Centro Federal de Educação Tecnológica de Minas Gerais
Universidade Federal de São João del-Rei
PPGEL - Programa de Pós-Graduação em Engenharia Elétrica

Vitor Vidal de Negreiros

Redução do custo computacional na síntese de controladores PI multivariáveis utilizando algoritmo evolutivo combinado com redes neurais

Belo Horizonte
2020

Vitor Vidal de Negreiros

Redução do custo computacional na síntese de controladores PI multivariáveis utilizando algoritmo evolutivo combinado com redes neurais

Dissertação apresentada no âmbito do PPGEL como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia elétrica.
Área de concentração: Modelagem e Controle de Sistemas - Linha de pesquisa: Sistemas de Controle.

Orientador: Eduardo Nunes Gonçalves

Belo Horizonte
2020

Agradecimentos

Agradeço,

Em primeiro lugar, a minha família, na figura de meus pais, Dilsa e Paulo, e de meu irmão, Henrique, por sempre estarem presentes e por todo o apoio que sempre proporcionaram.

Ao meu orientador, professor Dr. Eduardo Nunes Gonçalves, por seus ensinamentos, paciência e trabalho em equipe.

Ao Departamento de Engenharia Elétrica do CEFET/MG pelo suporte e recursos fornecidos para o desenvolvimento desta dissertação e pela excelência de seu quadro de professores.

A todos os professores que compartilharam seus conhecimentos, seja em sala de aula ou fora dela.

Ao apoio das agências CAPES, CNPq e FAPEMIG.

A todos aqueles que de forma direta ou indireta contribuíram para a realização deste trabalho.

Negreiros, Vitor Vidal de

N385r Redução do custo computacional na síntese de controladores PI multivariáveis utilizando algoritmo evolutivo combinado com redes neurais. / Vitor Vidal de Negreiros. – Belo Horizonte, 2020.
93 f. : il.

Dissertação (Mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais, Programa de Pós-Graduação em Engenharia Elétrica em associação ampla com a Universidade Federal de São João Del Rei, 2019.
Orientador: Prof. Dr. Eduardo Nunes Gonçalves

Bibliografia

1. Controladores PI. 2. Redes Neurais (Computação). 3. Otimização Multiobjetivo. I. Gonçalves, Eduardo Nunes. II. Centro Federal de Educação Tecnológica de Minas Gerais. III. Título

CDD 621.3981

Elaboração da ficha catalográfica pela Bibliotecária Elisângela Gonçalves Barbieri CRB-6: 2624 / CEFET-MG

Resumo

A síntese de controladores PI multivariáveis por meio de otimização é uma estratégia efetiva e que tem sido amplamente testada graças à grande variedade de algoritmos disponíveis. Esse processo tem como etapas fundamentais a modelagem do problema a ser otimizado e a escolha do método de otimização aplicado. A escolha da função objetivo tem reflexos no custo computacional e nas características do sistema em malha fechada obtido. O método de otimização, por sua vez, impacta na capacidade de se encontrar ou não uma solução satisfatória em tempo computacional aceitável. Este trabalho apresenta uma variante do método Evolução Diferencial que lança mão de redes neurais artificiais para reduzir o custo computacional do processo de síntese de controladores PI multivariáveis. O algoritmo é testado com funções de *benchmark* clássicas da área de otimização para sua validação e estudo. Dois sistemas típicos da área de controle, ambos com múltiplas entradas e saídas e atrasos de transporte são usados como exemplos para sintonia de controladores pelo método Evolução Diferencial clássico e pela nova versão. Os resultados mostram os benefícios e pontos de atenção em relação ao método proposto, além de comparar, como objetivo secundário, o impacto da utilização de funções objetivo baseadas em normas de sistemas e de sinais no procedimento de síntese. O método proposto diminui o número total de cálculos da função objetivo, reduzindo o custo computacional do processo de otimização. Em contrapartida, observou-se uma maior variância e menor taxa de convergência do algoritmo modificado. As funções objetivo baseadas em normas de sinais apresentam o melhor desempenho de controle, enquanto funções objetivo baseadas em normas de sistemas apresentam menor custo computacional.

Palavras-chave: Otimização em Sistemas de Controle e Automação. Controle MIMO. Controle PI centralizado. Método Evolução Diferencial. Redes neurais artificiais;

Abstract

The synthesis of multivariable PI controllers by means of optimization is an effective strategy that has been widely used thanks to the variety of optimization methods and its great capacity of finding good solutions. This procedure has two fundamental steps, the modeling of the problem to be optimized and the choice of the optimization method to be applied. The objective function has direct influence over the computational cost and on the characteristics of the closed-loop system. The algorithm to be utilized, by its turn, impacts on the capacity of finding or not good solutions with an acceptable computational cost. This work presents a variant of the Differential Evolution method which takes advantage of artificial neural networks to reduce the computational cost of the synthesis process for multivariable PI controllers. The new algorithm is tested with classical benchmark functions of the optimization area for its validation and study. Two typical systems of the control area, both with multiple inputs and outputs and with transport delays are used as examples of the application of the new algorithm to the synthesis of multivariable PI controllers. The results show the benefits and drawbacks of the proposed method. Besides that, as a secondary objective, the differences of using objective functions based on systems and signals norms are analyzed and discussed. The proposed method reduces the total number of objective function evaluations, decreasing the computational cost. However, it was observed an increase in the results variance and a decrease in convergence rate. The objective functions based on signal norms present a better control performance, while objective functions based on system norms have a lower computational cost.

Key-words: Optimization on Control and Automation systems. MIMO control. Centralized PI. Differential Evolution Method. Artificial Neural Networks;

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ISE	*Integral Square Error* (Integral do Erro ao quadrado)
IAE	*Integral Absolute Error* (Integral do Erro absoluto)
ITSE	*Integral Time Square Error* (Integral do erro ao quadrado multiplicado pelo tempo)
ITAE	*Integral Time Absolute Error* (Integral do erro absoluto multiplicado pelo tempo)
MIMO	*Multiple-input Multiple-output* (Múltiplas entradas e múltiplas saídas)
SISO	*Single-Input Single-Output* (Uma entrada e uma saída)
MPC	*Model Predictive Control* (Controle preditivo baseado em modelo)
ANN	*Artificial Neural Network* (Rede Neural Artificial)
IMC	*Internal Model Control* (Controle de Modelo Interno)
RGA	*Relative Gain Array* (Matriz de Ganhos Relativos)
RNGA	*Relative Normalized Gain Array* (Matriz de Ganhos Relativos Normalizados)
BLT	*Biggest Log Modulus Tunning*
DE	*Differential Evolution* (Evolução Diferencial)
PSO	*Particle Swarm Optimization* (Otimização por Enxame de Partículas)
ABC	*Artificial Bee Colony* (Colônia artificial de Abelhas)
ACO	*Ant Colony Optimization* (Otimização por Colônia de Formigas)
QUERO	Querosene
LGO	*Light Gas-Oil* (gasóleo leve)
HGO	*Heavy Gas-Oil* (gasóleo pesado)

Lista de símbolos

H_2	Norma H_2
H_∞	Norma H_∞
$K(s)$	Especialmente utilizada para denotar o controlador PI multivariável
$arg\,min_s\, F(s)$	Denota os valores de s que minimizam $F(s)$
$\star$	Indica valor ótimo
$T_r(s)$	Especialmente utilizada para representar um sistema de referência
$T_{zw}(s)$	Especialmente utilizada para representar um sistema em malha fechada
$\|G(s)\|_\infty$	Norma H_∞ de uma da função de transferência G(s)
$\|G(s)\|_2$	Norma H_2 de uma função de transferência G(s)
ω_n	Frequência natural não amortecida de um sistema de segunda ordem
ζ	Fator de amortecimento de um sistema de segunda ordem
τ	Constante de tempo de um sistema
CR	Constante de cruzamento do algoritmo Evolução Diferencial
F	Fator de mutação do algoritmo Evolução diferencial

Sumário

Capítulo 1

Introdução

Neste capítulo são apresentados a contextualização do problema que será investigado, a revisão da literatura pertinente à pesquisa, as motivações, objetivos do trabalho, contribuições e metodologia. Ao final, a organização geral da dissertação também é apresentada.

1.1 Definição do Problema

A síntese de controladores para sistemas multivariáveis e com atrasos de transporte usando-se otimização evolucionária pode demorar de poucos minutos até várias horas. A natureza da função objetivo e o método de otimização escolhido são fatores fundamentais para o custo computacional. Dado que o investimento em hardware demanda dispêndio financeiro e que os equipamentos físicos nem sempre estão disponíveis para utilização, diminuir o custo computacional por meio do desenvolvimento de algoritmos mais eficientes é importante para tornar o projeto de sistemas de controle baseado em otimização mais prático e permitir a obtenção de resultados com menor custo. Sendo assim, o problema atacado neste trabalho é o da redução do esforço computacional da síntese de controladores multivariáveis por meio da implementação de melhorias algorítmicas e da escolha adequada da função objetivo.

Em relação à formulação da função objetivo, além de impactar no tempo de execução, a função escolhida também impacta nas características do sistema em malha fechada. O projeto de controladores baseados em modelo de referência pode ser feito com diversas formulações da função objetivo. Em especial, há funções objetivo que são normas de sistemas, sendo as normas H_2 e H_∞ as mais comumente usadas. E também há funções que são normas de sinais, como os índices de desempenho ISE (Integral do erro ao quadrado) e IAE (Integral do erro absoluto). Esses índices são usados para medir o quão próximo a saída simulada do sistema está de um sinal de saída de referência definido *a priori*.

Dessa forma, além de endereçar o alto custo computacional do procedimento de síntese através da proposição de um novo operador para algoritmos evolucionários, o trabalho

também busca o entendimento das vantagens e desvantagens de cada formulação da função objetivo, tendo como foco a comparação entre funções objetivo baseadas em normas de sistema e em normas de sinais.

1.2 Revisão de Literatura

A revisão da literatura foi organizada em três seções, a primeira aborda as principais técnicas de controle multivariável, a segunda trata da aplicação de técnicas de otimização ao projeto de controladores e a última analisa a literatura referente ao desenvolvimento de algoritmos evolucionários mais eficientes.

1.2.1 Técnicas de controle multivariável

Grande parte dos sistemas industriais são do tipo MIMO (*multiple-input multiple-output*), isto é, são processos que possuem múltiplas entradas e múltiplas saídas. Em contraposição ao caso SISO (*single-input single-output*), esses sistemas, também chamados de *multiloop*, multimalhas ou multivariáveis, são mais difíceis de serem controlados devido às interações existentes entre as diferentes malhas de controle. Ou seja, o sinal de entrada aplicado a cada entrada causa perturbações em todas as saídas (Skogestad S. Postlethwaite, 1996).

Além da complexidade oriunda do acoplamento entre as malhas do sistema, a existência de atrasos de transporte também dificulta a sintonia de controladores e a estabilização do sistema em malha fechada.

O desenvolvimento de técnicas de análise e projeto de controladores MIMO aplicados a sistemas multivariáveis com atraso tem sido um grande foco da literatura. Dentre as estruturas que podem ser aplicadas, os controladores MPC (*Model Predictive Control*) e ANN-Fuzzy (baseados em redes neurais artificiais e lógica Fuzzy) têm recebido bastante atenção (Shah e Engell, 2011; Kramer e Morgado-Dias, 2018). Não obstante, possuem entendimento não trivial e sua aplicação prática nem sempre é tão direta como a dos controladores PI/PID.

Os tradicionais controladores PI/PID, por sua vez, são de fácil compreensão e implementação computacional. Devido a isso, as estruturas PI/PID multivariável continuam a ser essenciais para o controle de sistemas MIMO.

Para atacar o problema do acoplamento entre malhas e dos atrasos de transporte de forma eficiente, e ao mesmo tempo manter a simplicidade de estrutura e compreensão dos controladores, uma série de estruturas de controladores PI/PID multiloop foram desenvolvidas e assim classificadas: controlador centralizado, descentralizado com e sem desacoplador e esparso (Albertos, 2004a; Albertos, 2004b; Shen, Cai e Li, 2009).

No controle centralizado, para um sistema $N \times M$, isto é, N saídas e M entradas, o bloco de controle consiste em uma matriz de transferência de dimensões $M \times N$, em que cada elemento da matriz é dado por uma equação do tipo $K_{ij}(s) = Kp_{ij} + \frac{K_{I,ij}}{s}$, em que i representa é i-ésima entrada e j a j-ésima saída do sistema. Esse esquema é indicado para sistemas com alto grau de acoplamento entre as malhas, conforme indicado por Shen, Cai e Li (2009).

Referente a essa estrutura de controle, Shen, Sun e Xu (2014) apresentaram um método baseado em funções de transferência equivalente, aplicado com bons resultados a sistemas 2×3 e 4×5. Lieslehto (1996) propôs uma generalização para o caso MIMO $N \times N$ do projeto IMC para sistemas SISO e Wang et al. (1997) estenderam o projeto de controladores usando a técnica *relay auto-tuning* para o caso multivariável.Kumar, Rao e Chidambaram (2012) desenvolveram um método de síntese direta aplicado a duas colunas de destilação 2×2 e em Park, Sung e Lee (2017) e V e Manickam (2014) foram introduzidas técnicas analíticas simplificadas aplicadas a sistemas 2×2.

No controle descentralizado, o sistema MIMO quadrado é pensado como um conjunto de sistemas SISO que podem ser controlados separadamente. Assim, o controlador tem uma estrutura diagonal, onde os blocos da diagonal do controlador PI/PID, $K_{ij}(s)$, são projetados especificamente para cada um desses sistemas SISO. Nesse esquema é essencial o cálculo do pareamento entrada-saída, ou seja, determinar qual entrada deve controlar qual saída. As técnicas mais usadas para determinar o melhor pareamento entrada-saída são as matrizes de ganhos relativos RGA (*Relative Gain Array*) e RNGA (*Relative Normalized Gain Array*) e suas variações (Witcher e McAvoy, 1977; Bristol, 1966; Tung e Edgar, 1981; He et al., 2009).

O controle descentralizado é mais simples do que o centralizado pois para um sistema de ordem N, apenas N controladores devem ser projetados, e não N^2 como no controle centralizado. No entanto, o preço a pagar pela estrutura mais enxuta é o desempenho não satisfatório quando a planta possui forte acoplamento entre malhas de controle e atrasos de transporte significativos.

Dentre as técnicas de síntese de controle descentralizado, tem-se as denominadas *detuning factor methods*, com destaque para a técnica BLT (*Biggest Log Modulus Tunning*), apresentada por Luyben (1986) e usada como referência na maioria dos artigos sobre controle descentralizado. A estratégia de fechamento de malhas sequencial (*Sequential Closing Loop*) é desenvolvida por Hovd e Skogestad (1994). Campestrini, Filho e Bazanella (2009), Halevi, Palmor e Efrati (1997) e Thyagarajan e Yu (2003) desenvolveram a técnica de *relay-feedback auto-tuning*. Por fim, Xiong, Cai e He (2007) lançam mão de funções de transferência equivalentes no projeto do controlador descentralizado.

Nos esquemas descentralizados com desacoplador, projeta-se um bloco para desacoplar

todas as interações entre as malhas de um sistema $N \times M$, fazendo com que o sistema resultante de dimensões $N \times N$ se torne um conjunto de sistemas SISO. O projeto do desacoplador é a primeira etapa nessa técnica de controle, sendo que num segundo momento o projetista determina N controladores PI/PID conforme uma técnica de projeto descentralizado de sua escolha.

Assim sendo, as técnicas para o projeto do controlador em si são as mesmas já citadas para o controle descentralizado, com a diferença de que deve-se projetar *a priori* um bloco desacoplador. O projeto de desacopladores também é um tópico da área de sistemas multivariáveis, tendo sido desenvolvidas várias metodologias em Luyben (1970), Shinskey (1990), Gagnon, Pomerleau e Desbiens (1998), Yang (2005), Wang, Huang e Guo (2000) e Cai et al. (2008).

Não obstante o controle com desacoplador resultar num desempenho melhor quando comparado com controle descentralizado puro, a inserção de um bloco adicional na malha do sistema (desacoplador) torna o sistema em malha fechada muito complexo, além de que o desempenho do desacoplador é muito dependente do quão próximo o modelo usado é do sistema real.

A abordagem de controladores esparsos é um compromisso entre a simplicidade da estrutura descentralizada e a maior complexidade do controlador centralizado. No controlador esparso, alguns blocos adicionais fora da diagonal são projetados e outros são fixados iguais a zero, o que diminui o número de ganhos que devem ser calculados por um lado, mas por outro coloca duas dificuldades extra: quais elementos fora da diagonal usar? E como projetar os demais blocos? Dentre as estratégias que se propõem a responder essas perguntas destacam-se o Controle Bloco diagonal (Zhang, Bao e Peter, 2003) e a estrutura de controle triangular (Commault e Dion, 1983). A grande desvantagem dos controladores esparsos é a dificuldade de sintonia. Shen, Sun e Xu (2014) desenvolveram uma técnica híbrida que usa elementos de controle descentralizado e esparso ao mesmo tempo.

Para uma revisão detalhada da teoria de controle multivariável o leitor pode consultar Skogestad S. Postlethwaite (1996) e suas edições mais recentes.

Na presente dissertação, a estrutura de controlador utilizada é a centralizada, conforme será discutido no capítulo 2. Com o desenvolvimento da computação evolucionária todas as principais estruturas do controlador multivariável também passaram a ser estudadas como um problema de otimização. Na próxima seção a literatura referente à otimização aplicada a síntese de controladores MIMO é revisada.

1.2.2 Otimização aplicada ao controle MIMO

A literatura da área de otimização evolucionária aplicada ao controle de sistemas consiste em trabalhos que apresentam novas metodologias de projeto e também na introdução de novos algoritmos e variações para melhorar formulações já existentes.

Devido ao grande número de diferentes aplicações de algoritmos evolucionários na área de controle, a revisão bibliográfica foi restrita aos trabalhos cujos modelos usados para desenvolvimento de controladores são sistemas precisamente conhecidos e com atrasos de transporte, que são modelos da mesma natureza dos problemas tratados na presente dissertação.

No projeto de controladores PI/PID MIMO baseados em otimização, três principais escolhas guiam o processo de síntese (Reynoso-Meza et al., 2012): a escolha da estrutura do controlador (descentralizado com ou sem desacoplador, centralizado ou esparso), a escolha do algoritmo de otimização e a determinação da função objetivo.

No que diz respeito à estrutura do controlador, tanto projetos de controladores descentralizados e centralizados são comuns na literatura, com esquemas esparsos sendo menos frequentes.

Os algoritmos de otimização utilizados são os mais variados, indo desde as versões clássicas e variações do Algoritmo Genético (Srinivas e Patnaik, 1994), Evolução Diferencial (Storn e Price, 1997), Enxame de Partículas (Kennedy e Eberhart, 1995), Colônia Artificial de abelhas (Karaboga e Basturk, 2007), Método dos vagalumes (Yang, 2009), até algoritmos desenvolvidos mais recentemente como *Extreme Optimization* (Boettcher e Percus, 2001) e *Grey Wolf* (Mirjalili, Mirjalili e Lewis, 2014). Muitos desses trabalhos têm como foco a comparação de diferentes algoritmos e a avaliação de sua aplicabilidade a problemas de controle, e não exatamente a análise da teoria de controle subjacente ao projeto e seus resultados.

Em Coelho e Pessoa (2011) e Coelho e Mariani (2012), são projetados controladores PID para a destilaria de Wood & Berry usando algoritmo DE (*differential evolution*) e *Firefly* (Método dos vagalumes), respectivamente. Em ambos os casos os autores acrescentam uma melhoria aos algoritmos usando mapas Zaslavskii e Tinkerbel para aumentar a diversidade das populações. Iruthayarajan e Baskar (2009) projetaram controladores PI e PID usando o método DE e outros 3 algoritmos evolucionários com foco em comparar o desempenho dos algoritmos em relação ao controle de sistemas multivariáveis com atraso. Iruthayarajan e Baskar (2010) aplicaram o algoritmo CMAES (*Covariance Matrix Adaptive Evolution Strategy*) ao projeto de controladores PI para as destilarias de Wood & Berry e Ogunnaike & Ray. O mesmo algoritmo é usado em Sivananaithaperumal e Subramanian (2014) para projetar um controlador PID de ordem fracionária, também aplicado aos mesmos modelos de destilaria. Zeng et al. (2015) usaram o algoritmo *Extremal Opti-*

mization para o projeto de estruturas PI/PID descentralizadas com e sem desacoplador, comparando os resultados com os obtidos por meio dos algoritmos DE, Genético e PSO. Em Soares, e Silva e Goncalves (2018) o algoritmo Evolução Diferencial é usado para síntese de controladores PI esparsos aplicados a sistemas multivariáveis não-quadrados. Em Bachur et al. (2017) uma nova abordagem para síntese de controladores para sistemas incertos usando otimização multiobjetivo é apresentada.

Em Devikumari e Velappan (2015) o Método dos Vagalumes e Enxame de Partículas são usados para o controle da destilaria de Ogunnaike & Ray por meio da estrutura descentralizada. Kadhar, Baskar e Amali (2015) projetaram um controlador descentralizado com desacoplador simples usando 4 versões do algoritmo DE. Os autores usaram a função sensitividade do sistema em malha fechada como função objetivo.

Uma discussão sobre a formulação do problema de projeto de controladores PI para a destilaria de Wood & berry usando otimização multiobjetivo é apresentado em Reynoso-Meza et al. (2012). Os mesmos autores dão continuidade ao trabalho em artigo de 2014 (Reynoso-Meza et al., 2014), no qual é feito um levantamento dos principais índices de desempenho usados para otimização de controladores PI. Apesar de a discussão ser feita no âmbito da otimização multiobjetivo, a análise comparativa dos índices de desempenho faz parte do escopo deste trabalho.

Três algoritmos em particular se destacam pelo grande volume de exemplos e aplicações. São eles: DE, Genético e Enxame de Partículas, incluindo as suas variantes. Para este trabalho foi escolhido o método DE. Nos capítulos 3 e 4 é feita uma apresentação detalhada do algoritmo DE e da versão desenvolvida como parte da pesquisa.

Do ponto de vista da escolha da função objetivo, as duas principais abordagens observadas nos trabalhos revisados são:

- Caracterizar a função objetivo como um problema de minimização das normas H_∞ ou H_2 de diferentes funções de transferência.

- Caracterizar a função objetivo como um índice de desempenho do domínio do tempo: ISE (Integral do erro ao quadrado), IAE (Integral do erro absoluto), ITSE (Integral do erro ao quadrado multiplicado pelo tempo) e ITAE (Integral do erro absoluto multiplicado pelo tempo)

Dentre as pesquisas que utilizam índices de desempenho, os mais utilizados são os índices ISE e IAE.

No contexto do controle baseado em modelo de referência, o objetivo de controle é fazer o sistema em malha fechada ser aproximadamente igual a um sistema de referência previamente definido. A utilização de modelos de referência para sintonia de controladores é por si só uma área da teoria de controle. Os conceitos fundamentais e trabalhos

seminais dessa área podem ser encontrados em Ferreira (1999), Ichikawa (1985) e Quinn e Sanathanan (1990). Exemplos da aplicação de otimização com essa abordagem podem ser encontrados em Arvani, Teshnehlab e Aliyari Sh. (2009) e em Ishizaki, Yubai e Hirai (2011), onde um controlador PI para um sistema 2×2 é projetado tendo como função objetivo a norma H_∞ da diferença do sistema em malha fechada e de um sistema de referência, uma formulação típica do problema de *Model Matching* e que também será revisada no capítulo 2 como parte do presente estudo.

