ABORDAGEM CO-EVOLUTIVA COM PROCESSAMENTO PARALELO PARA A OBTENÇÃO DE SISTEMAS *FUZZY*

Airton Motoki Tamakoshi*, Jeremias Barbosa Machado*, Edmilson Marmo Moreira*, Fadul Ferrari Rodor*

* *Universidade Federal de Itajubá*
Itajubá, Minas Gerais

Emails: motoki@gmail.com, jeremias@unifei.edu.br, edmarmo@unifei.edu.br, fadulrodor@gmail.com

Abstract— This work aims to present a methodology to obtain nonlinear fuzzy models from training data. This methodology is based on optimization of fuzzy models through the use of genetic algorithm assembled to parallel processing techniques. This approach aims to improve the efficiency of the method ensuring a shorter processing time. The results obtained show that, after training, the system is able to model a nonlinear function and in addition improve the performance from the viewpoint for the necessary time used for such achievement.

Keywords— Nonlinear modeling, Fuzzy systems, Genetic algorithm, Parallel processing.

Resumo— Este trabalho tem como objetivo apresentar uma metodologia para a obtenção de modelos não lineares *fuzzy* a partir de dados de treinamento. Esta metodologia é baseada na otimização de modelos *fuzzy* por meio da utilização de algoritmo genético conciliado a técnicas de processamento paralelo. Esta abordagem visa melhorar a eficiência do método garantindo um tempo menor de processamento. Além disso, trata-se de uma metodologia fundamentada em um sistema de representação multiníveis, o que garante eficácia na obtenção dos modelos. Os resultados obtidos mostram que, após o treinamento, o sistema é capaz de modelar uma função não linear e, além disso, melhorar o desempenho do ponto de vista do tempo necessário para tal obtenção.

Palavras-chave— Modelagem não linear, Sistemas *fuzzy*, Algoritmo genético, processamento paralelo.

1 Introdução

A modelagem matemática de sistemas reais é importante em muitas áreas da ciência e tecnologia. Em engenharia, modelos são utilizados para análise, otimização, simulação e obtenção de sistemas de controle que demandam por processos de modelagem e identificação cada vez mais eficientes. Há uma demanda cada vez maior de soluções de modelagem para sistemas não-lineares (Nelles, 2001). Diante desta demanda e da complexidade dos modelos que representem adequadamente tais sistemas, tem-se encontrado na utilização de modelos baseados em teoria de conjuntos nebulosos, ou conjuntos *fuzzy*, uma solução eficiente (Yager and Filev, 1994).

No entanto, a obtenção de modelos *fuzzy*, em geral, se baseia no conhecimento de um especialista. Uma outra forma de obtenção de tais modelos se baseia em técnicas de otimização ou aprendizado de máquinas, especialmente com o emprego de técnicas de inteligência artificial (Nelles, 2001). Dentre estas técnicas, destam-se as utilizações de algoritmos genéticos ou algoritmos evolutivos. Algoritmo Genético (AG) é um algoritmo de busca estocástica, baseado no processo natural de seleção de indivíduos em uma população (Holland, 1975). Criado por John Henry Holland, o Algoritmo Genético é utilizado para solucionar problemas de otimização combinatória de diversas aplicações (Holland, 1975).

A aplicação do algoritmo genético clássico para a determinação de modelos *fuzzy* pode, em alguns casos, se tornar inviável devido à complexidade da representação do modelo via um único cromossomo e a eventuais dificuldades na otimização. Neste sentido, foi proposta por Delgado (2002) uma metodologia baseada em uma representação multiníveis que torna o processo de obtenção dos modelos *fuzzy* mais amigável. Mesmo assim, como citado pela própria autora em (Delgado, 2002), a extração de modelos em problemas de instâncias elevadas pode se inviabilizar devido à necessidade de um tempo elevado de processamento.

