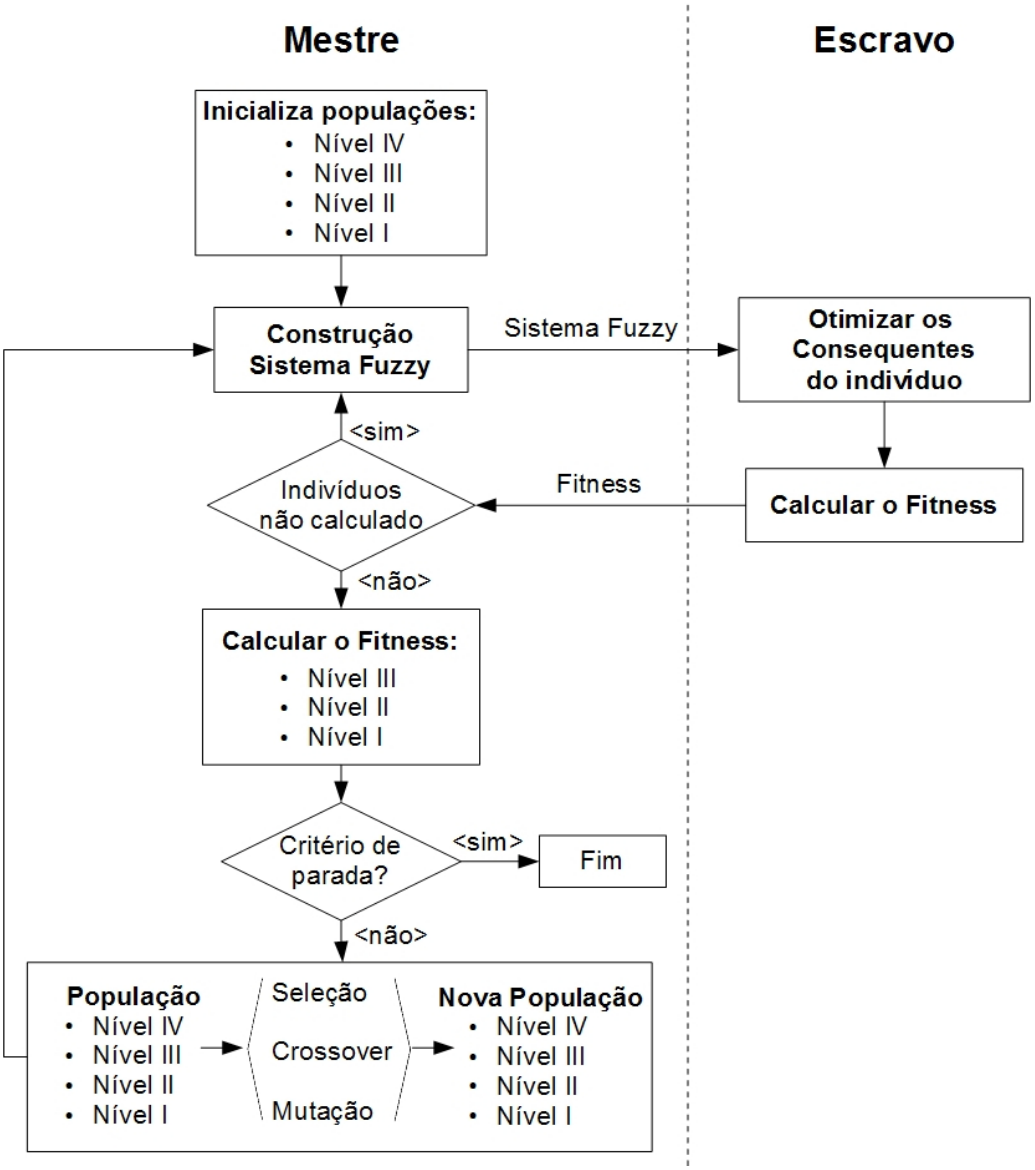

Figura 20 – Diagrama CoevolGFS Paralelo - Master-Slave

Após definido como será realizado a paralelização do CoevolGFS é necessário definir quais as estruturas físicas utilizadas para realizar a tarefa. A realização da paralelização do algoritmo será apresentada em dois cenários diferentes. O primeiro é a execução utilizando

diversos núcleos de um mesmo processador e o segundo cenário é a execução utilizando diversas máquinas conectadas via rede, formando um *cluster*.

A execução utilizando diversos núcleos de um mesmo processador é motivada pela existência de processadores com diversos núcleos, permitindo realizar com facilidade a execução em paralelo do algoritmo. A execução utilizando diversas máquinas conectadas via rede (*cluster*) é motivada para observar: a diferença na execução utilizando mais recursos computacionais e analisar o impacto da conexão via rede.

5 SIMULAÇÃO E RESULTADOS

Este capítulo tem como objetivo apresentar os resultados da paralelização do algoritmo em diferentes cenários. Além dos resultados são apresentados mais detalhes sobre a implementação do algoritmo e dos testes realizados.

O algoritmo executado tem como objetivo obter um modelo *fuzzy* para as funções não lineares apresentadas pelas três equações $F_1(5.1)$, $F_2(5.2)$ e $F_3(5.3)$. A forma de obtenção do modelo *fuzzy* ocorre a partir de uma base de treinamento de N dados de entrada no formato $(\overrightarrow{x}_i, f(\overrightarrow{x}_i))$, lembrando que $\overrightarrow{x}_i$ é o vetor de variáveis de entrada.

$$F_1 = f(x_1) = sin(6\pi x_1) + 2x_1 + \exp(-\frac{1}{0,03^2}(x_1 - 0,59^2)) \tag{5.1}$$

$$F_2 = f(x_1, x_2) = \frac{sen(x_1)}{x_1}\frac{sen(x_2)}{x_2} \tag{5.2}$$

$$F_3 = f(x_1, x_2, x_3) = \frac{sen(x_1)}{x_1}\frac{sen(x_2)}{x_2}\frac{sen(x_3)}{x_3} \tag{5.3}$$

O algoritmo apresenta as seguintes etapas de execução:

- Etapa I: Inicialização das populações dos quatro níveis;
- Etapa II: Otimização do consequentes;
- Etapa III: Cálculo do *fitness*;
- Etapa IV: Critério de parada;
- Etapa V: Operações genéticas.