O cálculo de normas de sistema difere da utilização de normas de sinais e tem grande impacto no tratamento matemático do problema quando a planta possui atrasos de transporte. Para esses problemas, a implementação com base em norma H_2 e H_∞ faz com que seja necessário aproximar os atrasos por funções de Padé, o que aumenta a ordem do sistema.

No que diz respeito à formulação do problema de otimização, Balestrino et al. (2006) fazem uma análise comparativa dos índices de desempenho ISE e IAE, concluindo que todos alcançam resultados similares para o caso SISO. Pareek e Gupta (2014) fazem uma análise comparativa dos índices IAE, ISE e ITAE utilizando algoritmo ABC (*Artificial Bee Colony*) para sintonia de um controlador PID SISO, indicando que os índices possuem resultados práticos semelhantes, com ITAE resultando em menor sobressinal e ISE nos menores tempos de subida. Sakthivel et al. (2015) projetaram um controlador PI para filtros ativos de harmônicos usando o método de Colônia de Formigas (ACO - *Ant Colony Optimization*), e compararam os quatro índices (ISE, IAE, ITSE e IAE), concluindo que o índice ISE usado como função objetivo resulta no melhor desempenho

Em Haji e Monje (2018) é projetado um controlador PID fracionário minimizando os índices ISE, IAE, ITAE e ITSE por meio do algoritmo do Morcego (*Bat Algorithm*) (Yang e Gandomi, 2012). Além de usar um algoritmo pouco conhecido na área de controle, o trabalho faz a análise comparativa das funções objetivo aplicadas para o caso MIMO. Os autores mostram que os índices influenciam no sobressinal e tempo de subida das respostas e, em geral, dependendo do algoritmo utilizado, o índice de desempenho que apresenta o melhor resultado nem sempre é o mesmo.

Os índices utilizados na formulação do problema de otimização impactam diretamente no controlador e no comportamento do sistema de malha fechada. A flexibilidade que a teoria de otimização dá ao projetista de poder formular o problema conforme os índices de sua preferência é uma grande vantagem, por outro lado, saber qual a melhor estratégia utilizar não é uma tarefa trivial, haja visto que a função objetivo escolhida tem impacto direto no resultado final e no tratamento computacional e matemático exigido pelo problema.

Como panorama geral da literatura atual, destaca-se a grande quantidade de algorit-

mos de otimização disponível e a alta frequência com que novos métodos de otimização e de projeto de controladores são desenvolvidos. Especificamente na área de controle, muitos autores se dedicaram a sintetizar controladores PI/PIDs centralizados e descentralizados para sistemas multivariáveis com foco em comparar o desempenho de diferentes algoritmos. As funções objetivo mais usadas com algoritmos evolucionários são índices de desempenho temporais (ISE, IAE, ITSE e ITAE) e as normas H_2 e H_∞.

1.2.3 Eficiência computacional dos métodos evolucionários

O fato dos métodos evolucionários serem baseados em populações de soluções candidatas que são avaliadas a cada iteração por meio da função objetivo, faz com que o esforço computacional para aplicação de otimização seja diretamente proporcional ao tempo necessário para avaliação da função objetivo. Tendo isso em vista, a redução do custo computacional dos algoritmos evolucionários têm seguido três linhas de desenvolvimento (Jin, 2005): a utilização de aproximações de funções, ou funções substitutas, as chamadas *surrogates*; a reformulação do problema original em um problema aproximado mais simples; e a utilização de aproximações de outras funções do processo de otimização que não a função objetivo, ou *fitness approximation*. Essa classificação foi proposta originalmente em Jin (2005) e usada como base para as revisões posteriores (Graning, Jin e Sendhoff, 2007; Santana-Quintero, Montano e Coello, 2010; Nakayama, Inoue e Yoshimori, 2006; Chugh et al., 2019).

A estratégia de aproximação de funções aplica a ideia de substituir a função objetivo original por uma aproximação que tenha tempo de cálculo significativamente menor. Assim, metamodelos como redes neurais, regressões polinomiais e aproximações de Kriging são usados para modelar a função objetivo original e substituí-la durante a execução do algoritmo.

A estratégia de reformular o problema original em uma aproximação simplificada consiste em reescrever o conjunto de equações e restrições original de forma a obter uma formulação mais simples e cujo custo computacional seja menor. Um exemplo são problemas de dinâmica de fluídos, em que as equações Navier-Stokes de terceira ordem são substituídas por equações de Euler de segunda ordem (Lee et al., 2008; Lattarulo, Seshadri e Parks, 2013). Na área de controle, um exemplo dessa estratégia é a utilização da realização mínima de um sistema em malha fechada ao invés de suas outras realizações canônicas possíveis de ordem maior.

Por fim, a terceira estratégia, a de aproximação de funções *fitness*, utiliza metamodelos para aproximar algum elemento ou comportamento do processo de otimização que não seja a função objetivo original. Aqui está incluído a utilização de metamodelos para ranquear ou classificar os indivíduos da população em subgrupos menores e, principalmente,

as estratégias de predição do valor da função objetivo a partir de dados previamente conhecidos. Nesse contexto se inclui a estratégia de *fitness inheritance* (Ducheyne, Baets e Wulf, 2008) e a utilização de algoritmos que comparam indivíduos da população corrente com indivíduos previamente conhecidos e armazenados. Assim, a cada iteração os elementos da população corrente são comparados aos elementos armazenados, e caso sejam semelhantes, o valor da função objetivo do indivíduo previamente conhecido é usado ao invés de chamar a função objetivo novamente.

Na revisão mais recente de Chugh et al. (2019), 45 algoritmos de otimização multiobjetivo que utilizam metamodelos para reduzir o custo computacional foram revisados. Algumas pontuações desse trabalho são importantes e portanto aqui elencadas.

- A maior parte dos algoritmos propostos na literatura para reduzir o custo computacional do processo de otimização usam a estratégia de aproximação de funções, sendo as estratégias de simplificação do problema original e de *fitness approximation* menos comuns.

- Uma das técnicas mais usadas para modelar e aproximar as funções ou comportamentos de interesse do projetista é a aproximação Kriging (Kleijnen, 2017), sendo aproximações por redes neurais, *Support Vector Regression* (Aytug e Sayin, 2009) e funções de base radial (Qasem et al., 2013) também comumente encontradas.

- O autor indica que alguns trabalhos baseados na estratégia de *fitness approximation* que usam metamodelos para classificar a qualidade dos indivíduos da população possuem resultados promissores. Especificamente, as pesquisas de Bandaru, Ng e Deb (2014), Seah et al. (2012), Loshchilov, Schoenauer e Sebag (2010a) e 2010b são destacadas.

- Embora a motivação de toda essa área de pesquisa seja a redução do custo computacional para solucionar problemas de dimensionalidade elevada e que demandam longo tempo de execução, a maioria dos algoritmos propostos são avaliados usando-se funções de *benchmark* da área de otimização cujo tempo de execução não se compara a problemas ilustrativos de sistemas reais.

Cabe mencionar ainda que em alguns trabalhos os termos *fitness approximation* e *function approximation* são usados como sinônimos, ambos denotando a utilização de modelos aproximados para substituir o cálculo da função objetivo original. Na classificação de Chugh et al. (2019), que é utilizada no presente contexto, os termos têm significados diferentes.

Eficiência computacional do método Evolução Diferencial

Os métodos evolucionários lançam mão de estratégias diferentes para evoluir uma dada população de soluções candidatas. As etapas de mutação, recombinação e seleção são comuns a várias técnicas evolucionárias, sendo que o mecanismo de implementação varia, mas a finalidade é a mesma. No caso do método Evolução Diferencial, que é o algoritmo escolhido nesta dissertação, os avanços e melhorias no método, tanto em relação à eficiência computacional quanto à sua capacidade de convergência para o mínimo global tem se baseado nas seguintes modificações:

- Adaptação contínua dos parâmetros CR e F (Qin, Huang e Suganthan, 2009)
- Novas estratégias de mutação (Islam et al., 2012)
- Controle adaptativo do tamanho da população (Yang et al., 2013)
- Novas técnicas de inicialização da população (Melo e Delbem, 2012)
- Novas técnicas de recombinação (Wang, Cai e Zhang, 2012)
- Variações com representação da população em espaços probabilísticos (Zhong e Zhang, 2012)
- Variações guiadas pela característica da população corrente (diversidade e topologia) (Zhong et al., 2013)
- Métodos com particionamento da população inicial e uso de clusterização (Mustafa et al., 2019)

Em Das e Suganthan (2011) e Das, Mullick e Suganthan (2016) é apresentado um panorama detalhado dos diferentes trabalhos que implementaram novas versões do método DE baseados nesses fatores. A grande maioria das variações do algoritmo DE focam em implementar modificações no processo de evolução da população, adotando mecanismos de adaptação dos parâmetros do algoritmo e novas estratégias de mutação e recombinação. No caso de projetos cuja função objetivo possui custo computacional muito alto, o principal fator que contribui para o grande tempo de execução é o número total de vezes que a função objetivo é calculada. Dessa forma, a etapa de seleção, que é onde a função objetivo é chamada, é uma etapa crucial para melhorar o custo computacional do algoritmo.

Algumas versões do algoritmo DE baseadas em metamodelos também foram apresentadas na literatura. Em Lu e Tang (2012) uma nova variante do DE foi desenvolvida. Os autores desenvolveram uma técnica para modelar a função objetivo que, de acordo com

o trabalho é especificamente sinérgica com os mecanismos de recombinação do método DE. Strofylas et al. (2018) e Lew et al. (2008) também apresentam versões que utilizam metamodelos no lugar da função objetivo original, ambos baseados em redes neurais. Liu e Sun (2011) e Park e Lee (2014), embora em anos distintos, apresentam versões muito semelhantes do algoritmo DE. Ambos utilizam a técnica *k-Nearest Neighbor* como metamodelo para substituir a função objetivo do problema de otimização. A vantagem dessa estratégia é que essa forma de predição não demanda um treinamento extensivo, como é o caso das redes neurais, por exemplo. Outra variante do DE é apresentada em Mlakar et al. (2015), onde além de substituir a função objetivo por um metamodelo, os autores introduziram um mecanismo para mitigar o efeito do erro de aproximação entre o metamodelo e a função objetivo original. Todos esses trabalhos usam a estratégia de aproximação da função objetivo original por um modelo cujo tempo de cômputo seja menor. Cabe destacar que os problemas para os quais essas variantes foram aplicadas não incluem problemas da área de engenharia, e praticamente todos os trabalhos dedicam esforços a melhorar e comparar as técnicas de metamodelos.

Como conclusão, ressalta-se que dentre os trabalhos estudados que usaram problemas de engenharia reais para avaliar a eficiência computacional dos métodos propostos, nenhum utilizou problemas da área de controle. Além disso, das melhorias apresentadas especificamente para o algoritmo DE, o foco da literatura tem sido o estudo e aplicação de novos metamodelos e da adoção de estratégias para mitigar os efeitos negativos do erro de aproximação em relação à função objetivo original.

1.3 Motivação

A motivação geral para a realização do trabalho é o alto custo computacional de projetos de controladores multivariáveis para sistemas de ordem elevada e com atrasos. A utilização de otimização para síntese dessa classe de controladores pode demorar desde algumas dezenas de minutos até várias horas, o que aumenta o custo do procedimento de síntese e torna o trabalho investigativo mais moroso e pouco prático. Uma alternativa para redução de tempo computacional é a aquisição de novas plataformas de hardware e técnicas de paralelismo. A compra de novos recursos físicos nem sempre é viável e consome recursos financeiros consideráveis, além de que a infraestrutura física, em geral compartilhada por várias pessoas, pode possuir taxa de disponibilidade abaixo do necessário dependendo do número de usuários e cronograma de manutenção. Assim sendo, o aumento da eficiência computacional dos métodos de otimização aplicados a síntese de controladores por meio de melhorias algorítmicas é importante e tende a se converter em ganhos de tempo e recursos.

A formulação da função objetivo e o número de vezes que ela é computada durante o processo de otimização são dois fatores-chave para reduzir o custo computacional. O problema de síntese de controladores pode ser formulado como otimização de normas de sinais ou de sistemas, cada qual com um custo computacional associado. Ao mesmo tempo, grande parte do custo computaciona se dá calculando a função objetivo para soluções que resultam em sistemas instáveis ou com baixo desempenho. Se for possível identificar *a priori* que uma solução tem baixa probabilidade de gerar um resultado satisfatório, essa solução poderia ser descartada sem a necessidade de calcular a função objetivo. Desse modo, é interessante propor uma metodologia que classifique as soluções antes de as mesmas serem avaliadas, sendo que para cada classe seja associada uma probabilidade de cálculo da função objetivo.

O presente trabalho visa melhorar a eficiência computacional do processo de otimização por meio da investigação de diferentes formulações da função objetivo, e principalmente pela proposição de uma versão modificada do algoritmo DE cujo número total de cálculos da função objetivo seja menor em comparação ao método original ao mesmo tempo que tenha desempenho satisfatório.

Do ponto de vista da literatura especializada, destacam-se os apontamentos de Chugh et al. (2019) de que estratégias que focam na avaliação da qualidade dos elementos de uma população tem mostrado resultados promissores. Além disso, dentre as modificações propostas especificamente para o método DE, poucos são os exemplos práticos na área de controle. A maior parte das pesquisas tem como objetivo melhorar os metamodelos para obtenção dos chamados *surrogate models*, isto é, aproximações da função objetivo que possam ser usadas como substitutas e que tenham menor dispêndio computacional. Dessa forma, há espaço para novas propostas que utilizem outras estratégias, particularmente para aplicações em problemas específicos da área de controle multivariável.

1.4 Objetivos do Trabalho

A realização dessa pesquisa tem os seguintes objetivos gerais:

- Propor e avaliar uma variante do algoritmo DE que minimiza o número de avaliações da função objetivo de modo a reduzir o custo computacional da síntese de controladores ótimos para sistemas MIMO.

- Desenvolver um mecanismo de seleção para o método DE que avalie a qualidade estimada dos indivíduos de uma população, classificando-os segundo a probabilidade dos mesmos gerarem resultados satisfatórios ou não.

- Comparar a utilização de normas de sinais e de sistemas, no contexto de controle

baseado em modelo de referência, para sistemas multivariáveis e com atrasos de transporte, e seus respectivos impactos no custo computacional e no desempenho do sistema em malha fechada.

1.5 Contribuições

As contribuições desta pesquisa são:

- Uma nova variante do algoritmo DE que possui custo computacional reduzido.
- A introdução na literatura de um estudo sobre melhoria do tempo computacional utilizando metamodelos que utilize exemplos práticos da área de controle multivariável.
- Apresentação de uma análise comparativa do projeto de controladores MIMO baseados em otimização evolucionária com funções objetivo baseadas em normas de sistemas e de sinais.

1.6 Metodologia

A metodologia utilizada consiste nas seguintes etapas:

- Revisão bibliográfica da literatura de otimização evolucionária aplicada ao projeto de controladores MIMO para plantas com atraso.
- Revisão bibliográfica da literatura sobre a diminuição do custo computacional dos métodos evolucionários, com foco no método Evolução Diferencial.
- Proposição, implementação e avaliação de versões modificadas do algoritmo DE em que a decisão de avaliação da função objetivo depende de uma probabilidade determinada por uma classificação baseada em redes neurais.
- Comparação de desempenho entre os algoritmos DE tradicional e modificado com diferentes configurações para a rede neural utilizada para cálculo da probabilidade de avaliação da função objetivo.
- Comparação dos resultados em termos das características temporais de desempenho e esforço computacional.

1.7 Organização do Documento

No capítulo 1 são feitas a introdução e a revisão bibliográfica. As motivações e objetivos do trabalho também são apresentados. No capítulo 2 a formulação dos problemas de controle é realizada e o conjunto de funções objetivo que será usado na sintonia dos controladores é definido. No capítulo 3 são apresentados os fundamentos do algoritmo DE, bem como a sua aplicação no contexto desta pesquisa e um estudo prático sobre o mecanismo de seleção. No capítulo 4 é proposto um operador baseado em redes neurais para o algoritmo DE desenvolvido para melhorar a sua eficiência computacional, além de uma série de experimentos comparativos. No capítulo 5 são apresentados os resultados e simulações para síntese de controladores dos sistemas definidos no capítulo 2. Por fim, no capítulo 6 são feitas as considerações finais e elencadas as linhas gerais para continuação da pesquisa.

Capítulo 2

Sintonia de controladores PI centralizados com normas de sinais e sistemas

Neste capítulo o problema de sintonia de controladores PI centralizados usando normas de sinais e sistemas é formulado. Os sistemas usados para aplicação do algoritmo DE modificado são apresentados, bem como as diferentes funções objetivo que são objeto de estudo do trabalho.

2.1 Formulação do problema

O controlador PI multivariável possui a seguinte estrutura:

$$K(s) = \begin{bmatrix} k_{11} & \dots & k_{1j} \\ \vdots & \ddots & \vdots \\ k_{i1} & \dots & k_{ij} \end{bmatrix} \tag{2.1}$$

Em que $k_{ij}(s)$ é dado por:

$$K_{ij}(s) = Kp_{ij} + \frac{K_{I_{ij}}}{s} \tag{2.2}$$

Sendo i o número de entradas da planta (número de linhas da matriz de transferência), j o número de saídas (número de colunas de matriz de transferência), Kp_{ij} e $K_{I_{ij}}$ os ganhos proporcional e integral, respectivamente, relacionando a i-ésima entrada com a j-ésima saída do sistema. O problema de sintonia do controlador PI centralizado pode ser escrito como o seguinte problema de otimização.

$$K^*(s) = \ arg\ min_k \ F_{obj}(K) \tag{2.3}$$

Sujeito a: $K(s) \ \in \ \mathcal{F}$, sendo $\mathcal{F}$ o conjunto de controladores com a estrutura especificada por (2.1) e (2.2) que estabiliza o sistema de controle em malha fechada e F_{obj} a função objetivo a ser otimizada.

A função objetivo pode ser formulada de diferentes maneiras, sendo que neste trabalho foi adotado um conjunto de funções objetivo baseadas no mesmo critério de fazer o sistema em malha fechada reproduzir o comportamento de um sistema ou sinal de referência.

Seja $T_r(s)$ um modelo de referência que possua a dinâmica desejada pelo projetista. Assim, define-se as seguintes funções objetivo:

- Norma H_∞ da diferença entre o sistema em malha fechada e o modelo de referência

$$J_{H\infty} = \|T_{zw}(s) - T_r(s)\|_\infty \tag{2.4}$$

- Norma H_2 da diferença entre o sistema em malha fechada e o modelo de referência

$$J_{H2} = \|T_{zw}(s) - T_r(s)\|_2 \tag{2.5}$$

- Integral ao quadrado de $E(t) = y_r(t) - y(t)$, em que $E(t)$ é a diferença de um sinal de saída de referência em relação ao sinal de saída simulado do sistema em malha fechada, tendo como *setpoint* uma função degrau unitário.

$$J_{ISE} = \int_0^{T_f} E(t)^2 dt \tag{2.6}$$

As funções objetivo (2.4) e (2.5) representam funções baseadas em um sistema de referência previamente definido. Assim, parte do trabalho de sintonia deve ser dedicado à determinação desse modelo de referência. Conforme apresentado em Silva, Bazanella e Campestrini (2018), a escolha do modelo de referência deve obedecer a certas restrições sob o risco do controlador resultante não ter o desempenho desejado. Por exemplo, o sistema de referência deve possuir os mesmos atrasos do sistema original e sua constante de tempo deve ser da mesma grandeza.

Já a função objetivo (2.6) compara o sinal de saída simulado do sistema em malha fechada com um sinal de saída de referência previamente disponível. Esse sinal de saída de referência pode ser a saída de um sistema de referência, o próprio *setpoint* do sistema ou um sinal considerado a saída típica para aquele sistema. Em projetos de otimização, comumente se utiliza a entrada do sistema como referência, definindo assim os índices de desempenho como ISE, IAE, ITSE e ITAE. A abordagem adotada aqui é usar um sinal de referência que possua uma dinâmica transitória que incorpore as especificações de controle (sobressinal, tempo de subida e de acomodação) e restrições físicas da planta (principalmente os atrasos de transporte), e não a entrada em degrau pura e simplesmente. Esse sinal pode ser a saída do sistema de referência ou a saída para a entrada típica do sistema, obtida de forma experimental.

Dessa maneira, ao usar um sinal ao invés de um sistema de referência, o projetista tem mais liberdade em definir as características da saída do sistema em malha fechada.

2.1.1 Sistema e sinal de referência

Para todos os exemplos ilustrativos, o sistema de referência possui a seguinte estrutura bloco diagonal:

$$T_r(s) = \begin{bmatrix} T_{r,1}(s) & 0 & \dots & 0 \\ 0 & T_{r,2}(s) & \dots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & 0 & T_{r,i}(s) \end{bmatrix} \tag{2.7}$$

Em que cada elemento $T_{r,i}$ da diagonal principal é uma função de transferência que especifica a relação de entrada e saída da i-ésima malha de controle. O modelo de referência deve ser uma matriz diagonal para garantir o desacoplamento entre as malhas de controle. Nesse sentido, é importante mencionar que a escolha de um controlador PI centralizado, e não um descentralizado, é preferencial, uma vez que a estrutura descentralizada não possui controladores fora da diagonal principal, que seriam os responsáveis por zerar os efeitos do acoplamento entre malhas.

Para cada exemplo ilustrativo os elementos $T_{r,i}$ devem ser determinados de forma que o modelo de referência atenda às limitações da planta real e ao mesmo tempo possua as especificações de controle desejáveis. Assim, os modelos de referência apresentados nas próximas seções possuem os mesmos atrasos de transporte das plantas originais e constantes de tempo condizentes com a natureza de cada problema.

Para poder comparar os resultados finais, todos os sistemas de referência introduzidos nas próximas seções são baseados no mesmo perfil transitório, apresentado em detalhes para cada sistema nas seções seguintes. O sinal de referência usado com a função objetivo (2.6) é a saída dos modelos de referência de cada caso, o que permite a comparação entre os resultados finais obtidos com todas as funções objetivo.

2.2 Sistema 2×2

O sistema 2×2 escolhido é o tradicional modelo de Wood & Berry (Wood e Berry, 1973), um sistema com alto acoplamento entre as malhas de controle e atrasos de transporte. O modelo é apresentado na figura 2.1.