Uma das alternativas de otimização no processamento do algoritmo genético está relacionado a paralelização. A paralelização consiste na resolução de um problema dividindo-o ou ampliando a capacidade de processamento através da utilização de vários processadores ou máquinas em um cluster de estações de trabalho (Javed et al., 2015). A paralelização do AG tende a fazer com que o algoritmo apresente um tempo de processamento menor do que a necessária quando é realizado em um único processador de forma tradicional.

Com o avanço da eletrônica e da computação cada vez mais se utilizam computadores que apresentam múltiplos *cores*. Estes computadores possibilitam que vários programas funcionem ao mesmo tempo, ou de forma paralelizada, dentro do mesmo processador. Neste sentido, o objetivo deste trabalho é propor e apresentar a implementação de uma metodologia que seja capaz de possibilitar a paralelização da metodolo-

gia proposta por (Delgado, 2002) dentro de um mesmo computador com múltiplos cores de forma a melhorar a sua capacidade de processamento e diminuir seu tempo de execução.

Neste artigo é apresentado uma breve descrição teórica dos temas relacionados à metodologia proposta. Para analisar a eficiência de tal metodologia, pretende-se obter um modelo *fuzzy* que seja possível de aproximar uma função não-linear. O principal objetivo é realizar um comparativo no tempo de processamento do algoritmo inicialmente apresentado em (Delgado, 2002) com o algoritmo paralelizado proposto.

Este artigo esta organizado da seguinte maneira. Na seção 2, a modelagem por modelos *fuzzy* é revisada. Na seção 3, apresenta-se de forma resumida a teoria de algoritmos genéticos. Nas seções 4 e 5, descrevem-se o algoritmo inicial de obtenção dos modelos *fuzzy* e a proposta deste trabalho baseado em processamento paralelo, respectivamente. Na sequência (seção 6) apresentam-se os resultados e por fim, na seção 7 apresentam-se as conclusões.

2 Sistemas *Fuzzy*

A teoria de conjuntos nebulosos (*fuzzy*) foi inicialmente proposta por Zadeh (1965). Esta teoria permite tratar a incerteza da informação, sendo uma extensão da teoria de conjuntos clássicos. Um sistema *fuzzy* é formado por uma base de regras do tipo **Se** <*premissa*> **Então** <*conclusão*>, permitindo atribuir conclusões para diversas faixas de valor para as variáveis de estado do problema. Um sistema *fuzzy*, em geral, possui as seguintes etapas (Yager and Filev, 1994):

- Fuzzificação;
- Inferência da base de regras;
- Defuzzificação.

Formalmente, um conjunto nebuloso A do universo de discurso Ω é definido por uma função de pertinência: $\mu_A : \Omega \rightarrow [0, 1]$. Essa função associa cada elemento x de Ω um grau de compatibilidade com o conjunto A. A aplicação da teoria dos conjuntos nebulosos permite inferir que um determinado valor de Ω pode assumir participação parcial perante a um conjunto.

As funções de pertinência podem assumir diversos formatos, variando conforme o problema tratado. Os formatos mais comuns são: triangular, trapezoidal e Gaussiana (Yager and Filev, 1994).

Quanto a classificação dos modelos *fuzzy*, estes geralmente são classificados com relação ao tipo de consequente presente nas regras. Dentre os mais utilizados destacam-se os modelos de Mamdani e Takagi-Sugeno (Yager and Filev, 1994). Neste trabalho, são de especial interesse os modelos *fuzzy* Takagi-Sugeno.

2.1 Modelo Takagi-Sugeno

O modelo *fuzzy* Takagi-Sugeno (TS) é formado por regras do tipo "se-então". Entretanto, os antecedentes são variáveis linguísticas e os consequentes são funções das variáveis presentes nos antecedentes. Através da utilização deste modelo é possível descrever de forma aproximada funções ou sistemas dinâmicos não lineares (Nelles, 2001). A regra nebulosa no modelo Takagi-Sugeno é apresentada na forma:

$$\begin{aligned} R_j : Se\ x_1\ é\ A_1^j\ e\ x_2\ é A_2^j\ \ldots\ e\ x_n\ é A_n^j \\ então\ y_i = g_i(w_i, x)\ ,\ j = 1 \ldots m \end{aligned} \quad (1)$$

onde m é o número de regras, x_k são as variáveis de entrada, A_k^j são os conjuntos nebulosos, $g_j(w_j, x)$ é a função de saída com w_j sendo o vetor dos parâmetros do consequente e x o vetor de variáveis.