O algoritmo é paralelizado na etapa II e apenas cálculo do *fitness* dos indivíduos de nível IV na etapa III. O que motivou a paralelização de tais etapas está relacionado ao tempo necessário para executar a tarefa. A Tabela 2 apresenta uma comparação do tempo consumido na execução do algoritmo como um todo e o tempo consumido apenas na etapa II. O teste para o levantamento do tempo consumido pelo algoritmo não é realizado em paralelo. A inclusão do cálculo do *fitness* da etapa III é adicionado ao paralelismo por motivos de simplicidade da comunicação.

Com base nos resultados apresentados na Tabela 2, é possível verificar que os tempos consumidos pela etapa II nas funções F_1, F_2 e F_3 representam respectivamente 84,5722%, 99,2706% e 99,903% do tempo real, respectivamente.

A estratégia utilizada no paralelismo é a do mestre-escravo, onde um processo é nomeado como mestre e os demais são os escravos. A execução utilizando essa estratégia requer pelo

Tabela 2 – Tempo total x Tempo Etapas II

Função	Tempo total[ms]	Tempo etapa II[ms]
F_1	42702	36114
F_2	156008	154870
F_3	806828	806045

menos dois processos (um mestre e um escravo). Os processos escravos realizam apenas a otimização dos consequentes e o cálculo do *fitness* de cada indivíduo, as demais tarefas executadas são responsabilidade do processo mestre.

A Figura 19 apresenta o diagrama de comunicação entre os processos para os casos estudados. O processo mestre executa as tarefas do algoritmo até o momento em que é necessário otimizar os consequentes e calcular o *fitness* dos indivíduos.

A comunicação do mestre com os escravos ocorre de forma simples. O processo mestre envia tarefas para serem executadas nos processos escravos. Finalizando o envio para todos os processos escravos disponíveis, o processo mestre retorna para o primeiro processo escravo buscando pela reposta, caso a resposta ainda não seja dada, o processo mestre espera sua finalização, caso contrário envia outra tarefa. Tal procedimento é executado para todos os processos escravos até não haver mais tarefas para serem executadas em paralelo.

O processo escravo, por sua vez, realiza o cálculo de otimização dos consequentes e o cálculo do *fitness* de cada indivíduo, retornando as respostas obtidas para o processo mestre. No momento em que a resposta é apresentada para o mestre, o processo escravo passa a estar disponível para calcular os valores de um novo indivíduo.

O algoritmo genético é configurado da seguinte forma:

- Seleção primária (80%): Torneio;

  Tamanho: 10%;

  Taxa de escolha: 50%;

- Seleção secundária (20%): Torneio por distância Euclidiana;

  Tamanho: 10%;

  Taxa de escolha: 50%;

- *Crossover*: Single Point;

- Mutação: Aleatória 5%;

- Tamanho da População

  Lv1: 50;

  Lv2: 300;

Lv3: 20;

Lv4: 30;

A seguir é apresentado os resultados da execução do CoevolGFS para as três funções apresentadas. Além da execução utilizando as três funções, o sistema é executado em dois cenários diferentes. O primeiro cenário é a paralelização utilizando um único processador com diversos núcleos. Neste cenário o computador utilizado possui as características descritas na Tabela 3. O segundo cenário e a paralelização utilizando diversos computadores conectados via rede (*cluster*). As principais características do *cluster* são apresentadas na Tabela 4, o *cluster* é customizado do Grupo de Pesquisa em Engenharia de Sistemas e de Computação (GPESC), da Universidade Federal de Itajubá (UNIFEI) (NUNES, 2016).

Tabela 3 – Características do Computador

Processador	i7-3770 3,40 GHz x 8
S.O.	Ubuntu 14.04 LTS
RAM	8GB

Tabela 4 – Características do *Cluster*

Processador front-end	Intel Core 2 Quad CPU Q6700 2.66GHz
Processador back-end	Intel Core 2 Quad CPU Q8200 2.33GHz
S.O.	Ubuntu 14.04 LTS
RAM front-end	2GB
RAM back-end	6GB
Número de máquinas	12
Conexão	Rede Ethernet 10/100

Os testes consistem em realizar 120 execuções para cada configuração apresentada. Utilizando o computador descrito na Tabela 3 são realizados testes utilizando de 1 a 8 núcleos, enquanto a execução utilizando o *cluster* descrito na Tabela 4 é realizado testes utilizando de 1 a 12 máquinas. A execução do algortimo genético utilizando apenas um núcleo do processador, ou uma única máquina do *cluster* é a execução sequencial do algoritmo.

Para realizar a avaliação da paralelização do algoritmo CoevolGFS duas medidas são utilizadas: *speedUp* e eficiência.

O valor do *speedup* indica a proporção de ganho do sistema paralelo quando comparado com o sistema não paralelo. A Equação 5.4 indica como é realizado o cálculo do *speedup*, onde T_n é o tempo de execução do algoritmo utilizando n processadores. A eficiência é dita como a média de utilização dos processadores. A Equação 5.5 indica como é

realizado o cálculo, onde n é o número de processadores utilizados (EAGER; ZAHORJAN; LAZOWSKA, 1989).

$$Sp(n) = \frac{T_1}{T_n} \tag{5.4}$$

$$Ef(n) = \frac{Sp(n)}{n} \tag{5.5}$$

5.1 PARALELIZAÇÃO DE F_1

Esta seção tem como objetivo apresentar os resultados dos testes do CoevolGFS utilizando a função F_1, conforme apresentado pela Equação 5.1. O propósito do teste é verificar o desempenho do algoritmo CoevolGFS quando aplicado a uma função que possui apenas uma variável de entrada.

As características do sistema executado são:

- Número de gerações: 1000;
- Equação do consequente: não linear;
- Número de treinos: 100;
- Intervalo:

  x_1: [0, 1];
- Número de regras: 10.

5.1.1 F_1: PARALELIZAÇÃO EM UM PROCESSADOR

No primeiro cenário, foi realizado o experimento com o CoevolGFS utilizando a função F_1 em um processador com múltiplos núcleos. O computador utilizado possui as características apresentadas na Tabela 3.

O primeiro resultado analisado é o tempo necessário para executar o algoritmo conforme a quantidade de núcleos utilizados do processador é incrementado. Os resultados são apresentados na Tabela 5.