Trata-se de uma coluna de destilação binária em que os produtos fracionados na parte inferior e superior da coluna são compostos de metanol, medidos em wt% de metanol

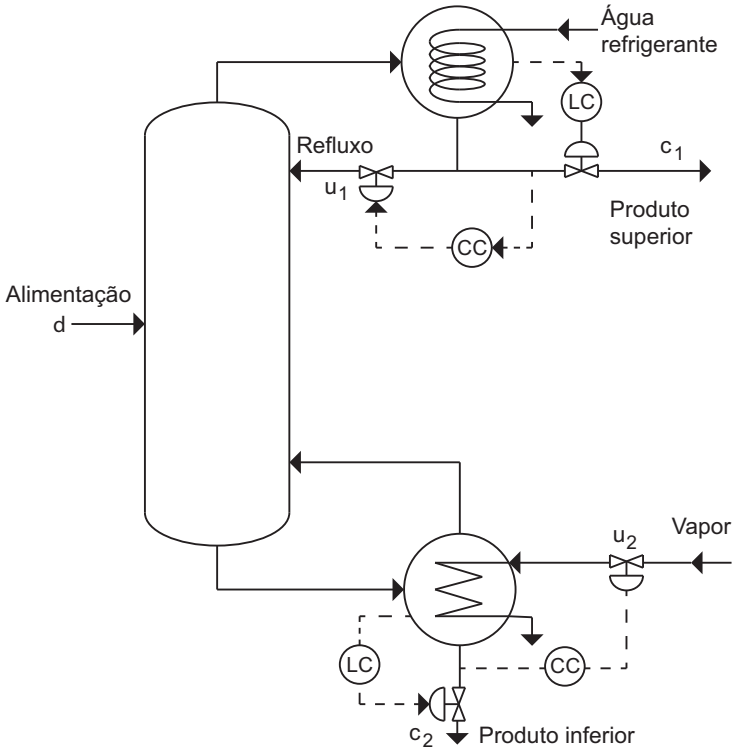


Figura 2.1: Coluna binária de destilação de Wood & Berry

(percentagem de peso por unidade de massa). As composições superior e inferior da coluna são as variáveis controladas. As variáveis manipuladas são as taxas de refluxo (u_1) e fluxo de vapor (u_2), que controlam, respectivamente, a composição superior da coluna e a composição inferior. As variáveis manipuladas são dadas em lb/min e as constantes de tempo do modelo são todas dadas em minutos. A matriz de transferência do sistema é dado por:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{12{,}8e^{-s}}{16{,}7s+1} & \frac{18{,}9e^{-3s}}{21s+1} \\ \frac{6{,}6e^{-7s}}{10{,}9s+1} & \frac{19{,}4e^{-3s}}{14{,}4s+1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.8}$$

Em que y_1 e y_2 são as composições finais em wt% de metanol e os sinais de entrada são a taxa de refluxo u_1 e a taxa de fluxo de vapor u_2. O problema de sintonia do controlador $K(s)$ para esse sistema consiste em um problema com 8 variáveis de otimização, em que os 4 controladores individuais que compõem $K(s)$ serão otimizados.

2.2.1 Síntese do controlador para o sistema 2×2

As funções de custo (2.4), (2.5) e (2.6) foram simuladas para um modelo de referência com a estrutura da equação (2.7). O modelo tem as funções de transferência da diagonal principal com a seguinte estrutura:

$$T_{r,i}(s) = \frac{\omega_{n,i}^2 e^{-\tau_{d_i} s}}{s^2 + 2\zeta_i \omega_{n,1} s + \omega_{n,i}^2} \tag{2.9}$$

Os valores das constantes estão na tabela 2.1

Tabela 2.1: Parâmetros do modelo de referência (2.9)

$T_{r,1}$			$T_{r,2}$		
$\omega_{n,1}$	ζ_1	$\tau_{d,1}$	$\omega_{n,2}$	ζ_2	$\tau_{d,2}$
0,22	1,7	1	0,22	1,7	3

O modelo de referência (2.1) foi determinado experimentalmente tendo em vista os critérios de sobressinal nulo, tempo de subida de 30 minutos e tempo de acomodação de 50 minutos.

2.3 Sistema 4×5

Como exemplo ilustrativo com maior número de entradas e saídas e que exige tempo de execução mais elevado que o sistema 2×2, será usado a coluna de destilação de petróleo bruto apresentada em Muske et al. (1991). A figura 2.2 mostra um diagrama simplificado da planta. A refinaria faz a destilação inicial do petróleo bruto em suas frações de refino baseado nas diferenças dos pontos de ebulição dos diversos constituintes do petróleo. Esta unidade separa o óleo bruto em seis frações: gás combustível, nafta, querosene (QUERO), gasóleo leve (LGO, *light gas oil*), gasóleo pesado (HGO, *heavy gas oil*) e óleo residual (RESID).

Na planta em questão, o petróleo bruto é bombeado em tanques de armazenamento, onde é feito dessalinização e pré-aquecimento, para em seguida ser parcialmente vaporizado em dois aquecedores paralelos. O vapor e o líquido dos aquecedores são então bombeados para a parte inferior da coluna de fracionamento. O gás combustível e a nafta deixam a parte superior da coluna como vazões de vapor e líquido. Os componentes QUERO, LGO e HGO são separados como frações intermediárias laterais. O óleo residual deixa a coluna como um produto da parte inferior.

A coluna possui três sistemas de remoção de calor: sistema de condensação do topo, bombeamento circular de nafta pesada e bombeamento circular de LGO.

Os sistemas de bombeamento circulares são projetados para nivelar o tráfego de líquido e vapor e maximizar a recuperação de calor. A temperatura do topo da torre é regulada pela manipulação externa da vazão de refluxo. Dessa forma, do ponto de vista de controle, têm-se as seguintes variáveis manipuladas: temperatura do topo (u_1), vazão de querosene (u_2), vazão de LGO (u_3), vazão de HGO (u_4), e temperatura de saída do aquecedor (u_5).

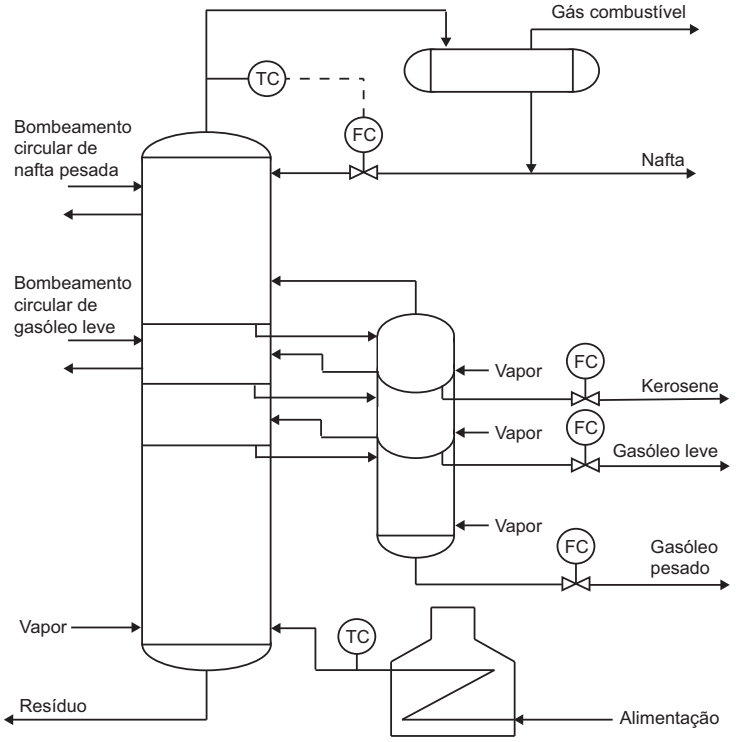


Figura 2.2: Coluna de destilação de Petróleo Bruto

As variáveis controladas são as temperaturas de corte: nafta/QUERO (y_1), QUERO/LGO (y_2), LGO/HGO (y_3), e a vazão de vapor d'água ou vazão de retificação (y_4). A variável y_4 é um indicador de excesso de calor latente na alimentação da coluna de destilação e, em geral, deseja-se manter sua faixa de variação próxima a zero (Muske et al., 1991).

O modelo do processo 4 x 5 é representado pela seguinte matriz de transferência:

$$G(s) = \begin{bmatrix} \frac{3{,}8(16s+1)}{140s^2+14s+1} & \frac{2{,}9e^{-6s}}{10s+1} & 0 & 0 & \frac{-0{,}73(-16s+1)e^{-4s}}{150s^2+20s+1} \\ \frac{3{,}9(4{,}5s+1)}{96s^2+17s+1} & \frac{6{,}3}{20s+1} & 0 & 0 & \frac{16se^{-2s}}{(5s+1)(14s+1)} \\ \frac{3{,}8(0{,}8s+1)e^{-s}}{23s^2+13s+1} & \frac{186{,}1(12s+1)e^{-s}}{37s^2+34s+1} & \frac{3{,}4e^{-2s}}{6{,}9s+1} & 0 & \frac{22se^{-2s}}{(5s+1)(10s+1)} \\ \frac{-1{,}62(5{,}3s+1)e^{-s}}{13s^2+13s+1} & \frac{-1{,}5(3{,}1s+1)}{5{,}1s^2+7{,}1s+1} & \frac{-1{,}3(7{,}6s+1)}{4{,}7s^2+7{,}1s+1} & \frac{0{,}6e^{-s}}{2s+1} & \frac{0{,}32(-9{,}1s+1)e^{-s}}{12s^2+15s+1} \end{bmatrix} \quad (2.10)$$

As constantes e atrasos de tempo estão todas em minutos.

2.3.1 Síntese do controlador para o sistema 4×5

As funções de custo (2.4), (2.5) e (2.6) foram calculadas para um modelo de referência com a estrutura da equação (2.7). As funções de transferência da diagonal principal possuem o seguinte formato:

$$T_{r,i}(s) = \frac{\omega_{n,i}^2 e^{-\tau_{d_i} s}}{s^2 + 2\zeta_i \omega_{n,1} s + \omega_{n,i}^2} \tag{2.11}$$

Os parâmetros do modelo (2.11) foram determinados experimentalmente, com objetivo de atenderem às mesmas características do modelo (2.1).

Tabela 2.2: Parâmetros do modelo de referência (2.11)

$T_{r,1}$			$T_{r,2}$			$T_{r,3}$			$T_{r,4}$		
$\omega_{n,1}$	ζ_1	τ_{d_3}	$\omega_{n,2}$	ζ_2	τ_{d_2}	$\omega_{n,3}$	ζ_3	τ_{d_3}	$\omega_{n,4}$	ζ_4	τ_{d_4}
0,22	1,7	0	0,22	1,7	0	0,22	1,7	1	0,22	1,7	2

Para esse sistema, a sintonia do controlador PI centralizado dado por (2.1) consiste em um problema de otimização com 40 variáveis de otimização, sendo 20 ganhos proporcionais e 20 ganhos integrais.

2.4 Considerações finais

Nos próximos capítulos são apresentados os métodos DE clássico e a versão modificada proposta. Experimentos práticos utilizando-se os dois problemas de controle apresentados nesse capítulo também foram feitos para comparar a dinâmica dos dois algoritmos e seus efeitos quando aplicados a problemas de controle.

Capítulo 3

Fundamentos do método DE

O presente capítulo apresenta os princípios fundamentais do método Evolução Diferencial, discute os parâmetros escolhidos para as execuções práticas e inclui um pseudo-código da versão implementada para referência dos leitores.

3.1 Operações básicas do algoritmo DE

O método Evolução Diferencial foi introduzido em Storn e Price (1997). Trata-se de uma técnica evolucionária que ganhou notoriedade devido a sua formulação simples, implementação prática e excelentes resultados com funções multimodais e não-diferenciáveis. Possui boa capacidade exploratória (investigar o espaço de buscas de forma abrangente) e também de convergência.

O fato de possuir poucos parâmetros de ajuste, sua simplicidade e, ao mesmo tempo, alta capacidade de otimizar funções complexas de forma eficiente, faz o DE uma técnica de excelência para aplicação em problemas de otimização. Desde a sua introdução, uma série de variações foram apresentadas. Uma revisão completa da evolução histórica e de todas as variantes disponíveis do algoritmo DE pode ser encontrada em Das, Mullick e Suganthan (2016).

O método utiliza N_p vetores de dimensão D como população base que é evoluída a partir do cálculo sucessivo das operações de mutação, cruzamento e seleção. Essas três operações são definidas a seguir em forma sequencial.

Seja a população inicial P dada por:

$$P = \left\{x_1, ..., x_{N_p}\right\} \tag{3.1}$$

Em que:

$$x_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{Di} \end{bmatrix}, \; para \;\; i = 1,2,...,N_p \tag{3.2}$$

Primeiro inicializa-se a população base P que será evoluída com a seguinte regra:

$$x_{j,i} = \underline{\lambda}_j + (\overline{\lambda}_j - \underline{\lambda}_j) \times rand\,, \;\; para \;\; j = 1,...,D \tag{3.3}$$

Em que *rand* gera um número aleatório entre 0 e 1 com distribuição de probabilidade uniforme e $\underline{\lambda}_j \leq x_{j,i} \leq \overline{\lambda}_j \;\; \forall \; j$, sendo $\underline{\lambda}_j$ e $\overline{\lambda}_j$ os limites inferiores e superiores para cada variável da solução inicial x_i. O algoritmo executa três operações básicas iterativamente até que o critério de parada seja satisfeito. As três operações do método DE são:

1. Mutação: para cada i-ésimo vetor pai, um vetor mutante é gerado de acordo com a seguinte regra:

$$v_{i,G+1} = x_{r_1,G} + F(x_{r_2,G} - x_{r_3,G}) \tag{3.4}$$

   em que r_1, r_2 e r_3 são todos índices aleatórios de 1 até N_P e sempre diferentes uns dos outros. Além disso, F pode ser um valor fixo, como usado em Bujok, Tvrdik e Polakova (2014) ou variar aleatoriamente em uma determinada faixa, como em Silva, Lopes e Guimaraes (2011). A implementação usada na dissertação usa F variando aleatoriamente entre 0,5 e 1, como usado em Eita e Shoukry (2014).

2. Cruzamento: um conjunto de N_p vetores candidatos é gerado a partir da recombinação de elementos dos vetores mutantes com elementos dos vetores da população pai, de acordo com a seguinte regra:

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} \;\; se \;\; rand \leq CR \;\; ou \;\; j = \delta, \\ x_{ji,G} \;\; caso \;\; contrrio \end{cases} \tag{3.5}$$

   Em que $j = 1,...,D$, $\delta = rnbr(D)$, para garantir que pelo menos uma variável da solução candidata venha da solução mutante, CR é a constante de Cruzamento, também entre 0 e 1, tendo sido adotado nesse trabalho $CR = 0{,}9$, conforme recomendado pela literatura (Draa, Bouzoubia e Boukhalfa, 2015). A função rnbr(D) gera um índice aleatório inteiro na faixa de 1 até D.

3. Seleção: avalia o valor da função objetivo para cada i-ésimo vetor candidato e para os vetores da população pai aplicando a seguinte regra:

$$x_{i,G+1} = \begin{cases} u_{i,G+1} \;\; se \;\; f(u_{i,G+1}) \leq f(x_{i,G}), \\ x_{i,G} \;\; caso \;\; contrrio \end{cases} \tag{3.6}$$

3.2 Método DE - Pseudocódigo

O seguinte pseudo-código exemplifica o método DE clássico, tal como foi implementado para este trabalho.

Algorithm 1 Método DE clássico

1: Inicializa parâmetros CR,D,N_p,G_{max}
2: Inicializa a população base de acordo com a equação (3.3)
3: **while** $G \leq G_{max}$ **do**
4: Operação mutação de acordo com a equação (3.4)
5: Operação cruzamento de acordo com a equação (3.5)
6: Operação seleção de acordo com a equação (3.6)
7: $G \leftarrow G + 1$
8: **end while**

Na próxima seção são apresentados os critérios para a escolha dos parâmetros do algoritmo e como os mesmos foram ajustados.

3.3 Ajuste do algoritmo e escolha dos parâmetros

As diferentes versões do algoritmo DE são definidas de acordo com as técnicas específicas usadas nas operações de cruzamento e mutação, e também no número de vetores usados na mutação. Essas duas operações são as que influenciam na capacidade de convergência e na diversidade do algoritmo.

A notação usada para classificar as diferentes variações do algoritmo é dada por *DE/ma/mb/mc*. Na notação, *ma* indica o método usado no passo de mutação, *mb* indica o número de vetores de diferença usados para criar os vetores mutantes (no algoritmo clássico é usado 1 vetor de diferença), e *mc* indica o mecanismo de cruzamento utilizado.

O algoritmo usado neste trabalho é o *DE/rand/1/bin*. O acrônimo bin quer dizer que a operação de cruzamento é baseada numa série de experimentos binomiais, isto é, o parâmetro *CR* define uma probabilidade fixa e todas as verificações de atendimento a esse limite da probabilidade feitas N_p vezes durante a operação de cruzamento são eventos independentes. Por fim, o acrônimo *rand* indica que os três vetores usados na mutação são escolhidos aleatoriamente dentre a população atual.

Os parâmetros que devem ser escolhidos pelo projetista são três, quais sejam: tamanho da população, N_p, número máximo de iterações (critério de parada), G_{max}, e fator de cruzamento, *CR*. O tamanho da população é dependente de cada tipo de problema.

Quanto maior o número de parâmetros a serem otimizados, a tendência é que seja necessária uma população maior. A literatura não é unânime em relação à melhor forma de determinar o tamanho da população, não obstante algumas das regras práticas sugeridas são: $5D \leq N_p \leq 10D$ (Storn e Price, 1997), $2D \leq N_p \leq 40D$ (Ronkkonen, Kukkonen e Price, 2005) e $3D \leq N_p \leq 8D$ (Gamperle, Muller e Koumoutsakos, 2002), em que D é o número de parâmetros a serem otimizados. A experiência e testes experimentais também são usados para ajustes finos, e interessantemente, o DE também pode funcionar bem com populações pequenas (menores que $1D$) (Neri e Tirronen, 2008; Weber, Tirronen e Neri, 2010). Embora as regras práticas sugeridas orientem as escolhas iniciais, o número exato deve ser definido com base em testes práticos (Piotrowski, 2017).

Para o fator de cruzamento não é recomendado usar valores próximos a 0 devido à necessidade de gerar populações com alto nível de diversidade. Recomenda-se usar valores maiores que 0,5. Para este trabalho, as simulações foram feitas com $CR = 0{,}9$. O parâmetro F usada na mutação está ligada à capacidade dos vetores mutantes estarem espalhados em um raio maior do espaço de buscas. Para este trabalho foi escolhido um valor de F diferente para cada vetor mutante, de acordo com a seguinte regra: $F = 0{,}5 + 0{,}5rand$, de forma que o mesmo varia aleatoriamente entre 0,5 e 1. Os critérios para escolha dos fatores CR e F foram baseados na literatura citada na seção anterior.

O critério de parada adotado é o número máximo de iterações, tendo sido determinado experimentalmente conforme explicado nos capítulos 4 e 5.

3.4 Estudo do comportamento da seleção no DE clássico

No contexto de problemas práticos, o maior custo computacional do método DE geralmente está na etapa de avaliação da função objetivo. Durante a seleção a função objetivo é avaliada para cada solução candidata para que o resultado seja comparado com os já existentes para a população pai. Tendo isso em vista, esse experimento foi realizado com objetivo de avaliar a quantidade de soluções candidatas que são descartadas no processo de seleção. Essas soluções candidatas implicam em custo computacional, e por serem descartadas, acabam por não contribuir diretamente no processo de otimização. É importante entender de forma mais consistente qual a proporção entre as soluções candidatas que são descartadas e as que são efetivamente selecionadas, e se existe alguma correlação com algum momento específico do processo de otimização, pois o conceito do algoritmo modificado que será proposto no próximo capítulo está ligado ao mecanismo de seleção e o número de soluções candidatas que são descartadas.

O experimento consiste em executar o método DE clássico para uma série de funções

de teste, e para cada execução calcular a proporção de soluções candidatas que foram descartadas e as que foram selecionadas a cada iteração do algoritmo. Para um problema com uma população de N_p indivíduos, a cada iteração um número N_s de soluções candidatas são selecionadas, e as demais $N_d = N_p - N_s$ soluções candidatas são descartadas. Os gráficos para esse experimento mostram o comportamento de N_s e N_d ao longo da execução do algoritmo.

As funções escolhidas para o teste foram as funções Rastringin, Rosembrock, Branin e Michalewicz. Detalhes sobre essas funções e suas aplicações podem ser encontrados em Akojwar e Kshirsagar (2016) e Jamil e Yang (2013). A função Rastringin é um exemplo de problema não-convexo e multimodal, com múltiplos mínimos locais que podem fazer um algoritmo de otimização convergir prematuramente. A função Rosembrock é não convexa e possui seu mínimo global localizado em um longo e estreito vale. As funções Branin e Michalewicz também são exemplos de funções multimodais e não-convexas. As equações são apresentadas a seguir:

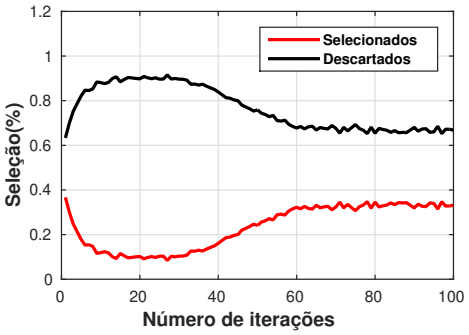
$$Michalewicz : f(x_1,x_2) = -\sin(x_1)\sin^{20}\left(\frac{x_2^2}{\pi}\right) - \sin(x_2)\sin^{20}\left(\frac{2x_2^2}{\pi}\right) \tag{3.7}$$

$$Branin : f(x_1,x_2) = \left(x_2 - \left(\frac{5.1}{4\pi^2}\right)x_1^2 + \frac{5x_1}{\pi - 6}\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10 \tag{3.8}$$

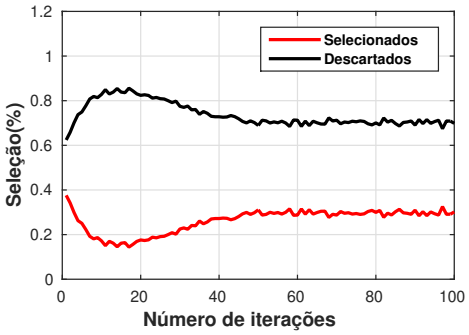
$$Rosembrock : f(x_1,x_2) = (1 - x_1)^2 + 100\left(x_2 - x_1^2\right)^2 \tag{3.9}$$

$$Rastringin : f(x_1,x_2) = 20 + x_1^2 - 10(\cos(2\pi x_1)) + x_2^2 - 10(\cos(2\pi x_2)) \tag{3.10}$$

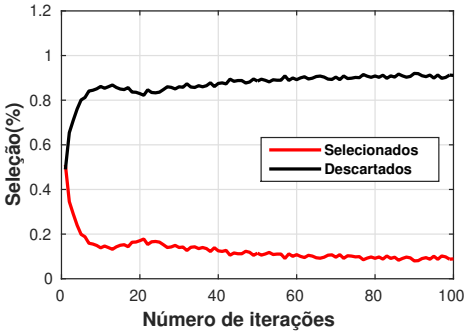
Além das quatro funções de teste apresentadas, o experimento também foi feito para os problemas de Wood & Berry (Wood e Berry, 1973) e da destilaria 4×5 (Muske et al., 1991); em ambos os casos foram usadas as funções objetivo (2.4), (2.5) e (2.6), de acordo com as formulações do capítulo 2. Foram feitas 100 execuções com as funções (3.7), (3.8), (3.9) e (3.10), e 10 execuções para cada função objetivo dos problemas de Wood & Berry e da destilaria 4×5. A figura 3.1 mostra os resultados para as funções Rastringin, Rosenbrock, Michalewicz e Branin. A figura 3.2 mostra os resultados para a destilaria de Wood & Berry e a figura 3.3 para a destilaria de petróleo bruto 4×5. Em todos os casos é mostrado a percentagem de soluções candidatas que foram descartadas e selecionadas, em média, no decorrer do processo de otimização.