A função a ser empregada nos consequentes das regras pode ser linear ou não linear. A obtenção do valor final da função modelada pelo sistema *fuzzy* TS se dá via um processo de inferência normalmente resultado da combinação das funções locais presentes em cada regra ponderada pelo seu valor de ativação (Yager and Filev, 1994).

3 Algoritmo Genético

O Algoritmo Genético é um tipo específico de algoritmo evolutivo. Seu funcionamento se baseia em conceitos advindos da teoria da evolução e em características biológicas do processo de reprodução de seres vivos (Bäck et al., 2000).

Para seu funcionamento inicialmente é gerada aleatoriamente uma população de indivíduos. Para cada indivíduo é calculado seu grau de aptidão através da função de aptidão ou função de *fitness*. Após criar a população inicial, indivíduos são selecionados para a operação de *crossover* e mutação, gerando novos indivíduos. O processo de criação de indivíduos é repetido até o algoritmo encontrar um critério de parada (de Castro, 2006) geralmente ligado a capacidade do indivíduo de representar uma solução ótima ou subótima para o problema sob análise. A Figura 2 apresenta o fluxograma do funcionamento do algoritmo genético.

A execução do algoritmo genético depende da definição de alguns componentes tais como (de Castro, 2006):

- **Indivíduo ou Cromossomo**

  Indivíduos ou cromossomos são representações de possíveis respostas para o problema.

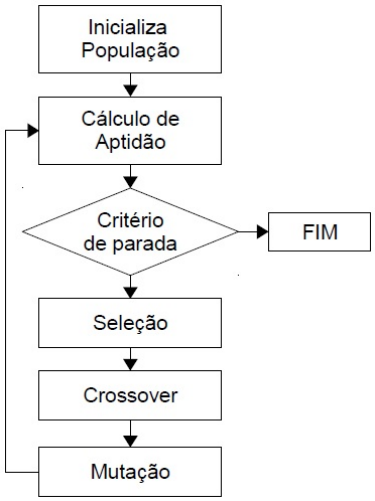

Figure 1: Fluxograma Algoritmo Genético

- **População**

  A população é composta por um grupo de indivíduos. A população permite explorar várias respostas e combinações de respostas para solucionar o problema de otimização.

- **Função de Aptidão e Grau de Aptidão**

  A função de aptidão (*fitness function*) é a função responsável em avaliar o grau de adaptação de um cromossomo à solução de um dado problema. A função varia conforme o problema a ser solucionado.

- **Seleção**

  A operação de seleção é responsável por escolher os bons cromossomos da população para realizar o cruzamento. Dentre todos os métodos de seleção, os mais comuns são: seleção por classificação (*rank selection*), seleção por roleta (*roulette wheel selection*), seleção por torneio (*tournament selection*) e seleção uniforme (*uniform selection*).

- **Cruzamento ou Crossover**

  A operação de cruzamento é responsável pela combinação de conteúdos dos cromossomos pais gerando um ou mais filhos. Dentre os métodos de cruzamento mais utilizados destacam-se: cruzamento em um ponto (*one-point crossover*), cruzamento em dois pontos (*two-points crossover*), cruzamento de três pais (*three parent crossover*), cruzamento uniforme (*uniform crossover*).

- **Mutação**

  A operação de mutação possui uma pequena probabilidade de ocorrer nos filhos gerados após o cruzamento. A mutação consiste em realizar pequenas alterações aleatórias no cromossomo filho permitindo explorar assim outras respostas ainda não exploradas.