A princípio é possível observar que o tempo médio necessário para executar cada um dos testes não apresenta uma redução de tempo conforme o número de núcleos é incrementado. Observa-se que a execução sequencial do algoritmo (utilizando apenas um núcleo) é o melhor resultado obtido. Com o uso de dois núcleos o tempo apresenta um acréscimo, que conforme o número de núcleos é incrementado, decai até a adição do quarto núcleo. O problema de otimização tratado é relativamente simples, no qual a utilização de vários

Tabela 5 – Resultados da Função F_1

Núcleos	Tempo [ms]		EQM		Sp	Ef
	Médio	D. Padrão	Média	D. Padrão		
1	21023	887	$2,3016.10^{-2}$	$1,3419.10^{-2}$		
2	34901	1506	$1,2655.10^{-2}$	$1,0102.10^{-2}$	0,6024	30,12
3	24519	642	$1,6943.10^{-2}$	$0,6073.10^{-2}$	0,8574	28,58
4	23954	454	$1,9129.10^{-2}$	$1,3278.10^{-2}$	0,8777	21,94
5	24895	473	$1,7960.10^{-2}$	$1,1512.10^{-2}$	0,8445	16,89
6	27221	877	$1,9019.10^{-2}$	$1,2308.10^{-2}$	0,7723	12,87
7	29297	1102	$1,9465.10^{-2}$	$1,2560.10^{-2}$	0,7176	10,25
8	30576	1362	$2,1617.10^{-2}$	$2,1617.10^{-2}$	0,6876	8,59

núcleos não gera ganho pois a implementação de um processo paralelo demanda a troca de mensagens entre o mestre e o escravo, o que aumenta o tempo de processamento final.

A Tabela 5 apresenta, juntamente com o tempo médio, o valor de erro quadrático médio (EQM) obtido em cada uma das execuções, possibilitando avaliar a qualidade das respostas conforme o número de núcleos é incrementado. Observa-se que os valores de EQM não possuem variações significativas, permitindo concluir que a paralelização do algoritmo não interfere nos resultados.

Além dos valores de tempo médio de execução e EQM, é apresentado os valores de *speedUp* e Eficiência, obtidos através das Equações 5.4 e 5.5 respectivamente. Os valores de *speedUp* não são superiores a 1, verificando-se também que o tempo necessário para a execução do algoritmo é apenas prejudicado. Da mesma forma, o valor de Eficiência apresenta um decaimento a cada novo núcleo adicionado para execução.

5.1.2 F_1: PARALELIZAÇÃO EM UM *CLUSTER*

O segundo cenário onde é realizado os experimentos do CoevolGFS utilizando a função F_1 é em um *cluster* com as características apresentadas conforme a Tabela 4.

O primeiro resultado apresentado na Tabela 6 é o tempo médio de execução do experimento para cada quantidade de máquinas utilizadas no *cluster*. Em primeiro momento é possível notar que a diferença do tempo de execução do experimento em questão não apresenta muita diferença quando é adicionado uma máquina a mais para processamento.

Além do tempo médio de execução, a Tabela 6 apresenta o EQM médio de cada uma das execuções. Com base nos valores obtidos é possível observar que a paralelização da função F_1 não interfere na precisão dos resultados do algoritmo.

Assim como nos testes realizados anteriormente, para avaliar o paralelismo é necessário verificar os valores de *speedUp* e eficiência. Com base nos resultados da Tabela 6, é possível notar que o algoritmo apresenta uma redução no tempo de execução quando se utiliza de 3 até 8 máquinas. A diferença apresentada no tempo de execução, mesmo que melhor, não

Tabela 6 – Resultados da função F_1 em *cluster*

Máquinas	Tempo [ms]		EQM		Sp	Ef
	Médio	D. Padrão	Média	D. Padrão		
1	43364	1079	$2,1324.10^{-2}$	$1,3256.10^{-2}$		
2	55845	1495	$1,9739.10^{-2}$	$1,3816.10^{-2}$	0,7764	38,8248
3	39102	769	$1,4252.10^{-2}$	$1,3164.10^{-2}$	1,1089	36,9665
4	36144	536	$1,5644.10^{-2}$	$1,4482.10^{-2}$	1,1997	29,9934
5	47975	406	$1,7087.10^{-2}$	$1,3518.10^{-2}$	1,1811	23,6222
6	38292	308	$1,8689.10^{-2}$	$1,5110.10^{-2}$	1,1324	18,8738
7	40249	262	$1,9665.10^{-2}$	$1,3876.10^{-2}$	1,0773	15,3911
8	42675	251	$1,6983.10^{-2}$	$1,1942.10^{-2}$	1,0161	12,7016
9	45318	294	$2,0122.10^{-2}$	$1,2233.10^{-2}$	0,9568	10,6319
10	48119	717	$1,9851.10^{-2}$	$1,3186.10^{-2}$	0,9011	9,0117
11	51293	822	$1,8433.10^{-2}$	$1,5156.10^{-2}$	0,8454	7,6855
12	54126	308	$1,9602.10^{-2}$	$1,3451.10^{-2}$	0,8011	6,6763

é significativa. Os valores de *speedUp* apresentam a porcentagem de uso das máquinas, que conforme o número de máquinas é incrementado a eficiência apresenta um decaimento significativo.

5.1.3 F_1: CONSIDERAÇÕES

Os testes do CoevolGFS em paralelo para realizar a aproximação da função F_1 não apresentou nenhuma melhoria em nenhum dos dois cenários. A justificativa para que o processamento não tenha melhorado está relacionada ao número de variáveis da função F_1.

A execução utilizando a função F_1 ocorre de forma rápida quando comparado com funções com maior número de variáveis, exemplo F_2 e F_3. Conforme apresentado na Seção 4.7, o custo computacional para o cálculo do consequente depende do tamanho do polinômio, já que é necessário calcular mais parâmetros w_j. Com um polinômio de tamanho reduzido a etapa de otimização é realizada mais rapidamente. Na estratégia mestre-escravo com a forma de comunicação adotada, os escravos passam mais tempo em espera do mestre do que efetivamente calculando os resultados, prejudicando a eficiência do algoritmo. A Tabela 2 apresenta a diferença proporcional do tempo de execução das funções F_1, F_2 e F_3, onde as funções apresentam 1, 2 e 3 variáveis, respectivamente.