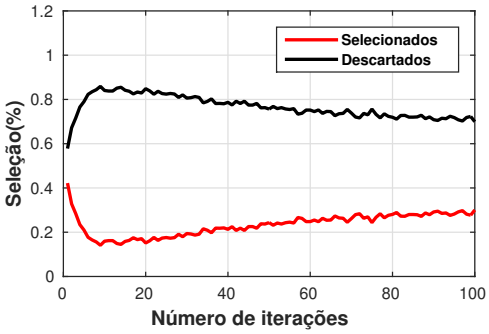
(a) Seleção de indivíduos - Rastringin



(b) Seleção de indivíduos - Rosembrock

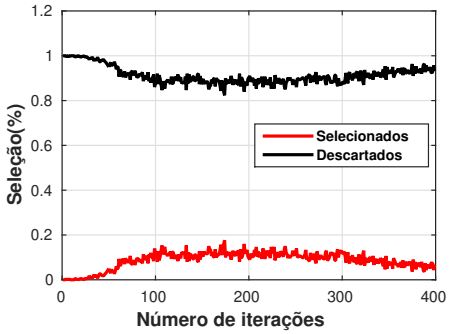


(c) Seleção de indivíduos - Michalewicz

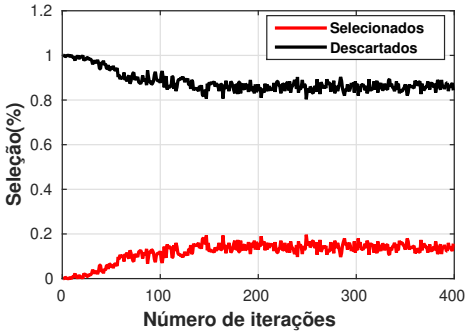


(d) Seleção de indivíduos - Branin

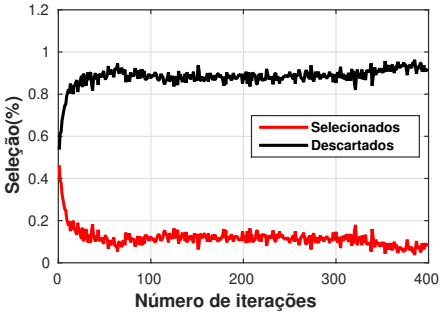
Figura 3.1: Comportamento da seleção para as funções de *benchmark*



(a) Seleção de indivíduos - WB Hinf

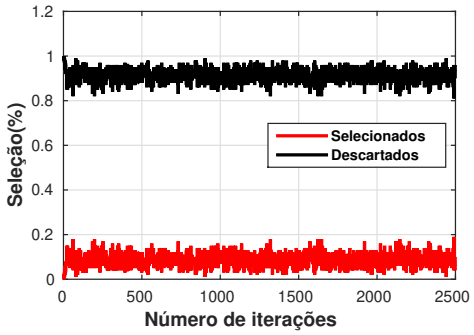


(b) Seleção de indivíduos - WB H2

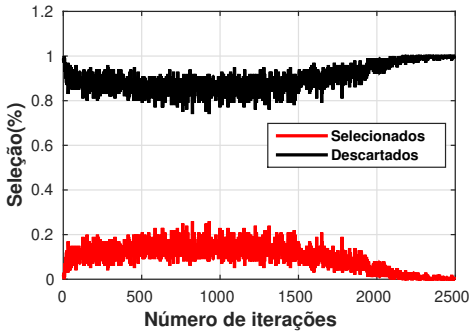


(c) Seleção de indivíduos - WB ISE

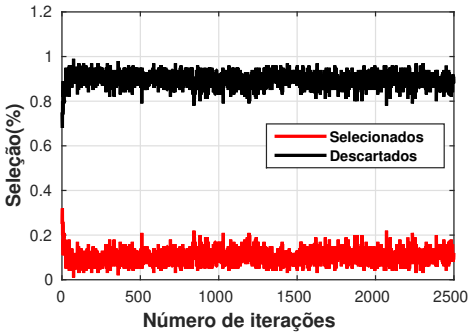
Figura 3.2: Comportamento da seleção para a destilaria 2×2



(a) Seleção de indivíduos - PB Hinf



(b) Seleção de indivíduos - PB H2



(c) Seleção de indivíduos - PB ISE

Figura 3.3: Comportamento da seleção para a destilaria 4×5

A evidência mais importante observada nos gráficos é o alto número de soluções candidatas que são descartadas. Em todos os testes, mesmo sendo funções e problemas diferentes, a porcentagem de soluções candidatas que não são selecionadas se mantém alta durante todo o processo de otimização. Isso significa que a maior parte dos cálculos da função objetivo acaba por ser desperdiçada, uma vez que os indivíduos associados não são selecionados. Dessa forma, a estratégia de tentar descobrir de antemão se a solução candidata vale a pena ser avaliada ou não tem grande potencial para economizar cálculos da função objetivo que não dariam contribuição para o processo de otimização.

Vale ressaltar que a análise apresentada foi feita para um conjunto pequeno de funções de *benchmark* e que possuem apenas duas variáveis de otimização. O mesmo experimento pode ser aplicado de forma ampla a problemas de dimensionalidade elevada, usando-se diferentes algoritmos e operadores evolucionárias.

3.5 Considerações finais

O algoritmo DE é eficiente e de fácil implementação, tendo sido aplicado com sucesso em uma série de projetos na área de controle (vide capítulo 1). Devido a sua alta capacidade de encontrar soluções globais e simplicidade de aplicação, foi escolhido como método de sintonia usado neste trabalho.

O ajuste dos parâmetros do algoritmo é importante para sua aplicação efetiva, tendo sido definidos segundo as regras práticas apresentadas, experiência do projetista e resultados experimentais.

Capítulo 4

Método DE-NN

Neste capítulo introduz-se o método DE-NN (*Differential Evolution-Neural Network*). O método implementa uma rede neural artificial capaz de classificar as soluções candidatas de uma dada população e determinar uma probabilidade estimada dessas soluções resultarem em valores melhores para a função objetivo. Dessa forma, é possível saber se vale a pena ou não calcular a função objetivo para cada solução candidata, poupando tempo sempre que a probabilidade indicar que não vale a pena computar a função custo.

A rede neural pode ser introduzida no processo de otimização a partir de abordagens diferentes, sendo que além dos parâmetros de otimização, é necessário também definir as especificações da rede neural, como o número de soluções usadas no treinamento, número de camadas intermediárias, as quantidades de neurônios de cada camada, o número de classes usado e quais os melhores momentos para treinamento/retreinamento da rede. Essas questões também são discutidas neste capítulo por meio de experimentos práticos.

4.1 Princípio geral

O fator que mais contribui para o custo computacional em um processo de otimização de problemas complexos é o cômputo da função objetivo. Mais especificamente, o número total de vezes em que a função objetivo é calculada. Em problemas de engenharia, a função objetivo envolve o cálculo de equações complexas e, muitas vezes, um processo de simulação ou otimização à parte. Este é o caso da otimização de funções objetivo que utilizam normas de sinais, como o índice ISE, por exemplo. Para calcular a integral do erro ao quadrado da diferença do sinal de referência para a saída simulada do sistema, é preciso simular o sistema em malha fechada com as soluções da população corrente.

Como foi verificado no Capítulo 3 através de experimentos realizados com o DE tradicional, na maior parte das vezes a solução candidata é descartada no processo de seleção após o cálculo da função objetivo. Tendo isso em vista, o algoritmo DE-NN se baseia no princípio de antever se vale a pena ou não fazer o cômputo da função objetivo, fazendo-o

de fato apenas nos casos em que a probabilidade do resultado ser bom for alta. Nos casos em que essa probabilidade for baixa, simplesmente continua-se o processo de evolução da população, descartando-se, na etapa de seleção, as soluções candidatas que não tiveram a função objetivo calculada.

A implementação dessa modificação consiste na introdução de uma nova função ao algoritmo. Esta função é responsável por receber os indivíduos da população candidata como entrada e fazer a classificação desses indivíduos de acordo com um conjunto de classes pré-estabelecido. Para cada classe é associada uma probabilidade na forma de um número entre 0 e 1, em que 0 indica probabilidade zero de o indivíduo resultar em valores melhores para função objetivo, e 1 indica 100%. Em resumo, trata-se de uma função que classifica a "qualidade" dos indivíduos da população candidata.

Essa nova função é implementada por meio de uma rede neural artificial que classifica cada indivíduo em valores pré-definidos de probabilidade estimada. Assim, o algoritmo DE-NN determina se uma solução candidata terá sua função objetivo calculada de acordo com a probabilidade associada a sua classificação, que é determinada pela rede neural. Desse modo, o número total de vezes em que a função objetivo é calculada diminui, impactando na diminuição do tempo de execução do algoritmo.

4.2 Aplicação da rede neural para classificação de indivíduos

O problema de modelagem proposto para a rede neural é um problema de classificação de um vetor de entrada u de dimensão D em uma dentre C classes definidas previamente. No contexto do DE-NN, cada uma dessas classes representa um nível de probabilidade de o indivíduo u estar associado a valores melhores para a função objetivo do problema.

O vetor B_i, que representa cada classe, é um vetor em que o i-ésimo elemento é 1 e os demais elementos são nulos. O que diferencia cada classe é a posição do elemento 1. Essa formulação é baseada no toolbox de redes neurais do $MATLAB^{®}$, que faz a classificação usando vetores em que um elemento é 1 e os demais são 0. Para um problema de classificação em quatro classes, o vetor B_i pode assumir os seguintes formatos:

$$B_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad B_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad B_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.1}$$

Essas classes são definidas com base nos dados disponíveis para o treinamento da rede neural. Ao tentar classificar um indivíduo que não pertence à base de dados original, a rede tentará indicar à qual classe é mais provável que esse indivíduo pertença. Isso é

feito atribuindo valores entre 0 e 1 para cada posição do vetor de saída da rede. Como cada posição indica uma classe diferente, a posição que contiver o valor mais alto indica a classe a qual mais provavelmente o vetor de entrada pertence. Como exemplo prático, os vetores P_1, P_2 e P_3 apresentados a seguir mostram possíveis resultados gerados pela rede neural para um problema envolvendo 4 classes:

$$P_1 = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.15 \\ 0.05 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 0.2 \\ 0.6 \\ 0.1 \\ 0.1 \end{bmatrix} \tag{4.2}$$

Para a classificação P_1, o valor da posição 1 é o mais próximo de 1, indicando que a rede classificou o indivíduo como provavelmente pertencente à classe B_1. Para a classificação P_3, o valor da posição 2 é o mais próximo de 1, indicando que a rede classificou o indivíduo como provavelmente pertencente à classe B_2. P_2 mostra o pior caso possível, em que a rede neural atribui a mesma probabilidade de o indivíduo pertencer a todas as classes, indicando que a rede neural não possui informação suficiente para fazer a classificação adequadamente. Nesse caso, o algoritmo DE-NN considera que a solução pertence à classe associada com a maior probabilidade possível.

Quanto menos precisa for a classificação, menor é a diferença entre os elementos do vetor P (caso de P_3). Quanto mais precisa a classificação, mais o vetor P tende a ter um elemento próximo a 1 e os demais nulos.

4.3 Metodologia para implementação da rede neural no algoritmo DE-NN

Para utilizar a classificação no processo de otimização, cada uma das classes definidas pelo usuário é associada a um valor entre 0 e 1. Essas probabilidades são definidas em um vetor ρ de dimensão igual ao número de classes usado no algoritmo. Cada elemento de ρ corresponde a uma probabilidade associada a uma das classes. Por exemplo, em um problema com 4 classes, define-se $\rho = [\rho_1 \ \rho_2 \ \rho_3 \ \rho_4]$, de forma que:

$$\begin{aligned} B_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \rho_1 = 0.8 \qquad B_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \rho_2 = 0.6 \\ B_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \Leftrightarrow \rho_3 = 0.4 \qquad B_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \Leftrightarrow \rho_4 = 0.2 \end{aligned} \tag{4.3}$$

Ou seja, para cada i-ésima classe, associa-se uma probabilidade ρ_i entre 0 e 1. Essas

probabilidades são parâmetros definidos pelo usuário. A rede neural pode ser treinada a partir de um conjunto inicial de soluções geradas aleatoriamente com esse objetivo ou pode usar as próprias soluções geradas pelo algoritmo DE. A rede pode ser retreinada durante o processo de otimização quando se obtém um novo conjunto de treinamento com a dimensão necessária escolhida pelo projetista. Uma vez que a rede é criada e treinada, ela passa a fazer parte da etapa de seleção. No algoritmo DE clássico a etapa de seleção calcula a função objetivo para toda a população candidata gerada na etapa de cruzamento (vide capítulo 3). No algoritmo modificado, uma vez que a rede neural foi criada, ela é usada antes de calcular a função objetivo para saber se vale a pena ou não fazer o cômputo, sendo o mesmo realizado apenas nos casos em que o indivíduo realmente tem chances de ser selecionado. O mecanismo proposto nesta dissertação usado para tomar a decisão de calcular ou não a função objetivo é descrito a seguir. Seja ρ_i a probabilidade associada à classe a que uma solução candidata pertence e *rand* uma função que gera números aleatórios entre 0 e 1 com distribuição uniforme. Se $rand > \rho_i$, então a solução candidata é descartada sem o cálculo da função objetivo, caso contrário o valor da função objetivo é calculado para comparação com a solução alvo no processo de seleção, conforme descrito na equação (3.6). Esse mecanismo faz parte da modificação proposta para o algoritmo DE e foi pensado de maneira a tornar o processo de decisão probabilístico. Desse modo, ao invés de uma decisão binária entre apenas as opções de calcular função objetivo para soluções melhor classificadas e não calcular para as demais, tem-se um mecanismo em que a probabilidade ρ_i associada a cada classe faz com que sempre exista a possibilidade de avaliação da função objetivo, sendo as chances de avaliação maior para soluções bem classificadas e menor para soluções classificadas como ruins.

4.3.1 Criação e treinamento da rede neural

A criação e treinamento da rede neural consiste em três etapas: criação da massa de dados inicial, preparação da base de dados para treinamento e criação/treinamento da rede neural.

A massa de dados inicial, nos casos em que a rede é criada e treinada *online* (no próprio algoritmo), consiste numa população inicial aleatória. Outra forma de incorporar a rede neural ao algoritmo é criar e treinar a rede *offline* (antes de iniciar o algoritmo) usando-se uma base de dados previamente existente. Essa abordagem, porém, tem o inconveniente de depender da existência de um banco de dados prévio.

O processo de criação e treinamento da rede neural é dado pela rotina descrita no algoritmo 2. Os inscritos na cor azul são comentários no corpo do algoritmo. A rotina recebe como entrada a população para treinamento formada por X_{train} e F_{train}, conjunto de vetores solução e valores da função objetivo correspondentes, respectivamente, o número

de classes utilizado, C, e o número de indivíduos da população de treinamento, MAX_{rn}. Ainda no algoritmo 2, I_p representa a i-ésima coluna da matriz identidade de dimensão C, e a função $sort(X_{train},F_{train})$ ordena os vetores X_{train} e F_{train} em ordem decrescente com base nos valores de F_{train}.

Algorithm 2 Rotina TreinaRede()

1: Inicializa paos parâmetros do algoritmo: X_{train},F_{train},C,MAX_{rn} e N_{neu}
2: $sort(X_{train},F_{train})$ %Ordena X_{train} e F_{train} em ordem decrescente
3: $p \leftarrow 1$ %Índice de classe
4: $i \leftarrow 1$ %Índice da solução
5: $j \leftarrow 1$
6: $I \leftarrow$ `Matriz identidade de ordem C`
7: %Cria vetor de classes para treinar a rede
8: **while** $p \leq C$ **do**
9: **while** $i \leq (-1 + j + MAX_{rn}/C)$ **do**
10: $F_{rn,i} \leftarrow I_p$
11: $i \leftarrow i + 1$
12: **end while**
13: $j \leftarrow j + MAX_{rn}/C$
14: $p \leftarrow p + 1$
15: **end while**
16: %Cria rede neural com função *patternet* com N_{neu} neurônios
17: $rede \leftarrow patternet(N_{neu})$
18: %Treina a rede com vetores X_{rn} (entradas) e F_{rn} (classes associada a cada entrada de X_{rn}) e o número de classes usado
19: $rede \leftarrow train(X_{train},F_{rn},C)$

O que permite que cada classe esteja associada a uma probabilidade maior de obter indivíduos promissores para a otimização é o fato de a base de dados usada para treinamento ser ordenada. Dessa forma, cada classe está associada a uma faixa de valores da função objetivo.

A título de exemplificação, considere um problema com $MAX_{rn} = 2000$ e $C = 4$. Ou seja, tem-se uma população inicial aleatória de 2 mil indivíduos para treinamento da rede, e deseja-se classificar esses indivíduos em 4 grupos, cada qual representando um nível de o quão promissor esse indivíduo é para o processo de otimização. Assim, calcula-se a função objetivo para todos os 2 mil indivíduos e esse resultado é ordenado de forma decrescente. Esses 2 mil resultados são divididos em 4 grupos. O grupo 1 representa a faixa dos 25% melhores valores calculados para a função objetivo. Os grupos 2 e 3 representam as faixas de valores seguintes e o grupo 4 a faixa dos 25% indivíduos com os piores valores para a função objetivo. O que a rede neural faz, ao final, é dizer a qual dessas faixas de valores um dado indivíduo pertence. Como cada faixa está associada a valores maiores ou menores da função objetivo, a classificação indica se o indivíduo tende a resultar em valores melhores,

e que portanto têm alta probabilidade de ser selecionado, ou valores piores, com baixa probabilidade de ser selecionado.

A rede neural para classificação de padrões é do tipo *feed-forward* com duas camadas. A primeira camada é constituída por N_{neu} neurônios que implementam a função *sigmoid*. A segunda camada, que é a camada de saída, implementa a função *softmax*, e possui tantos neurônios quanto o número de saídas da rede. O algoritmo usado para treinar a rede é o *scaled conjugate gradient backpropagation* (**??**, **??**). A configuração da rede com esses parâmetros é padrão do toolbox de redes neurais e se mostrou bem sucedida, embora o projetista possa experimentar outras variações.

Problemas de difícil classificação podem exigir um número maior de neurônios. O número de neurônios e o tamanho dos vetores usados no treinamento impactam no tempo de processamento da rede neural, sendo esse dado importante para o trabalho já que o objetivo é diminuir o custo computacional.

Para que o treinamento da rede seja efetivo, é preciso que o conjunto de dados usado no treinamento seja representativo do problema e que esteja organizado de forma correta. A rotina para criação e treinamento da rede neural é composta de duas seções (algoritmo 2), uma para preparação dos dados e outra para criação e treinamento da rede em si. A preparação dos dados consiste principalmente na adequação das dimensões das matrizes (uma contendo um conjunto de populações e outra com os valores da função objetivo para os elementos dessas populações) e ordenamento das mesmas. O ordenamento é o que permite classificar as populações em uma escala de qualidade, sendo os elementos associados aos melhores valores da função objetivo classificados como de maior qualidade (maior probabilidade de calcular a função objetivo), e os elementos associados aos piores valores classificados como de menor qualidade(menor probabilidade de calcular a função objetivo).

As próximas seções apresentam o método DE-NN em forma de pseudocódigo e uma série de experimentos práticos para definir alguns dos parâmetros de ajuste do algoritmo.

4.4 Algoritmo DE-NN

Das três operações básicas do método DE original, as etapas de mutação e cruzamento permanecem as mesmas no método modificado. A diferença se dá na operação de seleção, que é onde a função objetivo é calculada (foco do custo computacional). O procedimento de seleção modificado para incluir o uso da rede neural é descrito pelo algoritmo 3:

A função $NN(u_i)$ representa a rede neural treinada para classificar e retornar um vetor P em que cada elemento corresponde ao nível de adequação do indivíduo u_i de entrada para cada classe (ver equação (4.2)). O primeiro valor máximo de P determina a qual

Algorithm 3 Etapa de seleção - DE-NN

```
%Usa rede neural para determinar a probabilidade de se calcular a função objetivo
for i = 1 : N_p do
    P ← NN(u_i)
    j ← imax(P)
    ρ_a ← ρ_j

    If rand < ρ_a
      F_u = F_obj(u_i)
        If F_u < F_x(G_i)
          x_{G+1,i} = u_i
        end-If
      n_train ← n_train + 1
      F_train,i ← F_u
      X_train,i ← u_i
    end-If

end for

%Retreina a rede neural - vide algoritmo 2
If n_train ≥ MAX_rn
  NN ← TreinaRede(X_train,F_train,C,MAX_rn)
  F_train ← [ ] %Reinicializa F_train com um conjunto vazio
  X_train ← [ ] %Reinicializa X_train com um conjunto vazio
end-If
```

classe esse indivíduo pertence. Como mencionado anteriormente, para cada classe existe uma probabilidade de cálculo da função objetivo dada por um número entre 0 e 1, em que quanto mais próximo de 1, mais provável é de que o indivíduo u_i tenha a função objetivo calculada. Essa probabilidade é definida no vetor ρ, conforme os exemplos apresentados em (4.3). A função *imax* retorna o índice j do primeiro valor máximo de P. Esse índice é usado para determinar a probabilidade de avaliação ρ_a definida no vetor de probabilidades ρ. Assim, a função objetivo é calculada apenas caso $rand < \rho_a$, do contrário, a solução candidata u_i não é usada na seleção. Sempre que a função objetivo é calculada, a solução candidata e seu valor para a função objetivo são armazenados nos vetores X_{train} e F_{train}, respectivamente.

Na prática, o algoritmo modificado tende a calcular a função objetivo apenas quando a rede neural classificar o indivíduo com um alto valor (uma classe com alta probabilidade de resultar em valores melhores da função objetivo). O uso da função *rand* é uma forma de tornar a decisão probabilística. Ou seja, quanto mais bem classificado for o indivíduo, maior é a probabilidade de que o valor gerado pela função *rand* seja menor que o valor atribuído ao indivíduo, aumentando as chances de cômputo da função objetivo.

À medida que a população evolui, é provável que a rede criada inicialmente já não

tenha a mesma eficácia, já que foi treinada com valores aleatoriamente criados, e a população evolui convergindo para uma região em torno da solução ótima conforme o processo de otimização acontece. Para garantir que a rede continue a classificar de forma eficiente os indivíduos, um mecanismo para reinicialização da rede foi criado conforme as linhas 20, 21 e 22 do algoritmo 3, em que n_{train} é um contador iterado durante a etapa de seleção e MAX_{rn} especifica com quantos novos indivíduos a rede será reinicializada. Desse modo, quando o número de soluções no conjunto X_{train}, F_{train} atinge o tamanho máximo especificado, MAX_{rn}, a rede é treinada novamente e o conjunto de treinamento é reinicializado por meio da atribuição de vetores vazios.