4 Projeto Automático de Sistemas Nebulosos: CoevolGFS

Sistemas nebulosos baseados em regras usam teoria dos conjuntos para descrever um determinado sistema, seja ele um modelo, classificador ou mesmo um controlador. A obtenção de um modelo *fuzzy* pode se dar através de conhecimento de um especialista ou através de dados provenientes do sistema a ser modelado. O desenvolvimento de técnicas para extração automática e representação do conhecimento é motivado por situações onde a base de dados é extensa, sendo assim, uma tarefa complicada para um humano.

Um dos diversos métodos encontrados para a extração automática é o *genetic fuzzy system* (GFS), que é o uso do algoritmo genético no projeto de sistemas nebulosos. Nesta metodologia, o algoritmo genético é utilizado como ferramenta de otimização para a obtenção do sistema nebuloso.

Em algumas situações a aplicação de GFS pode se tornar inadequada. Isto pode ocorrer especialmente em problemas com grande dimensionalidade, o que acarretará em cromossomos de elevada dimensão, o que pode prejudicar a solução do problema por meio de GFS. Em (Delgado, 2002) é proposta uma solução para este problema com a partição da representação *fuzzy* em vários níveis hierárquicos. Desta forma cada nível é responsável por representar uma parte do modelo *fuzzy* sendo este mapeado em um respectivo cromossomo. A estrutura de níveis é descrita a seguir (Delgado, 2002):

4.1 Nível I - Funções de Pertinência

No nível I tem-se a representação da função de pertinência. O cromossomo apresenta uma sequência de cinco alelos com codificação real. O primeiro alelo S_k representa qual é o tipo da função de pertinência que é representado pelo cromossomo, podendo adotar os valores: 1 (trapezoidal), 2 (triangular) e 3 (Gaussiana).

4.2 Nível II - Regras Nebulosas

No nível II tem-se cromossomos com tamanho fixo e codificação inteira. Cada alelo possui um valor que representa um indivíduo da população de nível I. A quantidade de alelos do cromossomo de nível II é igual ao número de dimensões do problema tratado. É possível adotar valores nulos para os alelos, permitindo criar regras mais simplificadas.

4.3 Nível III - Bases Regras

Assim como no nível II, os cromossomos deste nível têm tamanho fixo e codificação inteira. Cada alelo possui um valor que representa um indivíduo da população de nível II. A quantidade de alelos

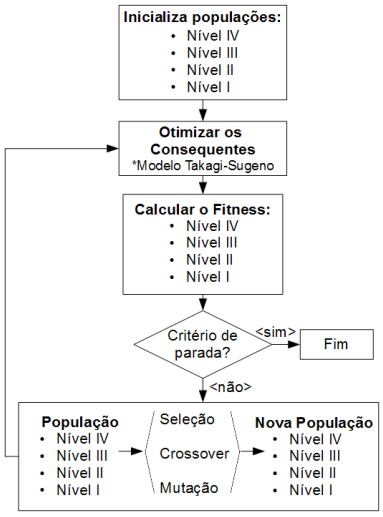

Figure 2: Fluxograma CoevolGFS

no nível III é igual ao número de regras que o sistema deve conter. Pode-se adotar valores nulos para diminuir o número de regras.

4.4 Nível IV - Sistema Nebuloso

Neste nível, os cromossomos possuem nove alelos de codificação mista. Sua representação garante a codificação para os modelos Mamdani e Takagi-Sugeno. Os nove alelos são:

1. Inferência (Escalonada);
2. Agregação dos Antecedentes (t-norma);
3. Parâmetro p_t;
4. Semântica da Regra (norma);
5. Parâmetro associado;
6. Agregação das Regras;
7. Defuzificação;
8. Índice da Base de Regras;
9. Índice da Partição Nebulosa.

Os parâmetros dos consequentes das regras em um modelo TS são obtidos via método de mínimos quadrados conforme apresentado em (Delgado, 2002).

Como se trata de um processo multiníveis, em cada um destes níveis haverá uma população e consequentemente um processo evolutivo associado. Como a evolução de um nível depende do efeito deste nível nos demais, denominou-se este processo como um processo coevolutivo. A estrutura geral do método pode ser resumida pelo fluxograma apresentado na Figura 2. Maiores detalhes sobre o método podem ser encontrados em (Delgado, 2002).