Os resultados da execução da função F_1 são apresentados na Figura 21, a ilustração apresenta a função original (dados de referência) e a estimada (dados estimados). Observa-se que o resultado obtido é similar a função original.

5.2 PARALELIZAÇÃO DE F_2

O objetivo desta seção é avaliar o desempenho do algoritmo CoevolGFS utilizando a função F_2, apresentado pela Equação 5.2. O propósito do teste é verificar o desempenho

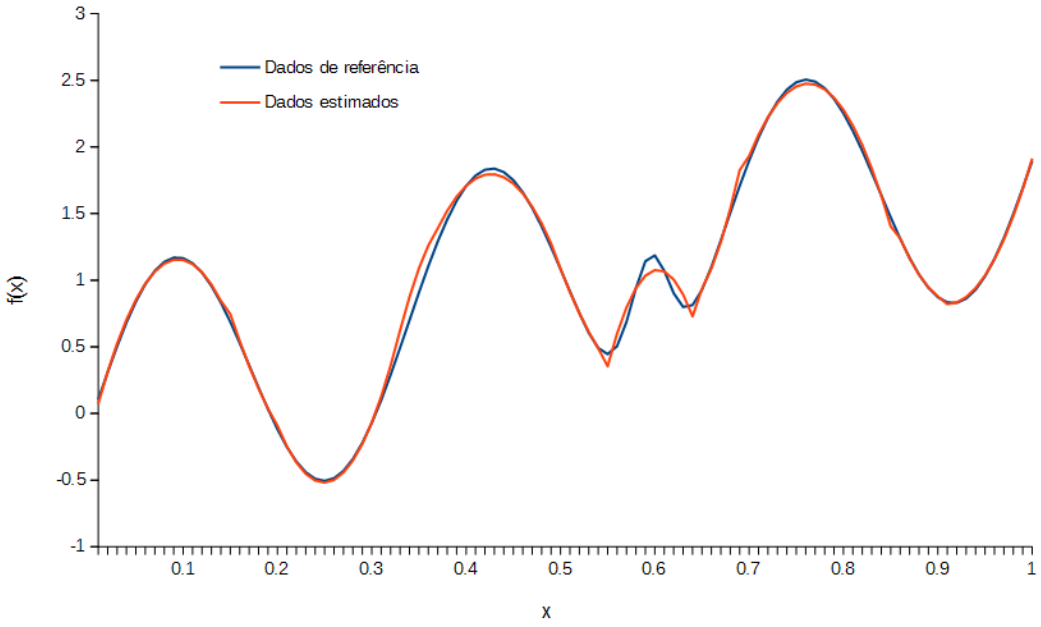

Figura 21 – Resultado da função F_1

do algoritmo em paralelo nos dois cenários apresentados, utilizando uma função com duas variáveis.

As características do sistema executado são:

- Número de gerações: 100
- Equação do consequente: não linear
- Número de treinos: 400
- Intervalo

  x_1: [-10, 10]

  x_2: [-10, 10]

- Número de regras: 25

5.2.1 F_2: PARALELIZAÇÃO EM UM PROCESSADOR

O primeiro teste realizado tem como propósito é avaliar o impacto da paralelização do CoevolGFS utilizando diversos núcleos de um mesmo computador, as características do computador são descritos na Tabela 3. O objetivo da metodologia é apresentar um sistema *fuzzy* que realize o mapeamento da função F_2 apresentada pela Equação 5.2. Os resultados obtidos são apresentados na Tabela 7.

Além do tempo médio de execução, a Tabela 7 apresenta o erro quadrático médio (EQM) de acordo com o número de núcleos de processamento utilizados. O valor do EQM

Tabela 7 – Resultados da função F_2

Núcleos	Tempo [ms]		EQM		Sp	Ef
	Médio	D. Padrão	Média	D. Padrão		
1	74797	7600	$1,7483.10^{-3}$	$3,1881.10^{-4}$		
2	79425	8240	$1,8793.10^{-3}$	$4,2183.10^{-4}$	0,9417	47,0865
3	43872	3218	$1,8292.10^{-3}$	$3,0977.10^{-4}$	1,7048	56,8297
4	32807	2192	$1,7124.10^{-3}$	$3,0698.10^{-4}$	2,2799	56,9977
5	28743	1952	$1,7532.10^{-3}$	$2,6425.10^{-4}$	2,6022	52,0453
6	26489	1757	$1,7604.10^{-3}$	$3,2206.10^{-4}$	2,8237	47,0616
7	24480	1247	$1,7881.10^{-3}$	$3,1920.10^{-4}$	3,0554	43,6490
8	23932	1145	$1,7741.10^{-3}$	$3,5294.10^{-4}$	3,1253	39,0674

de cada uma das execuções não apresenta variações significativas, permitindo concluir que a execução do algoritmo em paralelo não interfere na precisão dos resultados.

Na Tabela 7 observa-se que a execução utilizando um núcleo (74797ms) apresenta um tempo inferior a execução utilizando dois núcleos (79425ms). Com um núcleo não é realizada a comunicação, já que todo o algoritmo é executado de forma sequencial. Na execução com mais de um núcleo do processador a comunicação passa a ser necessária, já que um núcleo executa o processo mestre e os demais núcleos o processo escravo. No caso do sistema executado em dois núcleos, tem-se um processo mestre e um processo escravo. O cálculo do consequente e do *fitness* ocorre em apenas um processo, o escravo, não apresentando ganho em desempenho adicionando apenas o tempo de comunicação.

A execução com 3 núcleos passa a apresentar uma melhora significativa no tempo de execução. Isso decorre devido ao fato de existir dois processos escravos, permitindo executar o cálculo do consequente e do *fitness* de forma mais rápida. É importante destacar que o cálculo do consequente em uma função de duas variáveis é uma tarefa que demanda um tempo de execução significativo.

A eficiência na Tabela 7 apresenta um decaimento no momento em que a execução passa a utilizar 5 núcleos. O decaimento na eficiência é explicado pelo aumento na comunicação entre o processo mestre e os processos escravos e a forma como ocorre a comunicação.