O pseudocódigo dado no algoritmo 4 descreve o método DE-NN como um todo. Em vermelho estão assinalados os trechos que apresentam modificações em relação ao DE clássico.

Algorithm 4 Método DE-NN

1: Inicializa parâmetros do algoritmo: CR,D,N_p,G_{max},C e MAX_{rn}
2: $G \leftarrow 1$
3: %Cria população de treinamento aleatória inicial
4: **for** $i = 1 : MAX_{rn}$ **do**
5: $\quad X_{train,i} = \underline{\lambda} + (\overline{\lambda} - \underline{\lambda}) \times rand(D,1)$
6: **end for**
7: %Calcula função objetivo para cada indivíduo da população
8: **for** $i = 1 : MAX_{rn}$ **do**
9: $\quad F_{train,i} \leftarrow F_{obj}(X_{train,i})$
10: **end for**
11:
12: %Chama algoritmo 2 para criação e treinamento da rede neural artificial
13: $NN = TreinaRede(X_{train},F_{train},C,MAX_{rn})$
14:
15: Inicializa população base do algoritmo com os N melhores indivíduos de X_{train}
16: $k \leftarrow 1$
17: **for** $i = (MAX_{rn} - N) : N$ **do**
18: $\quad Fx_{G,k} \leftarrow F_{train,i}$
19: $\quad x_{G,k} \leftarrow X_{train,i}$
20: $\quad k \leftarrow k + 1$
21: **end for**
22:
23: **while** $G \leq G_{max}$ **do**
24: $\quad$ Operação mutação de acordo com a equação (3.4)
25: $\quad$ Operação cruzamento de acordo com a equação (3.5)
26: $\quad$ Operação de seleção com o algoritmo 3
27: $\quad G \leftarrow G + 1$
28: **end while**

4.5 Testes comparativos

O objetivo principal das modificações propostas é diminuir o número total de vezes em que a função objetivo é calculada, diminuindo assim o tempo total de execução do algoritmo. Para isso, novas funções que também gastam tempo computacional foram acrescentadas. Para que o resultado final seja interessante, o tempo despendido pelas novas funcionalidades do algoritmo não pode ser maior que o tempo economizado com número menor de cálculos da função objetivo.

Os elementos adicionados ao algoritmo que demandam tempo de processamento são a criação da população para treinar rede neural executada no início do algoritmo, o cálculo da rede neural e da probabilidade associada, executado toda iteração, e o retreinamento da rede a cada vez que determinado número de novos indivíduos é alcançado. A economia de tempo se dá pela diminuição do número de vezes que a função objetivo é calculada. Assim, para que haja melhoria do tempo computacional a seguinte condição tem de ser satisfeita:

$$T_{econ} > T_{net} + T_{train} \tag{4.4}$$

Em que T_{econ} é o tempo economizado com a diminuição do número de vezes que a função objetivo é calculada, T_{train} é o tempo para criar e reinicializar a rede neural durante o processo de otimização (incluindo a geração da base de dados inicial) e T_{net} é o tempo total para classificar os indivíduos durante o processo de otimização.

Cada um desses parâmetros é influenciado por variáveis que podem ser ajustadas no algoritmo. T_{train} é tão maior quanto maior for o tamanho da massa de dados usada para treinar e reinicializar a rede neural. T_{net} é tão maior quanto maior for o número de neurônios e de camadas intermediárias da rede neural. O tempo despendido pelas reinicializações da rede neural está incluído em T_{train} e é difícil de analisar, pois quanto menor o número de indivíduos armazenados usados para retreinar a rede, maior é o número de vezes que a rede será retreinada durante o processo de otimização. Por outro lado, quanto maior o número de indivíduos para retreinar a rede, menos vezes o retreinamento é feito, mas cada retreinamento individual é mais demorado.

Além disso, o número de classes e o número de indivíduos usado para treinar a rede neural também podem impactar T_{econ}. Para entender melhor a relação entre os parâmetros de ajuste do algoritmo e o seu impacto nos parâmetros T_{train}, T_{net} e T_{econ} foram feitos alguns estudos práticos que são apresentados a seguir.

4.5.1 Experimento I

O primeiro experimento foi correlacionar o tamanho da base de dados de treinamento e o número de neurônios da camada intermediária com o tempo que a rede neural demora para fazer uma classificação. Os dados usados para o experimento foram obtidos de execuções do algoritmo DE clássico para a síntese de controlador centralizado para o problema de Wood & Berry (Wood e Berry, 1973) e para a destilaria de petróleo 4×5 (Muske et al., 1991). Esses dois problemas foram apresentados no capítulo 2.

As bases de dados usadas são constituídas de 100, 200, 500, 1.000, 5.000 e 10.000 pontos. Cada base de dados consiste em um vetor de entrada e um vetor de saída que são usados para treinar a rede. Para o problema de Wood & Berry o vetor de entrada possui 8 elementos e o vetor de saída 10 elementos (classificação em 10 grupos diferentes). Para o problema da destilaria 4×5 o vetor de entrada possui 40 elementos e o vetor de saída também possui 10 elementos.

Para cada base de dados, a rede neural foi criada/treinada e executada 100 vezes. O experimento foi repetido com 5, 10, 15, 20, 25 e 30 neurônios na camada intermediária da rede neural. A tabela 4.1 mostra os dados para o experimento usando sistema de Wood & Berry e a tabela 4.2 usando o sistema da destilaria 4×5. Em todos os casos μ representa a média e σ o desvio padrão do conjunto de dados obtido.

Tabela 4.1: Custo computacional da rede neural para problema de Wood & Berry

Tempo de execução da rede neural (milisegundos)						
Número de vetores da base de treinamento	1×10^2	2×10^2	5×10^2	1×10^3	5×10^3	1×10^4
Número de neurônios	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
5	0,55 ± 1,5	0,52 ± 1,5	0,49 ± 1,4	0,70 ± 2,0	0,53 ± 1,5	0,49 ± 1,4
10	0,61 ± 1,7	0,58 ± 1,7	0,63 ± 1,8	0,59 ± 1,7	0,56 ± 1,6	0,56 ± 1,6
15	0,71 ± 2	0,70 ± 2,0	0,70 ± 2,0	0,68 ± 1,9	0,65 ± 1,8	0,64 ± 1,8
20	0,73 ± 2,1	0,77 ± 2,2	0,78 ± 2,2	1,1 ± 3,1	0,82 ± 2,3	0,71 ± 2,0
25	0,79 ± 2,3	0,81 ± 2,3	0,84 ± 2,4	0,88 ± 2,5	**1,4 ± 3,8**	0,84 ± 2,4
30	**0,94 ± 2,7**	**0,89 ± 2,5**	**0,96 ± 2,7**	**0,94 ± 2,7**	0,87 ± 2,5	**0,87 ± 2,5**
Tempo de treinamento da rede neural (segundos)						
Número de vetores da base de treinamento	1×10^2	2×10^2	5×10^2	1×10^3	5×10^3	1×10^4
Número de neurônios	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
5	0,54 ± 0,13	0,54 ± 0,07	0,50 ± 0,02	0,51 ± 0,02	0,58 ± 0,02	**0,68 ± 0,03**
10	0,52 ± 0,01	0,53 ± 0,02	0,50 ± 0,02	0,52 ± 0,02	0,60 ± 0,03	**0,72 ± 0,03**
15	0,53 ± 0,02	0.53 ± 0,02	0,51 ± 0,02	0,53 ± 0,08	0,62 ± 0,03	**0,75 ± 0,04**
20	0,53 ± 0,02	0,53 ± 0,03	0,51 ± 0,02	0,52 ± 0,04	0,62 ± 0,03	**0,75 ± 0,04**
25	0,53 ± 0,03	0,50 ± 0,02	0,51 ± 0,02	0,52 ± 0,02	0,64 ± 0,05	**0,77 ± 0,04**
30	0,53 ± 0,02	0,49 ± 0,02	0,51 ± 0,02	0,52 ± 0,02	0,64 ± 0,03	**0,79 ± 0,04**

Tabela 4.2: Custo computacional da rede neural para problema da Destilaria de Petróleo Bruto 4×5

Tempo de execução da rede neural (milisegundos)						
Número de vetores da base de treinamento	1×10^2	2×10^2	5×10^2	1×10^3	5×10^3	1×10^4
Número de neurônios	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
5	0,13 ± 0,85	0,14 ± 0,86	0,14 ± 0,89	0,14 ± 0,87	0,19 ± 1,2	0,14 ± 0,90
10	0,22 ± 1,4	0,15 ± 0,96	0,24 ± 1,5	0,15 ± 0,93	0,15 ± 0,93	0,22 ± 1,4
15	0,16 ± 1,0	0,17 ± 1,1	0,26 ± 1,7	0,17 ± 1,1	0,17 ± 1,1	0,16 ± 1,0
20	0,19 ± 1,2	0,26 ± 1,7	0,18 ± 1,1	0,18 ± 1,2	0,18 ± 1,2	0,19 ± 1,2
25	0,20 ± 1,3	0,21 ± 1,3	0,20 ± 1,3	0,21 ± 1,3	**0,32 ± 2,1**	0,20 ± 1,3
30	**0,22 ± 1,4**	**0,22 ± 1,4**	**0,22 ± 1,4**	**0,22 ± 1,4**	0,22 ± 1,4	**0,22 ± 1,4**
Tempo de treinamento da rede neural (segundos)						
Número de vetores da base de treinamento	1×10^2	2×10^2	5×10^2	1×10^3	1×10^4	1×10^4
Número de neurônios	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
5	0,54 ± 0,04	0,54 ± 0,04	0,55 ± 0,04	0,55 ± 0,04	0,71 ± 0,07	**0,86 ± 0,08**
10	0,55 ± 0,04	0,54 ± 0,04	0,55 ± 0,04	0,57 ± 0,04	0,75 ± 0,06	**0,94 ± 0,10**
15	0,52 ± 0,02	0,53 ± 0,03	0,55 ± 0,04	0,58 ± 0,04	0,77 ± 0,06	**0,99 ± 0,11**
20	0,53 ± 0,03	0,54 ± 0,03	0,56 ± 0,04	0,58 ± 0,04	0,82 ± 0,07	**1,07 ± 0,11**
25	0,52 ± 0,02	0,53 ± 0,02	0,56 ± 0,03	0,60 ± 0,04	0,88 ± 0,09	**1,14 ± 0,14**
30	0,53 ± 0,03	0,54 ± 0,03	0,56 ± 0,38	0,62 ± 0,05	0,92 ± 0,11	**1,22 ± 0,18**

Uma questão importante do ponto de vista da implementação computacional é a forma como a rede neural é executada. O $MATLAB^{®}$ permite que o usuário use o próprio objeto computacional criado pela função *patternet* para calcular a qual classe uma dada solução pertence, e também é possível criar uma função dedicada. O objeto computacional criado pelo toolbox armazena uma série de informações na memória além das constantes e dados da rede neural em si, fazendo com que sua utilização seja mais custosa. A função dedicada é a forma ideal de implementação, pois todo o custo computacional é gasto apenas com informações da rede neural em si. O experimento foi todo feito implementando a rede neural como uma função dedicada.

Nota-se que para o tempo de execução da rede neural tem-se um desvio padrão muito elevado para todos os resultados, indicando uma grande dispersão dos resultados. No entanto, é possível ver que aumentando-se o número de neurônios, a tendência é que a média e o desvio padrão fiquem maiores, apontando que o custo computacional tende a crescer ao se usar redes neurais com maior número de neurônios. O fato é condizente com a natureza da execução da rede neural, já que com um maior número de neurônios, maior é o número de cálculos envolvendo os ganhos e funções de ativação associadas a esses neurônios. Já em relação ao aumento da base de dados, a tendência é que os valores se mantenham. Como a estrutura da rede neural é a mesma, aumentar a base de dados

apenas altera os pesos e demais constantes internas, o que não implica em maior ou menor número de cálculos.

Com relação ao tempo de criação e treinamento da rede neural, tanto o tamanho da base de dados como o número de neurônios são importantes. Quanto maior o número de neurônios, maior é o número de constantes que têm de ser ajustadas pelo treinamento. Para a faixa de 5 a 30 neurônios testada é possível ver que o impacto ainda é pequeno, sendo notado principalmente quando a base de dados possui 10 mil pontos. Para esse caso o tempo médio de treinamento aumenta de 0,55 segundos (com 5 neurônios) para 0,62 segundos (com 30 neurônios).

Tanto a tabela 4.1 como 4.2 apresentaram as mesmas tendências. Foram notados alguns resultados fora da curva nos dados referentes ao tempo de execução da rede neural, o que está relacionado ao desvio padrão elevado. Para atestar que essa alta dispersão não é devido ao baixo número de amostras, um pequeno conjunto do experimento foi repetido com mil e dez mil amostras. Em ambos os casos o desvio padrão continua maior que o valor médio. Esta alta dispersão pode estar relacionada à natureza do tempo de execução da rede neural, que depende por sua vez, do sistema operacional, da dinâmica interna do $MATLAB^{®}$ e da precisão de suas funções de medição de tempo.

Essa primeira conclusão indica que aumentando o tamanho da base de dados e o número de neurônios, o tempo T_{train} tende a crescer. Por outro lado, para T_{net} o que importa é o número de neurônios da rede neural. Quanto maior o número de neurônios da camada intermediária da rede, maior é o valor de T_{net}.

Em resumo, o experimento aponta que ao aumentar o tamanho da base de dados e o número de neurônios o custo computacional tende a aumentar. Dessa maneira, deve-se usar uma base de dados e número de neurônios apenas com o tamanho necessário para que a rede neural tenha capacidade de funcionar com grau de precisão razoável. Nos próximos experimentos a relação entre a precisão da rede neural e o número de neurônios e tamanho da base de dados também é analisado.

4.5.2 Experimento II

O segundo experimento tem como objetivo avaliar a influência do número de neurônios da camada intermediária no desempenho da rede neural. Usando-se a mesma base de dados do experimento I, foram sintetizadas redes neurais com a mesma estrutura de camadas (uma camada intermediária e uma de saída), mas variando-se o número de neurônios da camada intermediária. Durante a criação e treinamento da rede, 85% dos pontos da base de dados são usados diretamente no treinamento, e os demais 15% são usados como população de teste. Como o valor da função objetivo é conhecido para todos os valores de entrada da base de dados, a classe a qual cada indivíduo pertence também

é conhecida *a priori*. Assim, é possível validar o resultado da classificação dado pela rede neural usando-se a população de teste como dado de entrada e comparando o resultado com as classificações conhecidas de antemão. A validação é dada em valores percentuais, isto é, para os indivíduos da população de teste, quantos foram classificados corretamente em relação ao total.

A tabela 4.3 apresenta os resultados. Foram criadas redes com 5, 10, 15, 20, 25, 30, 50 e 100 neurônios. Para cada número de neurônios o teste foi aplicado para as funções objetivo (2.4), baseada na norma H_∞, (2.5), baseada na norma H_2, e (2.6), baseada no índice ISE. O experimento foi feito para o sistema de Wood & Berry (Wood e Berry, 1973) e para a Destilaria de Petróleo Bruto 4×5 (Muske et al., 1991).

Tabela 4.3: Desempenho da rede neural em relação ao número de neurônios da camada intermediária

Percentagem de acerto da classificação da rede neural						
número de neurônios	**Destilaria Wood & Berry**			**Destilaria de Petróleo Bruto**		
	H_∞**(%)**	H_2**(%)**	*ISE(%)*	H_∞(%)	H_2**(%)**	*ISE*(%)
5	90.7	94.8	86.3	99.0	98.5	98.3
10	95.6	98.6	89.3	**99.4**	98.6	98.5
15	94.2	98.2	**90.8**	99.0	98.5	99.2
20	95.6	99.7	88.1	97.8	98.9	98.9
25	95.1	99.7	88.9	99.3	98.6	99.0
30	95.9	99.3	86.3	99.0	**99.0**	98.8
50	**96.2**	99.6	87.1	98.5	98.9	**99.2**
100	95.9	**99.9**	87.6	99.3	98.6	98.1

A tabela indica que em geral o índice de acerto da rede neural não fica abaixo de 86,3% para todos os casos, sendo que dos 48 testes, apenas 11 resultaram em valores menores que 95%. Para o problema de Wood & Berry, a função custo H_∞ apresentou diferença significativa apenas quando o número de neurônios aumentou de 5 para 10. Com 5 neurônios o índice de acerto é 90,7%, com 10 neurônios o índice aumenta para 95,6%, se mantendo praticamente estável até o número mais alto testado de 100 neurônios.

Esse padrão se repetiu para as funções objetivo H_2 e ISE, e também no problema da destilaria de Petróleo Bruto 4×5. Ou seja, redes neurais com camada intermediária de 5 neurônios apresentam um índice de acerto inferior às redes neurais com 10 neurônios. Aumentando-se de 10 até 100 neurônios, as variações são apenas marginais, não havendo grandes diferenças. Nesse sentido, sabendo que o número de neurônios impacta no tempo de execução da rede neural, parâmetro T_{net} da equação (4.4), o experimento aponta que o número de 10 neurônios é o mais interessante de ser usado para os problemas propostos.

Uma ressalva em relação ao experimento é que a população usada como teste é uma amostra da base de dados usada no treinamento da rede. Esse cenário difere das situações reais em que a rede neural pode receber como entrada dados totalmente diferentes da po-

pulação usada no seu treinamento. É importante ter em mente que o experimento, embora faça um apontamento importante em relação à escolha inicial do número de neurônios, não exime o projetista de eventualmente ter de analisar e aumentar o número de neurônios em situações nas quais a complexidade do problema assim o exija.

Os próximos experimentos continuam a testar a percentagem de acerto da rede neural, mas já no contexto do algoritmo DE-NN aplicado a diferentes problemas, e levando-se em conta também a solução final e a quantidade de cálculos da função objetivo economizados.

4.5.3 Experimento III

O terceiro experimento tem como objetivo validar o funcionamento prático do algoritmo modificado do ponto de vista da otimização. Para tal foram escolhidas quatro funções de *benchmark* da área de otimização para serem usadas com o algoritmo DE-NN. As funções escolhidas foram apresentados no capítulo 3 na seção de *Estudo do mecanismo de seleção no Algoritmo DE*. Tratam-se das equações (3.7), (3.8), (3.9) e (3.10). Foram executadas 2 mil rodadas de otimização e compilados o número de cálculos da função objetivo que foram economizados (em relação ao número de cálculos da função objetivo no DE clássico), e a percentagem de acerto do mecanismo de decisão do algoritmo DE-NN, isto é, de todas as vezes em que o mecanismo de seleção modificado economizou um cálculo da função objetivo, quantas vezes o vetor candidato era realmente pior (resultava em um valor maior da função objetivo que não seria aproveitado pelo processo de otimização) que o da população anterior.

Para computar esses dados, o algoritmo foi propositalmente alterado para que a função objetivo seja calculada em todos os casos, mesmo quando a rede neural indica que não vale a pena. Dessa forma é possível sempre comparar o valor da função objetivo para os indivíduos das populações anterior e candidata, calculando quantas vezes o algoritmo acertou ou errou. No entanto, com essa alteração não é possível avaliar o impacto no tempo de execução. O custo computacional foi medido pelo número de cálculos da função objetivo que teriam sido economizados, em média, pelo mecanismo de seleção modificado. Vale ressaltar que essa classe de problemas de otimização mais simples não é o foco do uso do DE-NN, uma vez que o tempo computacional economizado com a redução do número de cálculos da função objetivo não é maior que o tempo gasto para treinar a rede e classificar as soluções.

Os parâmetros do algoritmo foram os mesmos para todas as funções, sendo população de 20 indivíduos, número máximo de iterações de 100 e classificação das soluções candidatas em 10 classes diferentes. A rede neural foi criada com 200 pontos, e também é retreinada a cada 200 novos pontos armazenados. Para função Rastringin foram usadas 5 classes, pois o resultado com 10 classes não apresentou, em geral, solução final dentro

do esperado.

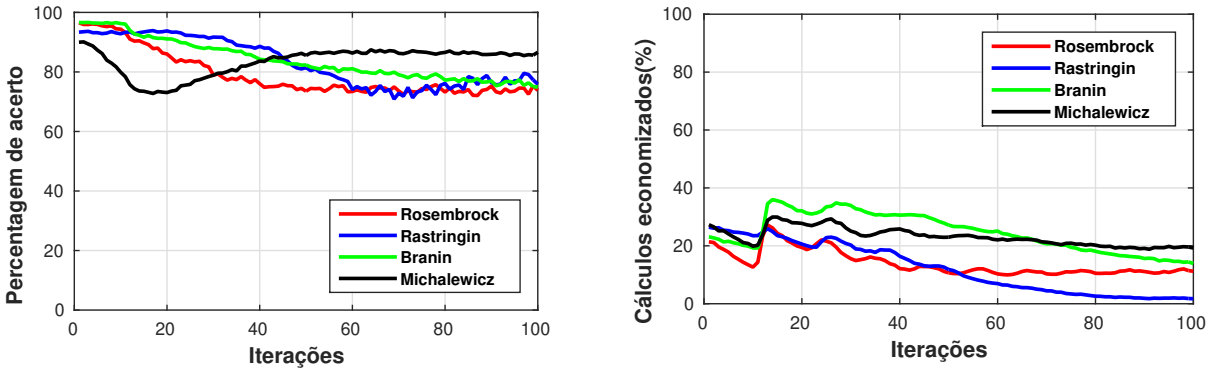
O objetivo principal é saber se a percentagem de cálculos economizados da função objetivo é coerente e realmente corresponde a indivíduos que não contribuíram para a evolução da população rumo ao mínimo global da função custo. Além disso, a solução final é comparada com o valor correspondente utilizando-se o DE clássico. Os resultados são apresentados na tabela 4.4.

Tabela 4.4: Validação prévia do algoritmo modificado

Algortimo modificado aplicado à funções de otimização				
Função custo	**Número de cálculos economizados (%)**	**Percentagem de acerto**	**Solução Final DE-NN**	**Soluçaõ Final DE clássico**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	14,00 ± 6,04	81,78 ± 5,77	$7{,}15 \times 10^{-6} \pm 1{,}45 \times 10^{-4}$	$3{,}03 \times 10^{-6} \pm 8{,}88 \times 10^{-5}$
Rastringin	12,28 ±4,48	89,34 ± 4,87	$4{,}50 \times 10^{-3} \pm 66{,}6 \times 10^{-3}$	$5{,}00 \times 10^{-3} \pm 70{,}2 \times 10^{-3}$
Branin	24,60 ±10,97	84,50 ± 6,00	$0{,}3979 \pm 2{,}45 \times 10^{-6}$	$0{,}3979 \pm 6{,}68 \times 10^{-7}$
Michalewicz	23,12 ± 9,60	81,86± 6,04	$-1{,}001 \pm 1{,}39 \times 10^{-7}$	$-1{,}0008 \pm 7{,}04 \times 10^{-4}$

O número de cálculos economizados para cada função é bastante diferente, e também nota-se um desvio padrão relativamente alto em todos os casos. Isso indica que a capacidade do algoritmo modificado em economizar cálculos da função objetivo depende da natureza da função em questão. Além disso, o alto desvio padrão mostra que o algoritmo modificado possui um baixo grau de repetibilidade em relação ao número de cálculos economizados para a função objetivo. Isso está relacionado à natureza estocástica do algoritmo (população inicial aleatória sempre diferente para cada execução, regras de cruzamento e seleção modificadas são dependentes de parâmetros gerados aleatoriamente).