5 Proposta

Neste artigo é proposta a obtenção de modelos *fuzzy* Takagi-Sugeno através de uma variação do processo CoevolGFS (Delgado, 2002). A metodologia CoevolGFS inicialmente proposta apresenta algumas desvantagens em sua implementação. Uma destas desvantagens, conforme citado pela própria autora, está relacionada ao tempo necessário para o processamento do algoritmo genético. Desta forma, neste artigo propõe-se uma solução para este problema através da aplicação de métodos de computação paralela no algoritmo original.

A paralelização deste método é baseada no paradigma mestre-escravo. Atualmente, é um dos paradigmas mais populares, devido às diversas possibilidades de explorar o paralelismo nos modernos sistemas computacionais, além de proporcionar algoritmos paralelos cujo entendimento é intuitivo (Yuan et al., 2016). A versão paralela funciona da seguinte forma: um processo principal (mestre) é criado sendo responsável em gerenciar o sistema, os demais processos criados (escravos) são responsáveis em executar tarefas enviadas pelo mestre.

O diagrama da figura 2 ilustra o funcionamento básico do CoevolGFS. O que se propõe neste trabalho é a paralelização das etapas "Otimizar os Consequentes" e "Calcular o *Fitness*". Estas duas etapas foram escolhidas para serem paralelizadas tendo em vista o maior custo computacional envolvido em ambas. Analisando-se o tempo de execução das tarefas envolvidas no algoritmo CoevolGFS, pode-se verificar que a etapa denominada "Otimizar os Consequentes" consome a maior parte do tempo de processamento de todo o algoritmo. Isso acontece devido aos cálculos matriciais existentes no procedimento de otimização do modelo Takagi-Sugeno (método de mínimos quadrados global).

Outra etapa de elevado custo computacional é aquela denominada "Calcular o *Fitness*". Nesta etapa, é paralelizado o cálculo do *fitness* dos indivíduos de nível IV (nível do sistema *fuzzy*). Tal cálculo utiliza o valor real de saída e o valor de saída estimado pelo sistema *fuzzy*. Já os demais níveis têm seus valores de *fitness* calculados de forma sequencial por serem dependentes dos indivíduos de nível superior. O cálculo do *fitness* ocorre em ordem decrescente do ponto de vista hierárquico, ou seja, inicia-se com cálculo dos indivíduos de nível IV.

Com relação ao restante do método, manteve-se sua abordagem sequencial de forma normal, pois em geral estas etapas demandam um custo computacional irrelevante frente aos processos citados acima.

Para implementar este método, foi utilizada a plataforma Java (versão 1.8) com a biblioteca

MPJ Express. Esta biblioteca é um sistema de troca de mensagens que permite aos desenvolvedores de aplicativos paralelizar seus códigos Java sequenciais para serem executados em *clusters* de computação de alto desempenho e processadores *multicore* (Javed et al., 2015).

Neste trabalho, os métodos das classes disponíveis na biblioteca MPJ foram utilizados para estabelecer a comunicação de dados entre o processo mestre e os demais processos escravos. No início da fase de "Otimização dos Consequentes", o processo mestre envia um indivíduo de nível IV para cada processo escravo que, por sua vez, realiza a otimização, calcula o *fitness* correspondente e envia este resultado para o processo mestre. Enquanto há indivíduos para serem otimizados, o processo mestre envia estes indivíduos para os escravos, conforme eles vão devolvendo os resultados parciais. O processo mestre, a partir dos resultados de otimização de todos os indivíduos, calcula os respectivos *fitness* dos indivíduos de nível III, II e I.

Na seção a seguir serão apresentados resultados a fim de demonstrar a validade da metodologia proposta.