Pode-se observar os dados apresentados na Tabela 7 que o algoritmo apresentou uma melhora significativa conforme o número de núcleos é incrementado.

5.2.2 F_2: PARALELIZAÇÃO EM UM CLUSTER

O segundo cenário em que testou-se o CoevolGFS Paralelo com a função F_2 consistiu na utilização de diversos computadores conectados via rede, um *cluster*. O propósito é avaliar o impacto da paralelização utilizando mais recurso computacional quando comparado ao primeiro cenário apresentado. Os resultados obtidos são apresentados na Tabela 8.

Além do tempo médio de execução é apresentado o EQM, permitindo verificar se ocorreu alguma alteração significativa nos resultados apresentados pelo sistema. Com base

Tabela 8 – Resultados da função F_2 em *cluster*

Máquinas	Tempo [ms]		EQM		Sp	Ef
	Médio	D. Padrão	Média	D. Padrão		
1	164561	11563	$1,7296.10^{-3}$	$3,1115.10^{-4}$		
2	167591	10345	$1,7212.10^{-3}$	$3,4438.10^{-4}$	0,9819	49,0960
3	84485	5992	$1,8147.10^{-3}$	$3,3937.10^{-4}$	1,9478	64,9271
4	59318	4089	$1,8149.10^{-3}$	$3,4116.10^{-4}$	2,7742	69,3554
5	47975	3453	$1,7764.10^{-3}$	$3,3507.10^{-4}$	3,4301	68,6028
6	39074	2414	$1,7224.10^{-3}$	$3,2715.10^{-4}$	4,2115	70,1920
7	34124	2121	$1,7422.10^{-3}$	$3,1930.10^{-4}$	4,8224	68,8920
8	32263	2088	$1,8109.10^{-3}$	$3,2236.10^{-4}$	5,1006	63,7576
9	28913	1547	$1,7265.10^{-3}$	$3,3327.10^{-4}$	5,6916	63,2399
10	26727	1605	$1,6860.10^{-3}$	$2,8293.10^{-4}$	6,1569	61,5688
11	24344	1185	$1,7272.10^{-3}$	$3,3014.10^{-4}$	6,7598	61,4528
12	23573	1209	$1,8121.10^{-3}$	$3,0728.10^{-4}$	6,9809	58,1742

nos resultados apresentados na Tabela 7 observa-se que a adição de uma ou mais máquinas não interfere na precisão do algoritmo.

Outros resultados apresentados são os valores de *speedUp* e eficiência. O resultado de *speedUp* apresentou um crescimento conforme a quantidade de máquinas utilizadas no *cluster* é incrementada. A execução com 12 máquinas apresentou um aumento na velocidade de 6,9809 vezes quando comparado com a execução sequencial do algoritmo.

A eficiência apresentou um crescimento até o uso de 6 processadores. A partir da adição da sétima máquina no *cluster* apresentou-se um decaimento na eficiência conforme o número de máquinas é incrementado. O que justifica o decaimento na eficiência está relacionado ao tempo de espera na comunicação das máquinas do *cluster*.

5.2.3 F_2: CONSIDERAÇÕES

A execução do CoevolGFS em paralelo apresentou bons resultados quando aplicado na obtenção de um modelo para a função F_2. A execução do algoritmo utilizando oito núcleos de um processador apresentou uma velocidade de execução 3,1253 vezes maior quando comparado com a execução utilizando apenas um núcleo, conforme apresentado na Tabela 7. Com relação à execução em um *cluster*, utilizar 12 máquinas apresentou uma redução do tempo de 6,9809 vezes quando comparado com a execução utilizando apenas uma máquina, conforme apresentado na Tabela 8.

O resultado do EQM em ambos os cenários, execução em multicore e *cluster*, não apresentou alterações nos resultados. Pode-se concluir que a paralelização da função F_2 não interfere na precisão dos resultados obtidos.

A Figura 22 apresenta o comportamento da função F_2, utilizada na atual execução do algoritmo. Os valores estimados pela algoritmo são apresentados na figura 23. Observa-se

que a estimativa apresentada como resultado final do algoritmo se aproxima de forma satisfatória quando comparado com a função original.