A figura 4.1 mostra a evolução da percentagem de acerto e da economia computacional do algoritmo DE-NN no decorrer do processo de otimização.



(a) Percentagem de acerto do DE-NN

(b) Cálculos da função objetivo economizados

Figura 4.1: Economia computacional e percentagem de acerto do mecanismo de seleção modificado do DE-NN

A percentagem de acerto varia entre quase 100%, no início do processo de otimização, até pouco mais de 70% ao final. O valor médio total fica acima de 80% para todas as 4 funções, conforme a tabela 4.4. A figura 4.1 mostra que esse índice é maior no início do processo e tende a cair nas etapas finais para as funções Rosembrock, Rastringin e Branin. Para a função Michalewicz a percentagem de acerto tem uma queda no início, se estabilizando e se mantendo constante em seguida até as iterações finais. A economia computacional varia de 20% a 40%, no início do processo de otimização, e entre 0% e 20% ao final. À medida que o algoritmo converge para a região próxima ao mínimo global da função, a rede neural passa a classificar as soluções candidatas com probabilidades cada vez mais altas, aumentando o número de cálculos da função objetivo e, consequentemente, diminuindo a economia computacional na etapa final.

Os resultados apontam que o algoritmo funciona como esperado no sentido de economizar cálculos da função objetivo que, em sua maioria, não teriam utilidade para o processo de otimização, como indicado pelos altos índices de acerto. Para que o algoritmo possa ter bons resultados, é preciso que o número de cálculos da função objetivo economizado seja maior do que os 200 cálculos feitos inicialmente para criar a rede neural (que corresponde a 10% do total de cálculos de um DE clássico). Em todos os casos isso ocorreu. Não obstante, vê-se também um comportamento indesejado, que é o desvio padrão relativamente elevado em relação à economia e percentagem de acerto.

Para as quatro funções testadas, o valor médio da solução final é similar para os dois algoritmos, sendo observadas algumas diferenças principalmente no desvio padrão. A variância do conjunto de dados relativo à solução final é alta tanto para o DE clássico como para o DE-NN, sendo que em alguns casos, como para a função Rastringin, chega a ser da mesma magnitude do valor médio encontrado. Isso é reflexo da natureza estocástica do algoritmo e, possivelmente, de situações específicas em que os algoritmos não convergiram para valores satisfatórios da função objetivo, puxando a variância do conjunto para cima.

Em geral o algoritmo DE-NN possui resultados similares ao DE clássico, ainda que com um número menor de cálculos da função objetivo, indicando que o mecanismo de seleção modificado descarta, majoritariamente, soluções candidatas que não seriam incorporadas ao processo de evolução.

Um exercício similar ao anterior, usando as mesmas funções matemáticas, foi correlacionar o tamanho da base de dados usada para treinar a rede neural com a percentagem de acerto do algoritmo modificado Os resultados estão compilados nas tabelas 4.5, 4.6, 4.7 e 4.8:

Para todas as tabelas, observa-se que aumentando-se o tamanho da base de dados, o número de cálculos economizados tende a ser menor. Uma base de dados maior significa que a rede neural será reinicializada poucas vezes durante a execução do algoritmo, e isso

Tabela 4.5: Relação da base de dados de treinamento com percentagem de acerto da rede neural - Rastringin

Função de Rastringin			
	Número de cálculos economizados(%)	**Percentagem de acerto**	**Solução Final DE-NN**
Número de vetores da base de dados	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
100	**28,62 ± 4,74**	85,22 ± 3,69	0,01± 0,09
200	20,59 ± 7,12	**85,97 ± 4,74**	0,02 ± 0,14
400	18,32 ±8,21	82,39 ± 6,66	$6{,}66 \times 10^{-8} \pm 5{,}15 \times^{-7}$
500	16,48 ± 7,88	83,60 ± 6,69	$6{,}98 \times 10^{-7} \pm 6{,}90 \times 10^{-6}$
1000	12,67 ± 6,98	80,66 ± 9,34	$\mathbf{2{,}08 \times 10^{-10} \pm 1{,}96 \times 10^{-+9}}$

Tabela 4.6: Relação da base de dados de treinamento com percentagem de acerto da rede neural - Rosembrock

Função de Rosembrock			
	Número de cálculos economizados (%)	**Percentagem de acerto**	**Solução Final DE-NN**
Número de vetores da base de dados	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
100	**24,81 ± 5,23**	80,29 ± 3,56	$4{,}93 \times 10^{-8} \pm 3{,}49 \times 10^{-7}$
200	14,22 ± 6,99	**82,92 ± 5,11**	$5{,}45 \times 10^{-9} \pm 5{,}15 \times 10^{-8}$
400	8,61 ± 5,21	80,18 ±7,63	$1{,}06 \times 10^{-10} \pm 1{,}05 \times 10^{-9}$
500	8,27 ± 5,17	79,22 ± 8,24	$8{,}10 \times 10^{-11} \pm 5{,}19 \times 10^{-10}$
1000	7,93 ± 6,30	78,67± 10,87	$\mathbf{6{,}82 \times 10^{-14} \pm 2{,}09 \times 10^{-13}}$

Tabela 4.7: Relação da base de dados de treinamento com percentagem de acerto da rede neural - Michalewicz

Função de Michalewicz			
	Número de cálculos economizados	**Percentagem de acerto**	**Solução Final DE-NN**
Número de vetores da base de dados	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
100	**31,69 ± 6,69**	82,05 ± 4,31	$-1{,}00 \pm 4{,}18 \times 10^{-7}$
200	23,72 ± 9,57	82,42 ± 5,81	$-1{,}00 \pm 2{,}13 \times 10^{-7}$
400	16,86 ± 9,43	82,02 ± 8,06	$-1{,}00 \pm 2{,}07 \times 10^{-8}$
500	17,48 ± 8,44	**83,43 ± 7,40**	$\mathbf{-1{,}00 \pm 1{,}64 \times 10^{-8}}$
1000	14,00 ± 8,60	81,09 ± 9,18	$-1{,}00 \pm 2{,}98 \times 10^{-8}$

impacta negativamente no número de cálculos economizados. Ao não atualizar a rede neural com frequência maior, a população do algoritmo evolui para regiões diferentes daquela na qual a rede foi treinada. Pelo indicado nas tabelas desse experimento, isso contribui para diminuição do número de cálculos economizados. A tendência da percentagem de acerto variou com cada problema. Para a tabela 4.8 ela aumentou, mas para os demais a tendência geral é de queda da percentagem de acerto à medida que a base de dados

Tabela 4.8: Relação da base de dados de treinamento com percentagem de acerto da rede neural - Branin

Função de Branin			
	Número de cálculos economizados(%)	**Percentagem de acerto**	**Solução Final DE-NN**
Número de vetores da base de dados	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
100	**30,64 ± 7,40**	82,58 ± 4,40	$0{,}3979 \pm 8{,}36 \times 10^{-7}$
200	23,15 ± 10,77	83,93 ± 5,77	$0{,}3979 \pm 4{,}13 \times 10^{-7}$
400	22,50 ± 9,60	84,24 ± 7,65	$0{,}3979 \pm 4{,}98 \times 10^{-8}$
500	19,37 ± 8,22	85,24 ± 8,14	$0{,}3979 \pm 3{,}15 \times 10^{-8}$
1000	12,77 ± 5,97	**88,66 ± 10,30**	$\mathbf{0{,}3979 \pm 1{,}34 \times 10^{-8}}$

cresce.

Outra observação é em relação à solução final. À medida que a base de dados aumenta, a solução final tende a ser melhor, o que está relacionado principalmente com o número maior de cálculos da função objetivo. Esse fato é mais pronunciado nas tabelas 4.5 e 4.6. Nas tabelas 4.7 e 4.8 a diferença praticamente não aparece, sendo observada apenas na diminuição do desvio padrão à medida que o tamanho da base de dados aumenta.

O experimento III mostra a necessidade de uma solução de compromisso entre o tamanho da base de dados usada para criar e reinicializar a rede neural e a economia obtida com a diminuição dos cálculos da função objetivo. Caso a economia no custo computacional esteja abaixo do esperado, pode-se diminuir o tamanho da base de dados para deixar o algoritmo mais rápido. Caso a economia no custo computacional esteja alta, mas a solução final esteja aquém do desejado, pode-se aumentar o tamanho da base de dados usada.

4.5.4 Experimento IV

O quarto experimento tem como objetivo estudar como o número de classes afeta o funcionamento e resultado do algoritmo DE-NN. A mesma suíte de funções matemáticas do experimento III foi utilizada. Cada função objetivo foi testada com um tamanho fixo para o conjunto de treinamento da rede neural de 200 pontos. Foram feitas 100 execuções de cada função objetivo, com população de 20 indivíduos, sendo que a cada rodada de 100 execuções o número de classes foi fixado nos seguintes valores: 3, 5, 10, 20 e 50.

Os resultados são apresentados nas tabelas 4.9, 4.10, 4.11 e 4.12. Foram compilados a percentagem de cálculos economizados da função objetivo, percentagem de acerto do mecanismo de seleção e solução final.

Tabela 4.9: Impacto do número de classes no algoritmo DE-NN - Rosembrock

Função Rosembrock			
Número de classes	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**Solução final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
3	3,14 ± 1,66	**91,41 ± 7,86**	$\mathbf{1{,}29 \times 10^{-12} \pm 6{,}30 \times 10^{-12}}$
5	7,22 ± 3,52	85,21 ± 7,68	$1{,}87 \times 10^{-9} \pm 1{,}87 \times 10^{-8}$
10	14,26 ± 5,73	81,04 ± 6,26	$2{,}72 \times 10^{-9} \pm 1{,}93 \times 10^{-8}$
20	22,61 ± 8,12	78,60 ± 4,52	$1{,}87 \times 10^{-9} \pm 7{,}32 \times 10^{-9}$
50	**36,50 ± 10,38**	75,00 ± 4,17	$4{,}50 \times 10^{-3} \pm 44{,}60 \times 10^{-3}$

Tabela 4.10: Impacto do número de classes no algoritmo DE-NN - Rastringin

Função Rastringin			
Número de classes	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**Solução final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
3	5,46 ± 1,90	**92,31 ± 4,91**	$\mathbf{4{,}11 \times 10^{-11} \pm 1{,}53 \times 10^{-9}}$
5	12,19 ± 4,36	89,53 ± 4,81	$1{,}83 \times 10^{-8} \pm 8{,}61 \times 10^{-8}$
10	21,53 ± 7,68	86,09 ± 4,65	$3{,}93 \times 10^{-5} \pm 2{,}88 \times 10^{-4}$
20	31,59 ± 9,01	83,89 ± 4,53	$10{,}20 \times 10^{-3} \pm 99{,}50 \times 10^{-3}$
50	**42,97 ± 9,13**	82,73 ± 4,96	$4{,}6 \times 10^{-3} \pm 23{,}1 \times 10^{-3}$

Os dados das tabelas mostram que ao aumentar o número de classes a tendência é que a economia com cálculos da função objetivo também seja maior. Por outro lado, a percentagem de acerto também diminui, embora se mantenha em patamares elevados, na casa dos 80% ou 90%. A solução final, em geral, também sofre variações no sentido de piorar à medida que o número de classes aumenta. Tal como no experimento III, os resultados apontam para uma solução de compromisso. Caso a economia no custo computacional esteja aquém do desejado, pode-se aumentar o número de classes. Caso

Tabela 4.11: Impacto do número de classes no algoritmo DE-NN - Michalewicz

Função Michalewicz			
Número de classes	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**solução final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
3	8,95 ± 4,15	**87,22 ± 5,83**	**-1,00 ± 4,75 × 10^{-8}**
5	16,23 ± 6,77	84,77 ± 6,13	-1,00 ± 1,13 × 10^{-7}
10	21,46 ± 9,37	81,00 ± 5,86	-1,00 ± 6,89 × 10^{-8}
20	32,36 ± 9,53	79,00 ± 6,98	-1,00 ± 2,14 × 10^{-7}
50	**40,43 ± 8,15**	78,00 ± 7,49	-1,00 ± 2,04 × 10^{-4}

Tabela 4.12: Impacto do número de classes no algoritmo DE-NN - Branin

Função Branin			
Número de classes	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**Solução Final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
3	7,89 ± 4,54	**91,20 ± 6,09**	**0,3979 ± 8,27 × 10^{-9}**
5	14,66 ± 7,61	88,34 ± 5,98	0,3979 ± 1,88 × 10^{-6}
10	23,39 ± 11,21	84,49 ± 5,71	0,3979 ± 8,34 × 10^{-7}
20	32,98 ± 10,95	80,58 ± 5,92	0,3979 ± 1,80 × 10^{-6}
50	**40,12 ± 10,46**	78,66 ± 5,51	0,3979 ± 3,81 × 10^{-6}

a economia no custo computacional esteja no patamar desejado, mas a solução final não esteja convergindo, pode-se diminuir o número de classes.

4.5.5 Experimento V

O quinto experimento trata-se na verdade de uma variação do algoritmo DE-NN. A motivação para o experimento é a seguinte: antes mesmo do processo de evolução da população começar, é necessário calcular a função objetivo várias vezes para criar a população de treinamento da rede neural. Ainda que os melhores indivíduos sejam escolhidos como sendo a população inicial do algoritmo em si (o que já representa uma evolução da população aleatória criada para treinamento), a criação da base de treinamento para a rede neural é uma etapa ineficiente do processo, pois a função objetivo é calculada de forma aleatória e não dentro do mecanismo evolucionário do algoritmo. Em problemas em que a base de dados necessária seja um pouco maior, pode até mesmo ser impeditivo despender parte do tempo de execução apenas criando essa massa de dados.

Dessa maneira, uma pequena variação pode ser feita para não desperdiçar o tempo de criação da massa de dados. Trata-se de um mecanismo de chaveamento, pelo qual o algoritmo começa como um DE clássico, e assim que o conjunto de treinamento alcança o tamanho mínimo especificado, a rede é criada e treinada pela primeira vez, passando a ser realizada a classificação das soluções para cálculo da probabilidade de cômputo da função objetivo. Desse modo, o algoritmo começa como o DE original, e a partir de certo momento passa a rodar com a rede neural classificando as soluções candidatas. Para a realização do experimento, todos os parâmetros, incluindo o critério de parada, são os mesmos que foram usados no experimento III (tabela 4.4) e foram feitas 2 mil execuções.

A tabela 4.13 mostra os resultados obtidos com essa variação.

Tabela 4.13: Avaliação da variante I do algoritmo DE-NN

Algortimo DE-NN - variante I			
Função custo	**Número de cálculos economizados**	**Percentagem de acerto**	**Solução Final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	13,10 ± 5,77	81,48 ± 6,11	$2{,}14 \times 10^{-4} \pm 6{,}4 \times 10^{-3}$
Rastringin	11,51 ± 4,51	89,03 ± 5,12	$6{,}5 \times 10^{-3} \pm 80{,}0 \times 10^{-3}$
Branin	20,68 ± 9,62	83,46 ± 6,09	$0{,}3979 \pm 2{,}69 \times 10^{-4}$
Michalewicz	**30,15 ± 6,94**	**87,63 ± 5,33**	$\mathbf{-1{,}0007 \pm 1{,}3 \times 10^{-3}}$
Algortimo DE-NN			
Função custo	**Número de cálculos economizados (%)**	**Percentagem de acerto**	**Solução Final - DE-NN**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	**14,00 ± 6,04**	**81,78 ± 5,77**	$\mathbf{7{,}15 \times 10^{-6} \pm 1{,}45 \times 10^{-4}}$
Rastringin	**12,28 ± 4,48**	**89,34 ± 4,87**	$\mathbf{4{,}50 \times 10^{-3} \pm 66{,}6 \times 10^{-3}}$
Branin	**24,60 ± 10,97**	**84,50 ± 6,00**	$\mathbf{0{,}3979 \pm 2{,}45 \times 10^{-6}}$
Michalewicz	23,12 ± 9,60	81,86± 6,04	$-1{,}001 \pm 1{,}39 \times 10^{-7}$

Os resultados da tabela 4.13 mostram que a variante I apresentou melhores resultados para a função Michalewicz, tendo o DE-NN apresentado resultados melhores para

as demais funções. Apesar das diferenças observadas, a variante I também atende aos requisitos de economia computacional (pelo menos 10% dos cálculos da função objetivo) e os valores da solução final estão na mesma faixa de grandeza do DE-NN. Para problemas em que a geração da massa de dados inicial tenha custo elevado, a variante I é uma opção viável.

4.5.6 Experimento VI

Esse experimento também é um estudo prático sobre o comportamento do algoritmo fazendo-se uma variação em sua estrutura. A rede neural é reinicializada durante todo o processo de otimização usando-se a base de dados armazenada. Essa base de dados pode ser constituída por todas as soluções candidatas que são calculadas a cada iteração, ou apenas pelas soluções candidatas selecionadas. A tabela 4.14 mostra a comparação entre os resultados do algoritmo DE-NN armazenando todas as soluções candidatas e armazenando apenas as soluções candidatas selecionadas. Os parâmetros do algoritmo DE-NN variante II são os mesmos do experimento III(tabela 4.4) e o número de execuções também é de 2 mil. Os dados do algoritmo que usa todas as soluções candidatas são os mesmos da tabela 4.4, mas são repetidos aqui para facilitar a comparação de resultados.

Nessa variação do DE-NN, o número de vezes que a rede é treinada tende a ser bem menor, pois menos soluções são incorporadas ao conjunto de treinamento. Como o número de soluções que são incorporadas a população do algoritmo é importante para a evolução do mesmo, e tendo em conta ainda que o próprio DE-NN por si só já incorpora menos soluções em média à população corrente do algoritmo devido ao mecanismo estocástico de classificação e seleção, essa variação tem potencial para impactar na solução final e no número de cálculos economizados da função objetivo.

Tabela 4.14: Avaliação da variante II do algoritmo DE-NN

Algortimo DE-NN - variante II			
Função custo	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**Soluçaõ Final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	13,28 ± 6,93	77,29 ± 6,08	$1{,}73 \times 10^{-5} \pm 5{,}56 \times 10^{-4}$
Rastringin	**17,43 ± 8,56**	81,53 ± 6,83	$7{,}60 \times 10^{-3} \pm 84{,}1 \times 10^{-3}$
Branin	18,29 ± 6,91	82,66 ± 8,53	$\mathbf{0{,}3979 \pm 1{,}49 \times 10^{-7}}$
Michalewicz	20,18 ± 7,91	79,37 ± 7,56	$-1{,}001 \pm 2{,}15 \times 10^{-7}$
Algortimo DE-NN			
Função custo	**Número de cálculos economizados (%)**	**Percentagem de acerto**	**Solução Final - DE-NN**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	**14,00 ± 6,04**	**81,78 ± 5,77**	$\mathbf{7{,}15 \times 10^{-6} \pm 1{,}45 \times 10^{-4}}$
Rastringin	12,28 ±4,48	**89,34 ± 4,87**	$\mathbf{4{,}50 \times 10^{-3} \pm 66{,}6 \times 10^{-3}}$
Branin	**24,60 ± 10,97**	**84,50 ± 6,00**	$0{,}3979 \pm 2{,}45 \times 10^{-6}$
Michalewicz	**23,12 ± 9,60**	**81,86 ± 6,04**	$\mathbf{-1{,}001 \pm 1{,}39 \times 10^{-7}}$

Do ponto de vista dos resultados para a solução final, vê-se que a utilização de apenas indivíduos selecionados para reinicialização da rede neural não apresentou grandes mudanças ou não valeu a pena para as funções Rastringin, Branin e Michalewicz. Para a função Rosembrock a variante II apresentou resultado pior em uma casa decimal. A economia nos cálculos da função objetivo diminuiu para as funções Rosembrock, Branin e Michalewicz e aumentou para a função Rastringin.

De forma geral, o DE-NN variante II apresentou economia computacional menor e percentagem de acerto da rede neural também menor. A solução final foi melhor ou pior dependendo da função objetivo. Embora para a função Rastringin a variante II tenha melhorado a economia computacional, os demais resultados indicam que não há uma melhora significativa. Tendo em vista ainda o efeito que essa variante tem no número de soluções candidatas armazenadas, a versão original do DE-NN ou a variante I são mais indicadas.

4.5.7 Experimento VII

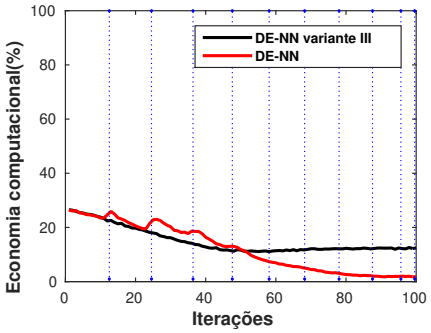
Uma terceira variação possível do algoritmo DE-NN é executar o algoritmo treinando a rede neural apenas uma vez, isto é, após a rede neural ser criada, ela é usada para classificar as soluções durante toda a execução do algoritmo sem ser reinicializada. A tabela 4.15 mostra os resultados obtidos com essa variação em relação ao DE-NN. São apresentados os valores médio e desvio padrão de um conjunto de 2 mil execuções. Os parâmetros dos algoritmos são os mesmos do experimento III(tabela 4.4), exceto que na variante III a rede neural não é reinicializada. Os dados do algoritmo DE-NN são novamente repetidos para facilitar a comparação.

Tabela 4.15: Avaliação da variante III do algoritmo DE-NN

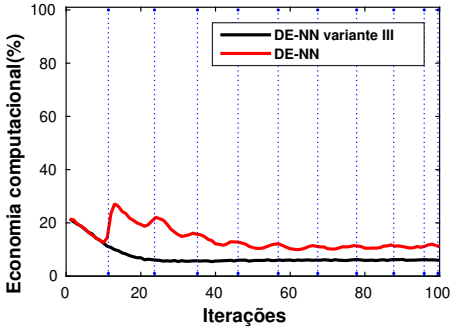
Algortimo DE-NN - variante III			
Função custo	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**Soluçaõ Final**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	7.32 ± 6,87	**87,62 ± 10,78**	$\mathbf{2{,}25 \times 10^{-7} \pm 8{,}41 \times 10^{-6}}$
Rastringin	14,98 ± 10,74	86,82 ± 8,64	$\mathbf{4{,}1 \times 10^{-3} \pm 59{,}9 \times 10^{-3}}$
Branin	9,95 ± 5,99	**94,31 ± 6,79**	$\mathbf{0{,}3979 \pm 1{,}79 \times 10^{-7}}$
Michalewicz	13,92 ± 8,16	**83,75 ± 6,88**	$-1{,}001 \pm 2{,}21 \times 10^{-5}$
Algortimo DE-NN			
Função custo	**Número de cálculos economizados (%)**	**Percentagem de acerto**	**Solução Final - DE-NN**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
Rosembrock	**14,00 ± 6,04**	81,78 ± 5,77	$7{,}15 \times 10^{-6} \pm 1{,}45 \times 10^{-4}$
Rastringin	12,28 ±4,48	**89,34 ± 4,87**	$4{,}50 \times 10^{-3} \pm 66{,}6 \times 10^{-3}$
Branin	**24,60 ± 10,97**	84,50 ± 6,00	$0{,}3979 \pm 2{,}45 \times 10^{-6}$
Michalewicz	**23,12 ± 9,60**	81,86± 6,04	$\mathbf{-1{,}001 \pm 1{,}39 \times 10^{-7}}$

A figura 4.2 mostra a evolução da economia computacional do DE-NN em comparação

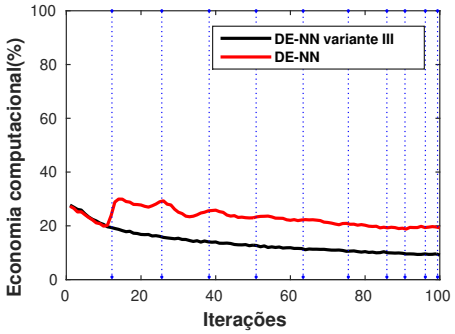
com essa nova variante. A economia computacional está apresentada como a quantidade de soluções candidatas cujo cálculo da função objetivo foi economizado em relação ao número total de cálculos da função objetivo realizado no DE clássico, para cada iteração. As linhas pontilhadas nos gráficos indicam os instantes em que, em média, a rede neural do DE-NN é reinicializada.