6 Simulação e Resultados

Para ilustrar a aplicação e a validade da metodologia proposta, realizou-se a obtenção de um modelo *fuzzy* para a função multivariável apresentada na equação (2):

$$f(x_1, x_2) = \frac{sen(x_1)}{x_1}\frac{sen(x_2)}{x_2} \qquad (2)$$

A Figura 3 apresenta a superfície gerada pela função $f(x_1, x_2)$ apresentada na equação 2.

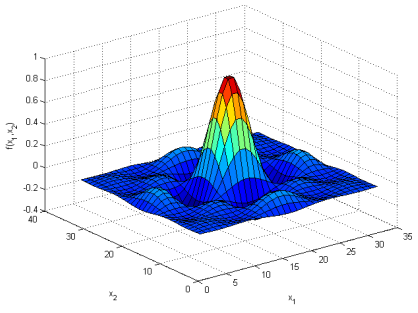

Figure 3: Função $f(x_1, x_2)$

A obtenção do modelo *fuzzy* se dá a partir de um conjunto de N dados de treinamento $(x_{1,i}, x_{2,i}, f_i)$ com $i = 1, \ldots, N$.

Para o processo de treinamento e obtenção do modelo utilizou-se os seguintes parâmetros no algoritmo genético:

- Seleção primária: Torneio;

  Tamanho 10%; Taxa de escolha 50%;

- Crossover: Single Point;

- Tamanho das populações: 50.

Conforme proposto neste artigo pretende-se verificar o impacto da paralelização do Coevol-GFS utilizando de 1 a 8 núcleos de processamento disponíveis na mesma máquina. As características do computador utilizado são apresentadas na Tabela 1.

Table 1: Características do Computador

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

Para o teste dedicado à avaliação para um determinado número de núcleos realizou-se um total de 100 experimentos para cada configuração. A Tabela 2 apresenta um resumo dos resultados obtidos com os experimentos realizados com relação ao tempo de processamento.

Table 2: Tempo x Número de núcleos

Núcleos	T. Médio [ms]	Desv. Padrão [ms]
1	74797	7600
2	79425	8240
3	43872	3218
4	32807	2192
5	28743	1952
6	26489	1757
7	24480	1247
8	23932	1145

A Tabela 3 apresenta os resultados de erro quadrático médio (EQM) para cada um dos testes realizados de acordo com o número de processadores utilizados.

Table 3: EQM x Número de núcleos

Núcleos	EQM Médio	Desv. Padrão
1	$1,7483.10^{-3}$	$3,1881.10^{-4}$
2	$1,8793.10^{-3}$	$4,2183.10^{-4}$
3	$1,8292.10^{-3}$	$3,0977.10^{-4}$
4	$1,7124.10^{-3}$	$3,0698.10^{-4}$
5	$1,7532.10^{-3}$	$2,6425.10^{-4}$
6	$1,7604.10^{-3}$	$3,2206.10^{-4}$
7	$1,7881.10^{-3}$	$3,1920.10^{-4}$
8	$1,7741.10^{-3}$	$3,5294.10^{-4}$

Na avaliação de um sistema paralelo, duas medidas de interesse são *Speedup* e Eficiência. O *Speedup* é definido como sendo a razão entre o tempo de execução de um programa num único processador em relação ao tempo obtido quando n processadores estão disponíveis. A Eficiência é definida como sendo a razão entre o *Speedup* e o número dos processadores utilizados. A eficiência indica a taxa de utilização dos processadores (Eager et al., 1989). Os valores de *Speedup* e Eficiência estão presentes na Tabela 4.

Table 4: Speedup e Eficiência

Núcleos	Tempo [ms]	Sp	Ef %
1	74797		
2	79425	0,9417	47,0865
3	43872	1,7048	56,8297
4	32807	2,2799	56,9977
5	28743	2,6022	52,0453
6	26489	2,8237	47,0616
7	24480	3,0554	43,6490
8	23932	3,1253	39,0674

Como pode-se observar pelos dados apresentados nas Tabelas 2 e 3 o algoritmo apresentou melhoras significativas conforme foi sendo incrementado o número de processadores. A queda relativa descrita pela medida eficiência (Ef) é explicada pelo aumento da comunicação entre o processo mestre e os escravos e pela forma circular em que indivíduos são enviados do mestre para os escravos. O valor do *speedup* tende a estabilizar em função do número de indivíduos criados no sistema, que no caso deste trabalho foram 50.