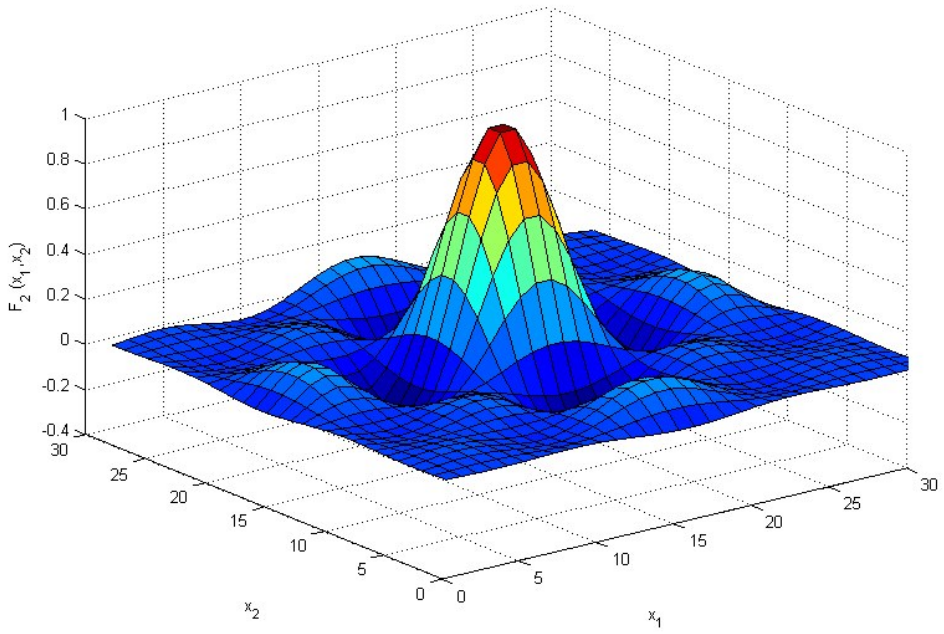

Figura 22 – Função original F_2

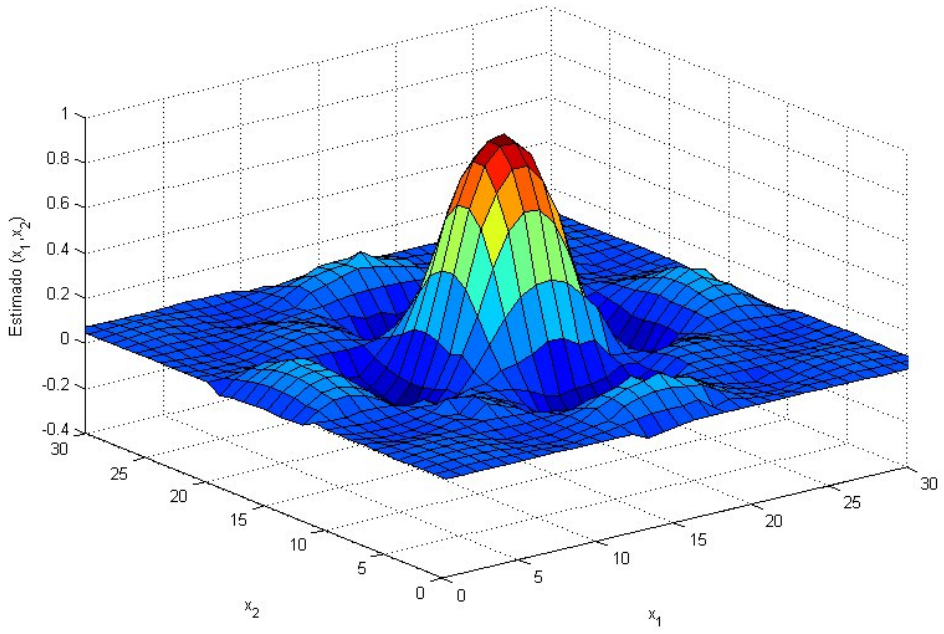

Figura 23 – Resultado da função F_2

5.3 PARALELIZAÇÃO DE F_3

O propósito desta seção é apresentar o desempenho do algoritmo CoevolGFS utilizando a função F_3. A motivação na escolha da função é dada pelo número de dimensões presentes. A função F_3 é a extensão da função F_2 em um universo de três dimensões de variáveis de entada.

As características do sistema executado são:

- Número de gerações: 100;
- Equação do consequente: não linear;
- Número de treinos: 1000;
- Intervalo:

  x_1: [-10, 10];

  x_2: [-10, 10];

  x_3: [-10, 10];
- Número de regras: 25.

5.3.1 F_3: PARALELIZAÇÃO EM UM PROCESSADOR

A função F_3 foi testada no primeiro cenário, sendo executada em um processador com diversos núcleos. As características do computador utilizado são apresentados na Tabela 3.

O primeiro resultado obtido é o tempo necessário para executar o algoritmo conforme a quantidade de núcleos é incrementada. Os resultados são apresentados na Tabela 9. Observando apenas o tempo necessário para cada execução na tabela, nota-se que o tempo médio necessário para realizar cada uma das execuções apresenta um decaimento significativo.

Juntamente com o tempo médio necessário para a execução é apresentado o valor de EQM médio de cada uma das execuções, possibilitando verificar se a adição de um novo núcleo interfere na qualidade das respostas apresentadas. Na Tabela 9 observa-se que não existe uma variação significativa quando adicionado um novo núcleo na execução, permitindo concluir que a paralelização utilizada não interfere nos resultados sob o ponto de vista da precisão dos modelos.

Os terceiros e quarto valores apresentado na tabela são os valores de *speedUp* e Eficiência, respectivamente. Em relação ao *speedUp* é possível observar um crescimento contínuo conforme é adicionado um novo núcleo para processamento. Já a eficiência apresenta um crescimento até a adição do terceiro núcleo.

Tabela 9 – Resultados da função F_3

Núcleos	Tempo [ms]		EQM		Sp	Ef
	Médio	D. Padrão	Média	D. Padrão		
1	279314	23703	$7,0496.10^{-4}$	$2,3742.10^{-4}$		
2	275425	24432	$7,6760.10^{-4}$	$2,3534.10^{-4}$	1,0141	50,71
3	166112	10578	$7,5700.10^{-4}$	$1,9000.10^{-4}$	1,6815	56,05
4	130116	7823	$7,6814.10^{-4}$	$2,5147.10^{-4}$	2,1467	53,67
5	110271	5385	$7,9949.10^{-4}$	$2,5964.10^{-4}$	2,5330	50,66
6	103020	4887	$7,3402.10^{-4}$	$2,5161.10^{-4}$	2,7113	45,19
7	97296	4621	$7,5784.10^{-4}$	$2,5906.10^{-4}$	2,8708	41,01
8	96350	3968	$8,4434.10^{-4}$	$2,8352.10^{-4}$	2,8990	36,24

5.3.2 F_3: PARALELIZAÇÃO EM UM CLUSTER

O segundo cenário onde se testou a metodologia proposta para a obtenção de um modelo *fuzzy* para a função F3 foi em um *cluster* com as especificações apresentadas na Tabela 4. Os resultados obtidos são apresentados na Tabela 10. O primeiro resultado apresentado é o tempo médio necessário para executar o algoritmo conforme a quantidade de máquinas é incrementada. A princípio, observa-se que o tempo necessário para a execução apresenta um decaimento significativo.

Juntamente com o tempo médio é apresentado o EQM médio de cada execução. Os valores de EQM não apresentam uma variação significativa conforme o número de núcleos é incrementado. É possível admitir que a paralelização do algoritmo utilizando a função F_3 não interfere na qualidade dos resultados apresentados.

O terceiro e quarto valores apresentados na Tabela 10 são os valores de *speedUp* e Eficiência, respectivamente. Em relação ao *speedUp* é possível observar um crescimento significativo conforme o número de núcleos é incrementado. Já a eficiência apresenta um crescimento até a adição do sexto computador. No entanto, observa-se que a partir da adição do terceiro computador a eficiência apresenta valores relativamente altos quando comparado com os demais resultados do presente trabalho.