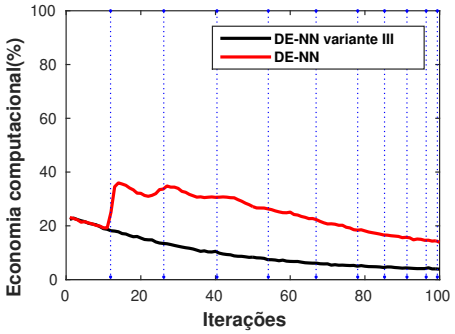
(a) Economia computacional - Rastringin



(b) Economia computacional - Rosembrock



(c) Economia computacional - Michalewicz



(d) Economia computacional- Branin

Figura 4.2: Comparação da economia computacional DE-NN x DE-NN variante III

Pela tabela 4.15 e pelos gráficos da figura 4.2 nota-se que a principal diferença da variante III em relação ao algoritmo DE-NN é a menor economia no custo computacional, que foi observado para as funções Rosembrock, Branin e Michalewicz. Como a rede neural é treinada com uma população inicial aleatória, não sendo mais reinicializada, a tendência é que as soluções sejam classificadas sempre com probabilidade maior à medida que a população evolui. No DE-NN tal como proposto no algoritmo 4, a reinicialização da rede neural é feita exatamente para atualizar a rede neural com informações novas à medida que a população evolui para regiões próximas ao mínimo global da função.

Nos gráficos da figura 4.2 isso é notado com maior clareza ao se observar que antes da primeira reinicialização da rede neural no DE-NN, ambos os algoritmos têm trajetória média praticamente igual em relação à economia no cálculo da função objetivo. A partir do momento em que a rede neural é reinicializada pela primeira vez no DE-NN, o gráfico de cada algoritmo passa a fazer uma trajetória diferente. O DE-NN mantém um patamar mais elevado de diminuição do custo computacional, e o DE-NN variante III diminui e

se estabiliza nas iterações finais em um valor pequeno de economia computacional. É possível notar que nos momentos em que a rede neural é reinicializada, o DE-NN tende a aumentar a economia computacional, mostrando a importância do retreinamento da rede para a manutenção de sua capacidade de descartar previamente soluções insatisfatórias.

A função Rastringin apresentou comportamento contrário; o DE-NN variante III, treinando a rede neural apenas no início, obteve maior economia de custo computacional e convergiu para praticamente o mesmo valor do DE-NN. Isso pode estar relacionado à convergência precoce do DE-NN. Para o DE-NN, uma vez que a região em torno do mínimo global já tenha sido alcançada pela população, a tendência é que o retreinamento da rede neural tenha menor efeito e as soluções candidatas tenham classificações muito próximas, resultando em probabilidades cada vez mais altas de cálculo para a função objetivo. O DE-NN variante III estabiliza o seu percentual de economia computacional nas mesmas iterações em que o DE-NN passa a cair, indicando que os dois algoritmos podem ter convergido para a mesma região, mas com tendência de se estabilizarem em patamares diferentes de economia computacional.

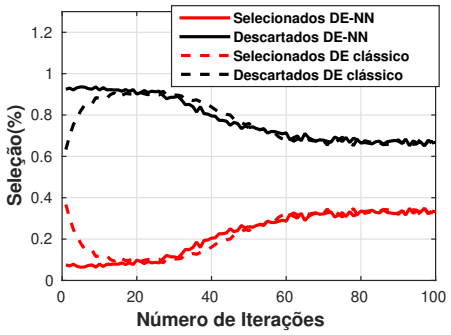
4.5.8 Experimento VIII

Por fim, como último experimento foi feito um estudo sobre o mecanismo de seleção do DE-NN. No capítulo 3 foi mostrado que o algoritmo DE clássico descarta a grande maioria das soluções candidatas durante todo o processo de otimização.

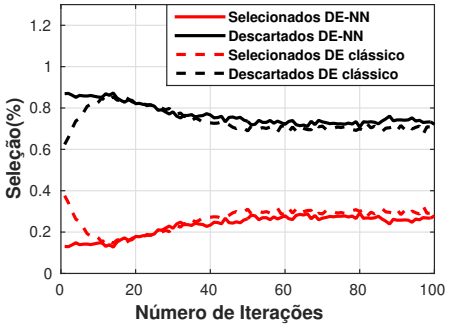
O mesmo teste foi repetido com o DE-NN para avaliar o impacto do mecanismo de classificação na etapa de seleção do método. Para tal, o algoritmo DE-NN foi executado 100 vezes para as funções de *benchmark* utilizadas nos demais experimentos. Foi compilado o número de soluções candidatas que foram descartadas a cada iteração, e o número de soluções candidatas selecionadas e incorporadas à população do algoritmo.

Os resultados constam na figura 4.3. Os gráficos mostram que a tendência do comportamento de seleção não se altera significativamente. O comportamento geral é de que a maior parte das soluções candidatas seja descartada, tanto no DE clássico como no DE-NN. No entanto, há uma diferença no início dos algoritmos. O DE clássico tem uma etapa inicial em que o número de soluções candidatas selecionadas e descartadas começam relativamente próximos, mudando à medida que o algoritmo evolui, e se estabilizando em valores distantes (número de soluções descartadas maior que o de soluções selecionadas). O DE-NN já começa com os dois valores afastados, mantendo o padrão durante todo o processo de evolução.

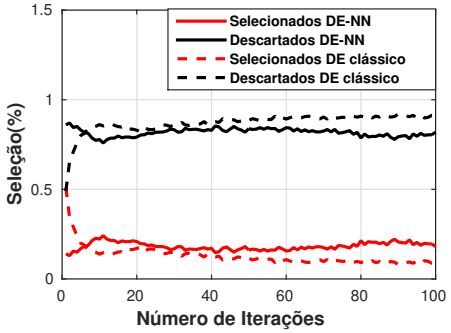
Dessa maneira, vê-se que o comportamento geral do mecanismo de seleção não se altera, sendo a grande diferença para o custo computacional o fato de todas as soluções descartadas no DE original terem um cômputo da função objetivo associado. Já no DE-



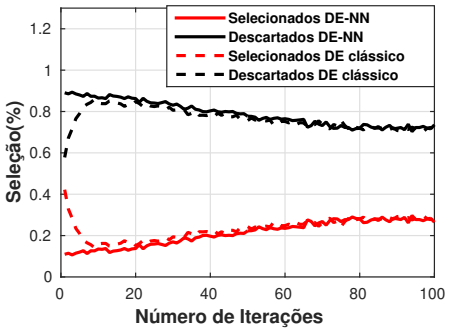
(a) Seleção de indivíduos - Rastringin



(b) Seleção de indivíduos - Rosembrock



(c) Seleção de indivíduos - Michalewicz



(d) Seleção de indivíduos - Branin

Figura 4.3: Comportamento da seleção para as funções de *benchmark* com DE-NN

NN, grande parte das soluções candidatas são descartadas pelo mecanismo de classificação (rede neural), não sendo necessário calcular a função objetivo de fato.

4.6 Considerações finais

O método DE-NN se baseia principalmente no mecanismo de seleção modificado. O desempenho do novo método depende essencialmente do funcionamento da rede neural que classifica os indivíduos da população candidata. Os experimentos práticos mostraram que o conceito do algoritmo funciona com alto índice de acerto (experimento III), sendo sua aplicação em problemas reais de controle bastante promissora.

Para entender melhor o impacto computacional das escolhas referentes à rede neural, foram feitos os experimentos I, II. Para refinar o entendimento do comportamento do algoritmo e sua dependência em relação ao tamanho da base de dados usada para treinar a rede neural e o número de classes usado no processo de classificação, foram feitos os experimentos III e IV. Os experimentos V, VI e VII testaram pequenas variações do algoritmo para avaliar possíveis melhorias, em especial, o experimento VII avaliou a importância da reinicialização da rede neural para a diminuição do custo computacional. Por fim, o experimento VIII avaliou o comportamento do mecanismo de seleção do DE-NN em comparação com o DE clássico, tendo identificado que a proporção de soluções candidatas

descartadas no DE-NN é similar ao observado no DE clássico.

Como resultado final dos experimentos, têm-se os seguintes apontamentos.

- Tempo de execução da rede neural aumenta com número de neurônios da camada intermediária. Tempo de treinamento cresce com aumento da base de dados usada e do número de neurônios.

- O número de 10 neurônios para a camada intermediária tende a ser uma boa escolha, haja visto que números maiores que isso não implicam em aumento significativo na precisão da rede (experimento II).

- A economia do custo computacional tende a aumentar usando-se base de dados menores (reinicializando a rede neural mais vezes) e aumentando o número de classes. Por outro lado, a solução final tende a piorar muito caso esses dois parâmetros sejam levados a extremos, sendo necessário uma solução intermediária.

- Pelo experimento V, para problemas com funções objetivo custosas e que exijam uma massa de dados grande para treinar a rede neural, vale a pena começar o algoritmo como DE clássico e chavear para a versão modificada apenas após a primeira base de dados ter sido armazenada, evitando-se assim a etapa de criação de uma base de dados exclusivamente para treinar a rede neural, que adicionaria custo computacional ao algoritmo (parâmetro T_{train} da equação 4.4).

- Pelo experimento VI, observou-se que usar apenas as soluções selecionadas para o treinamento da rede impacta principalmente na diminuição do custo computacional, tendo, em geral, um efeito negativo.

- O experimento VII atestou a importância da reinicialização da rede neural para a diminuição efetiva do custo computacional. Mostrou também que em casos específicos, como na função Rastringin, as características da função objetivo impactam na economia computacional.

- Pelo experimento VIII, a proporção de soluções candidatas que são descartadas pelo DE-NN no processo de otimização é similar à observada no DE clássico.

Esses direcionamentos serão usados no capítulo 5 nas simulações e resultados práticos para os problemas da área de controle.

Capítulo 5

Simulações e resultados

Todas as simulações foram feitas com processador Intel Core i7 de 1.8GHz com quatro núcleos e 8GB de memória RAM.

5.1 Resultados para sistema 2×2

A tabela 5.1 apresenta os resultados compilados para as baterias de testes com o modelo de Wood & Berry (Wood e Berry, 1973) utilizando-se as funções objetivo (2.4), (2.5) e (2.6). Foram feitas 30 execuções com DE clássico e 30 com DE-NN, cada qual com uma população de 30 indivíduos. O número máximo de iterações para cada caso foi determinado com base nas execuções do DE clássico, tendo sido adotado o número com o qual a maioria das execuções converge para um valor satisfatório. Para as execuções com a função (2.5) esse número é de 350 iterações, para a função (2.4) são 500 iterações e para a função (2.6) 400 iterações. O problema de sintonia do controlador $K(s)$ para esse sistema consiste em um problema com 8 variáveis de otimização, em que os 4 controladores individuais que compõem $K(s)$ serão otimizados (vide capítulo 2).

Foram compilados a integral do erro ao quadrado (ISE) da saída simulada em relação à entrada, o tempo de subida, tempo de acomodação, sobressinal e tempo de execução, sendo mostrados os valores médio e o desvio padrão. A rede neural é treinada com 500 indivíduos quando usada a função objetivo (2.6) e 2 mil indivíduos com as outras duas. Em todos os casos a rede é treinada para classificar as soluções candidatas em 5 classes diferentes.

A tabela 5.2 mostra a diminuição no número de cálculos da função objetivo obtido com o DE-NN. Os dados são apresentados em percentual, tendo como referência o número total de cálculos da função objetivo obtido com algoritmo DE clássico.

Um primeiro dado que chama a atenção na tabela 5.1 é o aumento significativo do desvio padrão em praticamente todos os parâmetros obtidos com o algoritmo DE-NN em relação ao DE clássico. Em relação às diferenças entre os dois algoritmos, ve-se que para

Tabela 5.1: Comparação entre DE clássico e DE-NN para destilaria 2×2

Resultados usando DE clássico					
Funções	**ISE**	**Tempo de subida(min)**	**Tempo de acomodação(min)**	**Sobressinal(%)**	**Tempo de execução(min)**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
2.4 - (H_∞)	**19,77 ± 0,18**	55,54 ± 1,25	138,87 ± 7,13	3,670 ± 0,30	3,55 ± 0,08
2.5 - (H_2)	20,29 ± 0,18	**49,78 ± 1,02**	169,81 ± 6,40	5,50 ± 0,54	**2,50 ± 0,25**
2.6 - (ISE)	19,88 ± 0,03	62,60 ± 0,11	**98,53 ± 0,29**	**1,60 ± 0,03**	14,58 ± 0,17
Resultados usando DE modificado					
2.4 - (H_∞)	**19,44 ± 1,14**	57,09 ± 5,46	151,43 ± 26,98	3.46 ± 1,57	1,99 ± 0,09
2.5 - (H_2)	20,44 ± 1,81	**51,34 ± 12,13**	179,69 ± 56,90	6,45 ± 5,90	**1,4 ± 0,07**
2.6 - (ISE)	19,70 ± 0,81	62,23 ± 4,44	**107,56 ± 21,07**	**1,94 ± 1,20**	10,65 ± 2,11

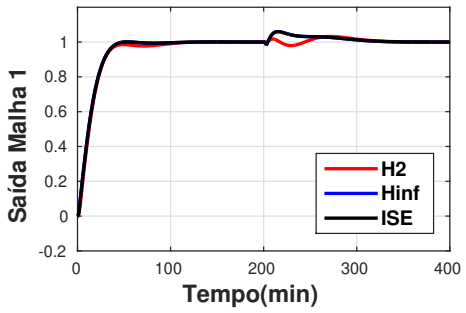
Tabela 5.2: Redução no número de cálculos da função objetivo com DE-NN - Destilaria de Wood & Berry

Funções objetivo	**2.4 - (H_∞)**	**2.5 - (H_2)**	**2.6 - (ISE)**
Redução no número de cálculos da função objetivo (%)	64,97 ±1,84	67,33 ± 1,64	33,99 ± 16,52

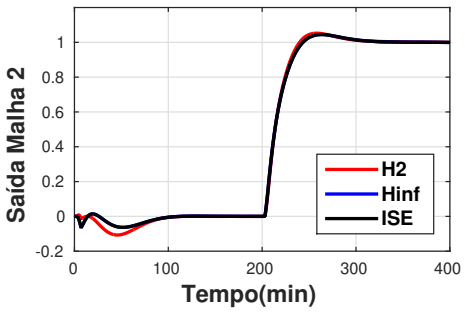
a função objetivo (2.6) os valores médio do índice ISE e tempo de subida se mantiveram próximos aos observados nas execuções com o DE clássico. O tempo de acomodação e sobresinal apresentam diferenças, com os valores obtidos com DE clássico dentro da faixa de variação dos obtidos com DE-NN. O tempo de execução, por sua vez, caiu de 14,58 minutos para 10,65 minutos. Essa diminuição no tempo de execução também se repetiu na mesma proporção para as outras duas funções objetivo. Para as funções (2.5) e (2.4), o tempo de subida e índice ISE são similares com os dois algoritmos. O tempo de acomodação e sobresinal apresentam diferenças maiores.

Comparando-se as diferentes funções objetivo, a função (2.6) apresenta os maiores tempos de subida e de execução, além dos menores tempo de acomodação e sobresinal. O índice ISE é similar para as três funções objetivo. Essas diferenças são observadas tanto nas execuções com algoritmo original como com o DE-NN. Entre as funções (2.5) e (2.4), a primeira apresenta maior tempo de acomodação e sobresinal, porém o tempo de subida e de execução são menores.

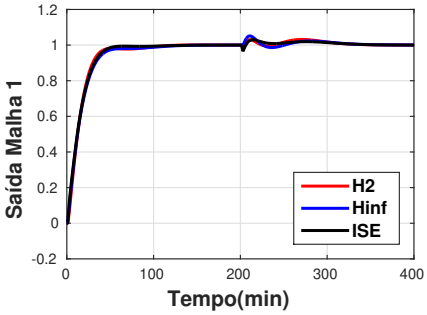
De forma geral, vê-se que o algoritmo DE-NN apresenta repetibilidade menor do que o DE clássico. Os critérios de desempenho de controle, embora com uma variância maior, são similares aos obtidos com o DE clássico e o custo computacional é cerca de 60% menor para as funções (2.4) e (2.5) e 30% menor para a função (2.6).



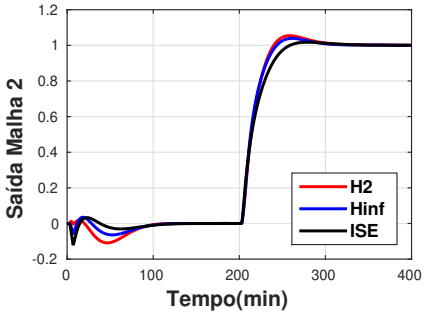
(a) Malha 1 - DE clássico



(b) Malha 2 - DE clássico

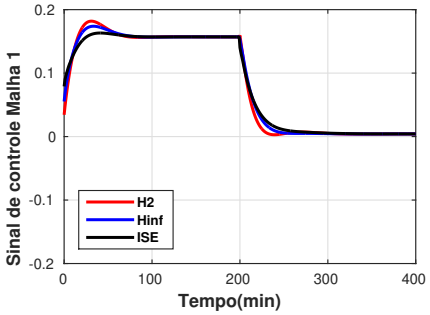


(c) Malha 1 - DE-NN

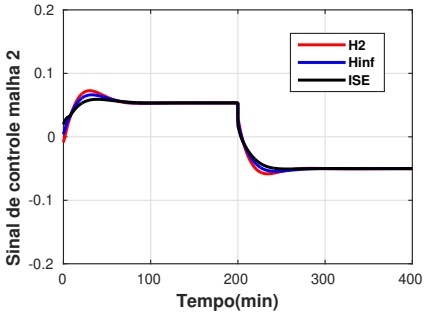


(d) Malha 2 - DE-NN

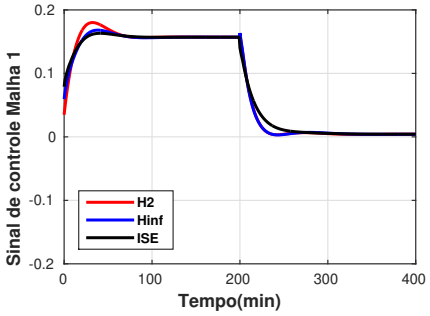
Figura 5.1: Saídas simuladas para o sistema 2×2



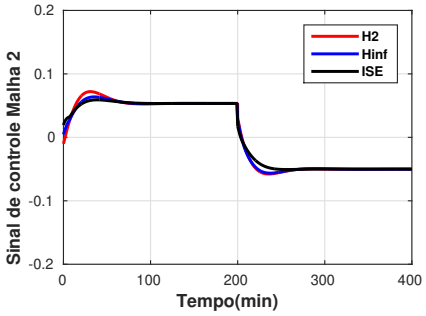
(a) Malha 1 - DE clássixo



(b) Malha 2 - DE clássico



(c) Malha 1 - DE-NN



(d) Malha 2 - DE-NN

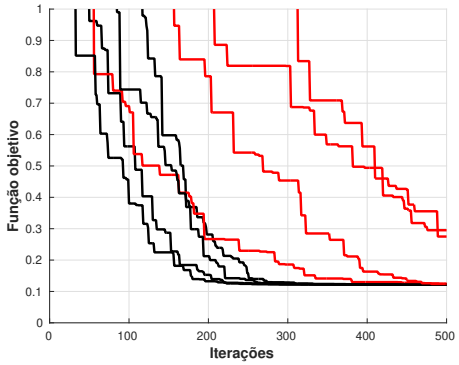
Figura 5.2: Variáveis manipuladas para o sistema 2×2

A figura 5.1 apresenta as saídas simuladas para as duas malhas da destilaria de Wood & Berry quando aplicadas entradas degrau unitário nos instantes 0 e 200 para os sinais

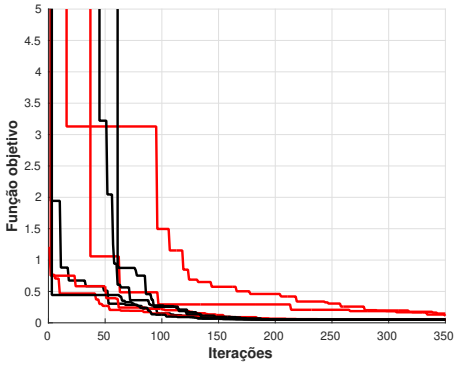
de entrada 1 e 2 respectivamente. A figura 5.2 mostra as variáveis manipuladas.

Para plotagem dos gráficos das figuras 5.1 e 5.2 foram escolhidas as melhores execuções do conjunto apresentado na tabela 5.1, sendo as execuções com menor índice ISE consideradas as melhores.

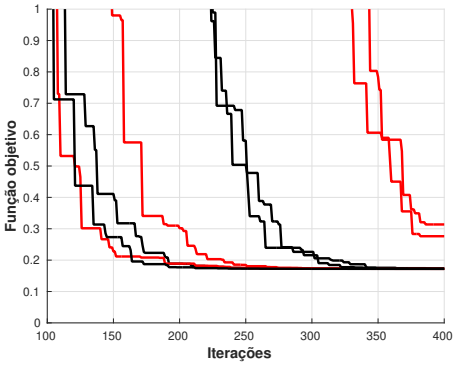
A figura 5.3 apresenta os gráficos de convergência para as simulações com DE clássico e do DE-NN. Para cada gráfico, foram escolhidas as 2 melhores e as 2 piores execuções de cada algoritmo em termos de convergência.