É importante destacar que o aumento no tempo total de execução verificado com dois processadores (79425 ms) em relação ao tempo com um único processador (74797 ms) ocorreu devido ao fato da versão sequencial não necessitar de comunicação entre os núcleos e, desta forma, não utilizar a biblioteca MPJ. Com dois processadores, por sua vez, há o custo computacional no cálculo de otimização dos consequentes, que ocorre em apenas um núcleo (o que contém o processo escravo). Entretanto, a partir de 3 núcleos de processamento, pela carga deste cálculo ser divida em mais de um escravo, o aumento do *speedup* é significativo.

Com relação à precisão do método na modelagem da função por meio de um sistema *fuzzy* pode-se verificar através da Tabela 4 que este se manteve eficiente mesmo com a alteração no algoritmo. Pode-se perceber que não há comprometimento na eficácia do ponto de vista da identificação quando se utiliza o processamento paralelo.

7 Conclusão

Este artigo apresenta uma abordagem paralela para arquiteturas *multicore* do algoritmo CoevolGFS proposto por Delgado (2002). O algoritmo inicialmente proposto apresenta vantagens na obtenção de mo-delos *fuzzy* via otimização por algoritmo genético quando comparado a técnicas clássicas aplicadas a mesma função. No entanto, demanda um esforço computacional significativo para a solução do problema. Neste contexto, a abordagem aqui apresentada alcançou um desempenho relevante justificando a necessidade de paralelização do método. Sendo assim, considerando-se as arquiteturas computacionais atuais com múltiplos núcleos, a implementação desenvolvida se mostra ainda mais vantajosa, pois esta não demanda a aquisição de equipamentos específicos, mas somente a utilização de recurso computacional já disponível. Além disso, a metodologia apresentada e as ferramentas utilizadas apresentam flexibilidade possibilitando a sua fácil adaptação em arquiteturas computacionais mais robustas como *clusters* de alto desempenho, grades computacionais e até mesmo uma estrutura de nuvem.

Os resultados experimentais apresentados mostram a eficiência do método proposto.

8 Agradecimentos

Os autores agradecem à FAPEMIG, à CAPES e ao CNPQ pelo apoio financeiro.

References

Bäck, T., Fogel, D. B. and Michalewicz, Z. (2000). *Evolutionary Computation 1: Basic Algortithms and Operators*, 1st edn, Institute of Physics Publishing, Bristol and Philadelphia.

de Castro, L. N. (2006). *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*, 1st edn, Chapman and Hall/CRC Computer and Information Science Series.

Delgado, M. R. B. S. (2002). *Projeto Automático de Sistemas nebulosos: uma Abordagem Co-Evolutiva.*, PhD thesis, Universidade Estadual de Campinas. Campinas, SP.

Eager, D. L., Zahorjan, J. and Lazowska, E. D. (1989). Speedup versus efficiency in parallel systems, *IEEE Transactions on Computers* **38**(3): 408–423.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Javed, A., Qamar, B., Jameel, M., Shafi, A. and Carpenter, B. (2015). Towards scalable java hpc with hybrid and native communication devices in mpj express, *International Journal of Parallel Programming* pp. 1–31.

Nelles, O. (2001). *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*, Springer-Verlag.

Yager, R. R. and Filev, D. P. (1994). *Essentials of Fuzzy Modeling and Control*, John Wiley and Sons.

Yuan, X., Zhang, T., Dai, X. and Wu, L. (2016). Master–slave model-based parallel chaos optimization algorithm for parameter identification problems, *Nonlinear Dynamics* **83**(3): 1727–1741.

Zadeh, L. A. (1965). Fuzzy sets, *Information and control* **8**(3): 338–353.