5.3.3 F_3: CONSIDERAÇÕES

A execução do CoevolGFS em paralelo na aproximação da função F_3 apresentou bons resultados quando o objetivo é reduzir o tempo médio de execução. A execução do algoritmo utilizando oito núcleos de um mesmo processador apresentou uma velocidade de execução 2,899 vezes maior quando comparado com a execução utilizando apenas um núcleo, conforme apresentado na Tabela 9. A execução utilizando no *cluster* com 12 máquinas apresentou uma redução no tempo de 9,19 vezes quando comparado com a execução utilizando apenas uma máquina, conforme apresentado na Tabela 10.

Assim como nos demais testes o resultado do EQM em ambos cenários (multicore e

Tabela 10 – Resultados da função F_3 em cluster

Máquinas	Tempo [ms]		EQM		Sp	Ef
	Médio	D. Padrão	Média	D. Padrão		
1	783060	34678	$6,8127.10^{-4}$	$2,3453.10^{-5}$		
2	711746	38393	$8,1855.10^{-4}$	$2,7417.10^{-5}$	1,1001	55,0097
3	363130	17936	$8,3183.10^{-4}$	$2,9436.10^{-5}$	2,1564	71,8805
4	247791	12818	$7,7475.10^{-4}$	$2,6349.10^{-5}$	3,1601	79,0039
5	198971	11109	$7,8694.10^{-4}$	$2,4446.10^{-5}$	3,9355	78,7109
6	156417	7783	$7,6919.10^{-4}$	$2,4135.10^{-5}$	5,0062	83,4370
7	132961	7678	$8,0122.10^{-4}$	$2,6081.10^{-5}$	5,8893	84,1341
8	125632	8007	$8,1535.10^{-4}$	$2,8072.10^{-5}$	6,2329	77,9118
9	108796	5953	$8,3119.10^{-4}$	$2,3756.10^{-5}$	7,1975	79,9723
10	105318	6449	$8,1281.10^{-4}$	$2,5181.10^{-5}$	7,4351	74,3515
11	86302	4117	$8,3163.10^{-4}$	$2,3522.10^{-5}$	9,0734	82,4857
12	85207	4676	$8,1259.10^{-4}$	$2,9112.10^{-5}$	9,1900	76,5835

cluster) não apresentaram alterações significativas nos resultados, podendo concluir que a paralelização da função função F_3 apresentou uma melhoria no tempo de execução e não interferiu na qualidade dos resultados apresentados pelo algoritmo.

6 CONCLUSÕES

Neste trabalho apresentou-se uma abordagem para solucionar o problema relacionado ao tempo necessário de processamento do algoritmo CoevolGFS apresentado originalmente por Delgado (2002). O CoevolGFS é uma técnica híbrida conhecida como *genetic fuzzy system* (GFS). Esta técnica é responsável pela obtenção de sistemas nebulosos evoluídos através de técnicas de aprendizagem de algoritmos genéticos. Na estrutura mais comum do GFS cada sistema *fuzzy* é codificado em um único indivíduo da população do algoritmo genético, o que não é viável para situações mais complexas. A principal particulariedade do CoevolGFS é que ele utiliza um sistema de coevolução entre indivíduos de populações diferentes, onde cada tipo de indivíduo representa um solução parcial do problema. Desta forma, atribuem-se noções de hierarquia e cooperação entre os indivíduos.

O CoevolGFS desenvolvido no presente trabalho é caracterizado por utilizar o modelo de Takagi-Sugeno com otimização dos consequentes pelo método local realizando o mapeamento com funções não-lineares. No trabalho realizado por Delgado (2002), os resultados obtidos foram superiores aos métodos já existentes. Esta superioridade se deu com relação à interpretabilidade nos resultados e a maior precisão destes. Outra característica importante do algoritmo é a forma como é realizada a codificação do sistema *fuzzy*, conforme apresentado no Capítulo 4. Além das vantagens do algoritmo é também apresentado o problema relacionado ao custo computacional para a solução de problemas com elevado número de variáveis e/ou dados.

A motivação para o desenvolvimento deste trabalho era o de solucionar, ou minimizar, o problema do custo computacional. O custo computacional está diretamente relacionado ao número de variáveis do sistema, principalmente quando utilizam-se funções do tipo não-linear para realizar a construção do consequente, conforme apresentado no Capítulo 4. Realizar a execução do algoritmo em um menor tempo ou aplicar em um sistema com maior número de variáveis, amplia as áreas de aplicação da solução proposta por Delgado (2002).

Como solução para este problema propôs-se a utilização de técnicas de paralelização computacional na solução do CoevolGFS. A paralelização do algoritmo foi realizada com foco na etapa de otimização dos consequentes. Conforme estudos e análise realizada, pode-se concluir que esta é a etapa do método que demanda maior tempo de execução.

Para o teste e validação do método proposto realizou-se a análise do método aplicando-o na obtenção de diferentes modelos *fuzzy* TS em situações distintas. Sendo assim, analisou-se problemas relacionados ao número de dimensões do sistema sob estudo e à estrutura utilizada para processamento (múltiplos núcleos e *cluster*).

Com base nos resultados apresentados é possível considerar que os testes utilizando

a função caracterizada por apresentar uma única variável de entrada não apresentaram ganho expressivo. Este fato é justificado pelo tempo necessário para realizar a etapa II em relação ao tempo de espera do mestre para distribuir as tarefas para cada um dos escravo. A execução utilizando a função F_2, caracterizada por apresentar duas variáveis de entrada, apresentou uma diferença máxima na velocidade no primeiro cenário de 3,12 vezes e no segundo cenário 6,98 vezes. Já a função F_3, caracterizada por apresentar três variáveis de entrada, resultou em uma diferença máxima na velocidade no primeiro cenário de 2,89 vezes e no segundo cenário de 9,19 vezes. Em ambas as funções F_2 e e F_3, cujo respectivo custo computacional se apresentou elevado, houve ganhos significativos quando o CoevolFGS foi resolvido de acordo com a proposta apresentada neste trabalho. Além disso, em todos os testes realizados a qualidade da resposta, erro entre saída exata e a estimada, não apresentou variações significativas originadas da paralelização. Sendo assim, é possível concluir que realizar a paralelização, conforme foi apresentado, não interfere na qualidade de estimação dos modelos finais.

Desta forma, pode-se concluir que a proposta de paralelização do CoevolGFS é uma alternativa viável quando aplicada aos sistemas que demandem mais maior custo computacional de processamento, ou em sistemas mais complexos que demandam a utilização de consequentes formados por funções não-lineares. Mesmo demandando maior trabalho durante o desenvolvimento quando comparado com o GFS simples, a metodologia proposta apresenta resposta com maior eficiência quando comparada com a metodologia apresentada por (DELGADO, 2002).

Pode-se destacar alguns pontos importantes a serem estudados futuramente. As sugestões são:

- Analisar e desenvolver outras estratégias de paralelismo além da estratégia mestre-escravo, aplicando outras formas de comunicação.
- Busca por formas de codificações mais adequadas para o processamento em paralelo.
- Realizar a paralelização com outros recursos computacionais, como por exemplo: GPU e *cluster* de Raspberry Pi.

Como resultado do trabalho, publicou-se um artigo no XXI Congresso Brasileiro de Automática - 2016, com o título “Abordagem Coevolutiva com Processamento Paralelo para a Obtenção de Sistemas Fuzzy”.

REFERÊNCIAS

ALBA, E.; TROYA, J. M. A survey of parallel distributed genetic algorithms. *Complex.*, John Wiley & Sons, Inc., New York, NY, USA, v. 4, n. 4, p. 31–52, mar. 1999. ISSN 1076-2787.

CANTU-PAZ, E. *Efficient and accurate parallel genetic algorithms*. [S.l.]: Springer Science & Business Media, 2000.

CORDÓN, O. *Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases*. [S.l.]: World Scientific, 2001.

DARWIN, C. *The origin of species*. [S.l.]: Lulu. com, 1872.

DELGADO, M. *Projeto automático de sistemas nebulosos: uma abordagem co-evolutiva*. Tese (Doutorado) — PhD thesis, DCA/FEEC/UNICAMP-Campinas/SP/Brasil, 2002.

DELGADO, M. R.; ZUBEN, F. V.; GOMIDE, F. Hierarchical genetic fuzzy systems. *Information Sciences*, Elsevier, v. 136, n. 1, p. 29–52, 2001.

EAGER, D. L.; ZAHORJAN, J.; LAZOWSKA, E. D. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, IEEE, v. 38, n. 3, p. 408–423, 1989.

FENG, G. A survey on analysis and design of model-based fuzzy control systems. *Fuzzy systems, IEEE Transactions on*, IEEE, v. 14, n. 5, p. 676–697, 2006.

FOGEL, D. B. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, IEEE, v. 5, n. 1, p. 3–14, 1994.

FOGEL, L. J. *Artificial Intelligence Through Simulated Evolution.[By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh*. [S.l.]: John Wiley & Sons, 1966.

GRIFFITHS, A. J. et al. Introdução à genética. In: *Introdução à genética*. [S.l.]: Guanabara Koogan, 2013.

HOFFMANN, F.; NELLES, O. Structure identification of tsk-fuzzy systems using genetic programming. In: CITESEER. *Proceedings of IPMU*. [S.l.], 2000. v. 99, p. 438–445.

HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* [S.l.]: U Michigan Press, 1975.

IYODA, E. M. Inteligência computacional no projeto automático de redes neurais híbridas e redes neurofuzzy heterogêneas. Biblioteca Digital da Unicamp, 2000.

JACOBSON, L.; KANBER, B. *Genetic Algorithms in Java Basics*. [S.l.]: Springer, 2015.

JANG, J.-S. R.; SUN, C.-T.; MIZUTANI, E. Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence. prentice hall, 1997.

JANIKOW, C. Z.; MICHALEWICZ, Z. An experimental comparison of binary and floating point representations in genetic algorithms. In: *ICGA*. [S.l.: s.n.], 1991. p. 31–36.

JAVED, A. et al. Towards scalable java hpc with hybrid and native communication devices in mpj express. *International Journal of Parallel Programming*, Springer, p. 1–31, 2015.

JONES, G. Genetic and evolutionary algorithms. *Encyclopedia of Computational Chemistry*, Citeseer, v. 2, p. 1127–1136, 1998.

JONG, K. A. D. Analysis of the behavior of a class of genetic adaptive systems. 1975.

KONFRST, Z. Parallel genetic algorithms: advances, computing trends, applications and perspectives. In: *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. [S.l.: s.n.], 2004. p. 162–.

LIMA, I.; PINHEIRO, C.; OLIVEIRA, F. S. *Inteligência artificial*. [S.l.]: Elsevier Brasil, 2014.

LUQUE, G.; ALBA, E. *Parallel genetic algorithms: theory and real world applications*. [S.l.]: Springer, 2011.

MAMDANI, E. H.; ASSILIAN, S. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, Elsevier, v. 7, n. 1, p. 1–13, 1975.

MICHALEWICZ, Z.; SCHOENAUER, M. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, MIT Press, v. 4, n. 1, p. 1–32, 1996.

MILLER, B. L.; GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, [Champaign, IL, USA: Complex Systems Publications, Inc., c1987-, v. 9, n. 3, p. 193–212, 1995.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998. 2, 3, 4, 5 p.

MORIARTY, D. E.; MIIKKULAINEN, R. Hierarchical evolution of neural networks. In: IEEE. *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. [S.l.], 1998. p. 428–433.

NELLES, O. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. [S.l.]: Springer Science & Business Media, 2013.

NUNES, L. F. *CronoSim - Uma Ferramenta Distribuída para Simulação de Eventos Discretos e Validação de Protocolos de Sincronização*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2016.

PEDRYCZ, W.; GOMIDE, F. *An introduction to fuzzy sets: analysis and design*. [S.l.]: Mit Press, 1998.

POTTER, M. A.; JONG, K. A. D. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, MIT Press, v. 8, n. 1, p. 1–29, 2000.

RECHENBERG, I. Cybernetic solution path of an experimental problem. Ministry of Aviation, Royal Aircraft Establishment, 1965.

ROSEN, L. *Open source licensing*. [S.l.]: Prentice Hall, 2005.

SCHWEFEL, H.-P. *Evolutionsstrategie und numerische Optimierung*. [S.l.]: Technische Universität Berlin, 1975.

TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, IEEE, n. 1, p. 116–132, 1985.

WANG, L.-X. *A course in fuzzy systems*. [S.l.]: Prentice-Hall press, USA, 1999.

ZADEH, L. A. Fuzzy sets. *Information and control*, Elsevier, v. 8, n. 3, p. 338–353, 1965.