(a) Convergência para função objetivo (2.4) DE-NN(vermelho) e DE clássico(preto)



(b) Convergência para função objetivo (2.5) DE-NN(vermelho) e DE clássico(preto)



(c) Convergência para função objetivo (2.6) DE-NN(vermelho) e DE clássico(preto)

Figura 5.3: Comparação das taxas de convergência para o sistema 2×2

Vê-se que a taxa de convergência do algoritmo DE-NN é menor que do DE clássico. Para as funções objetivo (2.6) e (2.5) as melhores execuções do DE-NN possuem convergência similar ao DE clássico. No entanto, as piores execuções do DE-NN se afastam bastante do comportamento do DE clássico. Para a função (2.4) mesmo as melhores execuções do DE-NN não conseguiram o mesmo nível de convergência do DE clássico. As execuções do DE-NN mostram níveis variados de convergência, um reflexo direto da menor repetibilidade evidenciada pelo maior desvio padrão nos dados da tabela 5.1.

5.1.1 Experimento e análise para o sistema 2×2

Durante as execuções que resultaram na tabela 5.1, foi observado que com as funções objetivo (2.4) e (2.5), o número de indivíduos usados para treinar a rede neural tem impacto muito grande no número de vezes que o algoritmo DE-NN converge e também na solução final. As normas H_2 e H_∞ são calculadas apenas para sistemas estáveis, de forma que para sistemas instáveis, a função objetivo é penalizada por meio da atribuição de um valor muito alto.

No início do processo de otimização grande parte dos sistemas em malha fechada encontrados ainda são instáveis devido ao fato de o algoritmo ter começado com uma população aleatória e ainda não ter tido tempo de convergir para regiões factíveis. Dessa forma, caso a rede neural seja treinada logo no início, a base de dados usada no treinamento será constituída em sua maioria de soluções infactíveis. Com uma base de dados enviesada por possuir majoritariamente informações de soluções infactíveis, a rede tende a classificar a maioria das soluções candidatas como soluções ruins, fazendo com que o número de cálculos da função objetivo seja extremamente baixo. Assim, tem-se um efeito cascata em que, por fazer com que a função objetivo seja calculada muito poucas vezes, a rede neural não consegue formar uma nova base de dados consistente, continuando a descartar a maior parte das soluções candidatas e fazendo o algoritmo terminar a execução sem ter convergido.

Para entender melhor esse comportamento foi repetido o experimento III do capítulo 4, aplicado agora à destilaria de Wood & Berry. Os resultados são apresentados na tabela 5.3. Foram feitas 10 execuções com cada função objetivo, variando-se o tamanho da base de dados usada para treinar a rede neural em 200, 300, 500 e 1000 pontos. Todos os demais parâmetros do algoritmo DE-NN foram mantidos iguais aos usados para gerar a tabela 5.1. As execuções que não convergiram não foram incluídas no cálculo dos valores médio e desvio padrão.

Tabela 5.3: Impacto do tamanho da base de dados para treinamento da rede neural no algoritmo DE-NN - Destilaria de Wood & Berry

Função baseada em norma de sistema (H_2)				
Número de vetores da base de dados	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**solução final**	**Número de execuções que não convergiram**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	
200	52,86 ± 1,47	69,66 ± 4,40	$0{,}05 \pm 1{,}5 \times 10^{-3}$	5
300	60,66 ± 0,20	53,97 ± 28,67	0,53 ± 1,07	5
500	56,00 ± 0,01	68,36 ± 23,00	0,15 ± 0,27	2
1000	52,40 ± 2,26	79,81 ± 0,07	0,05 ± 0,01	0
Função baseada em norma de sistema (H_∞)				
Número de vetores da base de dados	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**solução final**	**Número de execuções que não convergiram**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	
200	51,30 ± 0,63	74,14 ± 3,55	0,21 ± 0,09	6
300	54,67 ± 4,52	69,15 ± 11,85	0,28 ± 0,13	6
500	56,59 ± 0,53	72,17 ± 11,03	0,27 ± 0,14	6
1000	54,38 ± 1,20	82,27 ± 5,42	0,20 ± 0,07	0

As funções objetivo (2.5) e (2.4) apresentam limitações em relação à convergência quando o tamanho da base de dados para treinar a rede não é suficientemente grande. Para o problema em questão, apenas para bases de dados com mais de 1000 pontos todas as 10 execuções convergiram. Em todos os outros casos, mais de uma execução resultou em um sistema em malha fechada instável. Desse modo, para as funções objetivo (2.5) e (2.4) é recomendado usar bases de dados maiores. Para o sistema de Wood & Berry, especificamente, o experimento aponta que pelo menos 1000 pontos são necessários, sendo que os dados da tabela 5.1 foram obtidos com 2000 pontos.

A função (2.6) por sua vez se mostrou mais flexível, funcionando bem com bases de dados de tamanhos diferentes, e sempre convergindo e resultando em sistemas em malha fechada estáveis.

5.2 Resultados para o sistema 4×5

A tabela 5.4 apresenta os resultados para o sistema de destilaria de Petróleo Bruto (Muske et al., 1991) aplicando os algoritmos DE clássico e DE-NN. Foram feitas 10 execuções e calculados os valores médio e desvio padrão. O número máximo de iterações para cada caso foi determinado baseado nas execuções do DE clássico, tendo sido adotado o número com o qual a maioria das execuções converge para um valor satisfatório. Para as execuções com a função (2.5) esse número é de 1800 iterações, para a função (2.4) são 4000 iterações e para a função (2.6) 5000 iterações.

Para esse sistema, a sintonia do controlador PI centralizado dado por (2.1) consiste em um problema com 40 variáveis de otimização, sendo 20 ganhos proporcionais e 20 ganhos integrais (vide capítulo 2).

Tabela 5.4: Comparação entre DE clássico e DE-NN para destilaria 4×5

Resultados usando DE clássico					
Funções	**ISE**	**Tempo de subida(min)**	**Tempo de acomodação(min)**	**Sobressinal(%)**	**Tempo de execução(min)**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$
2.4 - (H_∞)	29,29 ± 1,21	115,34 ± 20,50	275,10 ± 33,49	5,03 ± 1,90	53,77 ± 0,15
2.5 - (H_2)	37,15 ± 0,24	141,31 ± 1,59	329,567 ± 6,72	6,74 ± 0,40	**22,49 ± 0,03**
2.6 - (ISE)	**26,98 ± 0,23**	**93,80 ± 7,87**	**191,87 ± 26,02**	**2,21 ± 1,24**	150,48 ± 3,55
Resultados usando DE modificado					
2.4 - (H_∞)	27,99 ± 2,79	117,01 ± 30,44	282,77 ± 67,09	6,51 ± 4,50	37,34 ± 1,54
2.5 - (H_2)	38,08 ± 2,98	145,42 ± 12,26	346,58 ± 67,14	6,15 ± 2,02	**18,25 ± 1,67**
2.6 - (ISE)	**27,08 ± 0,36**	**91,52 ± 3,13**	**188,21 ± 20,93**	**1,80 ± 1,16**	100,93 ± 7,16

A tabela 5.5 mostra a diminuição no número de cálculos da função objetivo, em percentual, tendo como referência o número total de cálculos da função objetivo com DE clássico.

Tabela 5.5: Redução no número de cálculos da função objetivo com DE-NN - Destilaria 4×5

Funções objetivo	**2.4 - (H_∞)**	**2.5 - (H_2)**	**2.6 - (ISE)**
Redução no número de cálculos da função objetivo (%)	34,34 ± 5,72	41,38 ± 3,32	37,63 ± 7,07

Do ponto de vista das diferenças entre as funções objetivo utilizadas, vê-se que o custo computacional é maior para a função (2.6), o que está ligado ao fato de se tratar de uma norma de sinal, sendo necessário a simulação do sistema em malha fechada para o cômputo da norma em cada avaliação da função objetivo. Já no caso entre as funções (2.4)

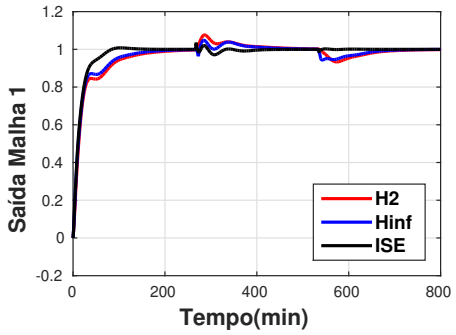
e (2.5), há uma diferença da ordem de 20% apontando para menor número de cálculos da função objetivo com a função (2.4), mas do ponto de vista do tempo de execução, a mesma função apresentou resultados quase duas vezes maiores.

Em relação aos parâmetros de controle, a função (2.6) apresenta o menor ISE, tempo de subida e sobressinal. As outras duas funções apresentam especialmente tempo de acomodação mais elevado. O parâmetro ISE usando-se a função objetivo (2.5) foi cerca de 50% maior do que com as outras duas funções. De maneira geral, a função objetivo que melhor atende os critérios de especificação é a função (2.6), sendo que dentre as outras duas, a função objetivo que utiliza norma H_∞ apresentou melhores resultados. Essas diferenças de resultado entre as funções objetivo são observadas tanto nas execuções com DE-clássico como com DE-NN.

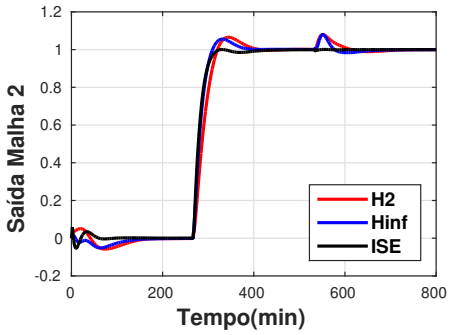
Já do ponto vista da comparação entre os algoritmos DE-clássico e DE-NN, utilizando-se a função objetivo (2.6) os parâmetros de controle são próximos, sendo o sobresinal e principalmente o tempo de execução as grandes diferenças. Para a função objetivo (2.4), foram observadas variações um pouco maiores no tempo de acomodação e sobressinal. Para a função (2.5) os valores obtidos com o DE-NN foram maiores para tempo de subida e acomodação e menor para sobresinal. Nas execuções com DE-NN o desvio padrão é maior para todos os casos, apontando maior dispersão dos resultados. Em todos os casos o tempo de execução com DE-NN foi cerca de 30% menor e a quantidade de cálculos da função objetivo diminuiu, em média, entre 34,34% e 41,38%, conforme a tabela 5.5.

A figura 5.4 mostra a saída simulada usando DE clássico, e a figura 5.5 a saída simulada usando-se o DE-NN, em ambos os casos foram aplicados sinais degrau unitário nos instantes 0, 260 e 530 às entradas de referência 1, 2 e 3, respectivamente. Das quatro saídas desse sistema, a saída 4 deve permanecer em repouso, sendo seu sinal de referência fixado em zero. A figura 5.6 mostra a saída do controlador centralizado (variáveis manipuladas) utilizando-se o DE-clássico e a figura 5.7 mostra as variáveis manipuladas utilizando-se o DE-NN. Tal como para o sistema 2×2, os gráficos apresentados se referem à melhor execução do conjunto de dados obtido, medido pelo índice ISE.

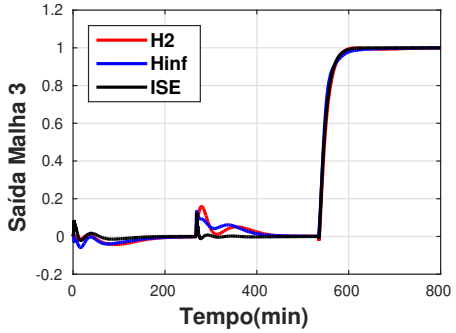
Graficamente é difícil apontar diferenças significativas entre os resultados do DE-clássico e do DE-modificado. Apenas no caso da função objetivo (2.4) é possível notar que o sobressinal na malha de controle referente ao segundo sinal de entrada (degrau aplicado no instante 230) é maior na execução utilizando DE clássico.



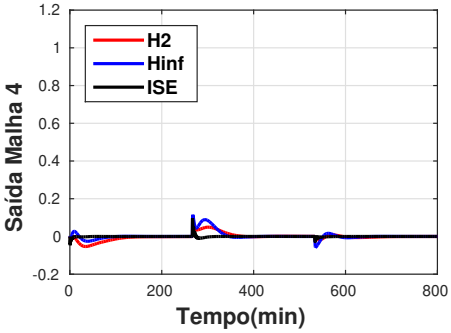
(a) Saídas para Malha 1



(b) Saída para malha 2

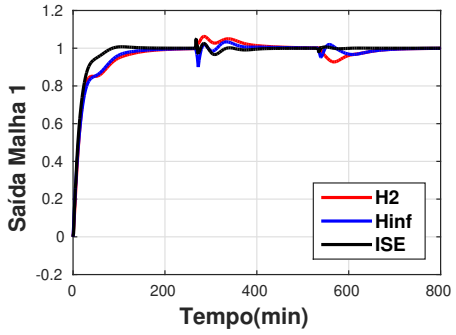


(c) Saídas para Malha 3

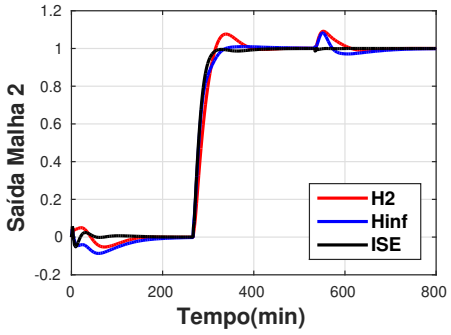


(d) Saída para malha 4

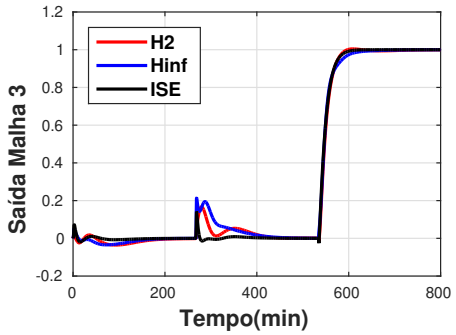
Figura 5.4: Saídas simuladas para o sistema 4×5 usando DE clássico



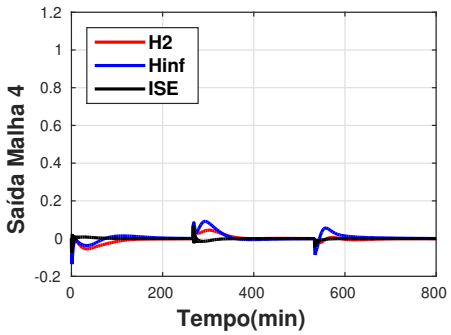
(a) Malha 1



(b) Malha 2

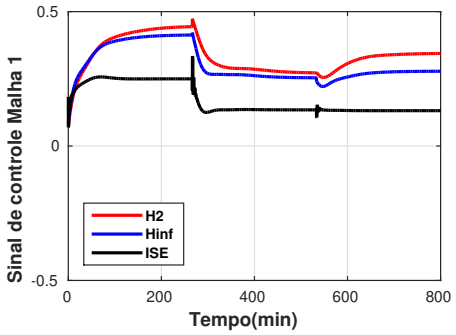


(c) Malha 3

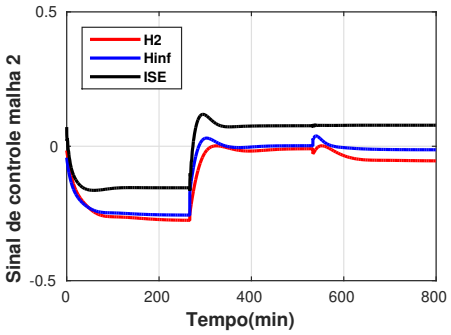


(d) Malha 4

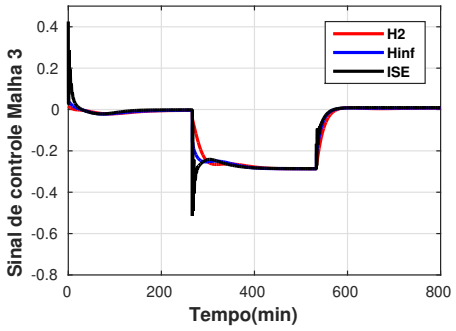
Figura 5.5: Saídas simuladas para o sistema 4×5 usando DE-NN



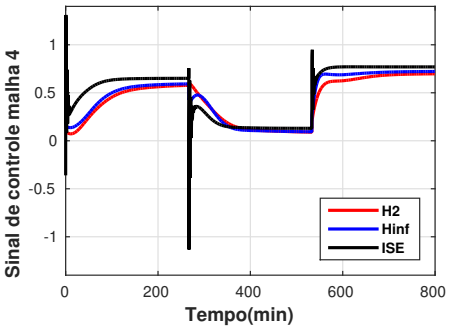
(a) Malha 1



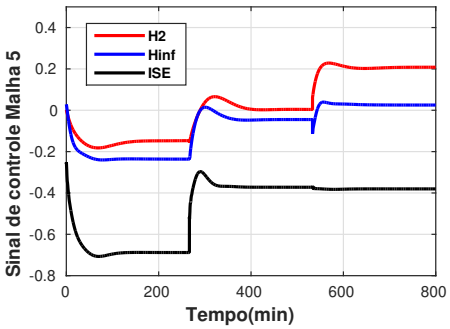
(b) Malha 2



(c) Malha 3

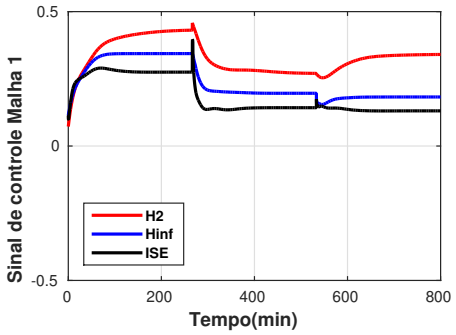


(d) Malha 4

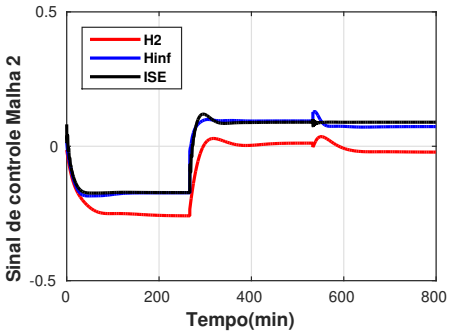


(e) Malha 5

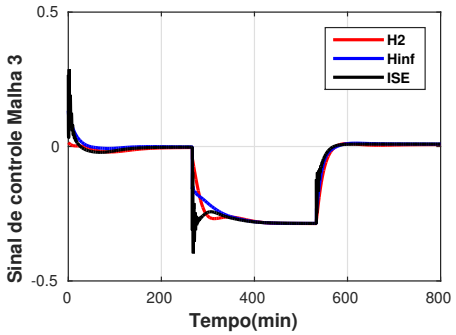
Figura 5.6: Variáveis manipuladas para o sistema 4×5 com DE clássico



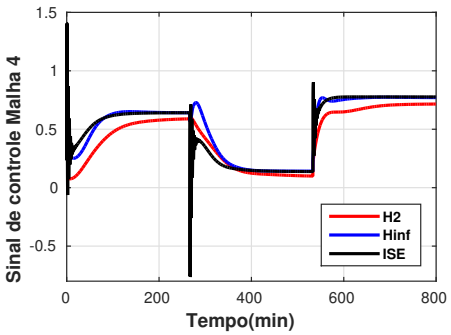
(a) Malha 1



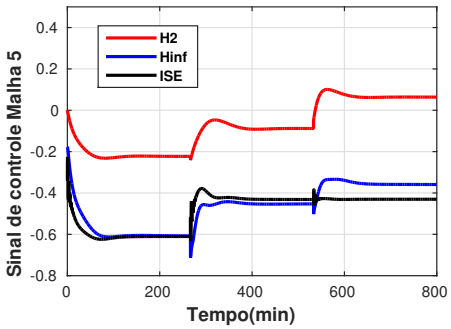
(b) Malha 2



(c) Malha 3



(d) Malha 4

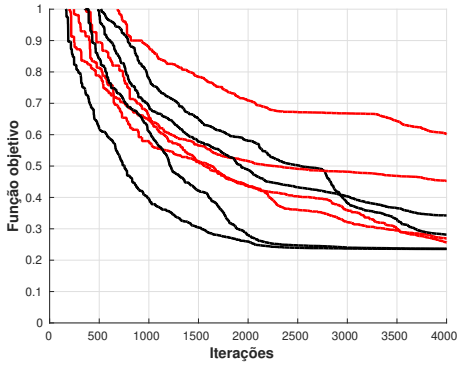


(e) Malha 5

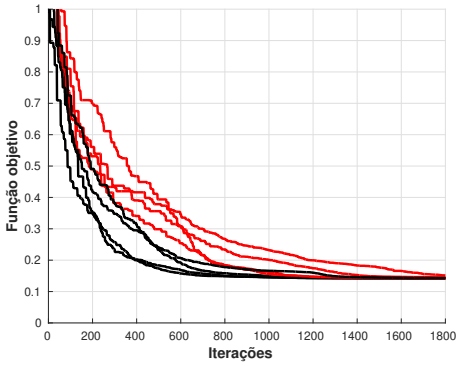
Figura 5.7: Variáveis manipuladas para o sistema 4×5 com DE-NN

A figura 5.8 apresenta os gráficos de convergência para as simulações com DE-NN e DE clássico. Para cada gráfico, foram incluídas as 2 melhores e as 2 piores execuções de cada algoritmo. Em vermelho são mostrados os gráficos para o algoritmo modificado, e em preto os gráficos para o DE clássico.

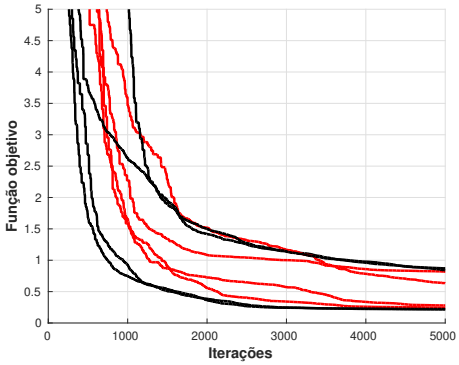
O algoritmo DE-NN possui uma taxa de convergência menor em relação ao DE clássico, principalmente durante a primeira metade das iterações, convergindo para valores próximos ao DE clássico à medida que o algoritmo progride. Para a função objetivo (2.4) os resultados mostram que especialmente para essa função o algoritmo DE-NN, além de convergir mais lentamente, tem dificuldades de alcançar valores de mínimo semelhantes ao DE clássico. Para a função (2.6) os gráficos de convergência do DE clássico e do DE-NN estão alinhados, não havendo diferenças significativas. Para a função (2.5) o DE-NN



(a) Convergência para função objetivo 2.4
DE-NN(vermelho) e DE clássico(preto)



(b) Convergência para função objetivo 2.5
DE-NN(vermelho) e DE clássico(preto)



(c) Convergência para função objetivo 2.6
DE-NN(vermelho) e DE clássico(preto)

Figura 5.8: Comparação das taxas de convergência para a destilaria 4×5

converge mais lentamente durante a maior parte das iterações, alcançando o DE clássico na segunda metade das iterações.

Os gráficos de convergência para a destilaria 4×5 estão em linha com os observados para a destilaria 2×2. Em ambos os casos o algoritmo DE-NN apresentou convergência mais lenta, sendo que para a função (2.4) o algoritmo tem mais dificuldades em alcançar os valores de mínimo obtido com o DE clássico.

5.2.1 Experimento e análise para o sistema 4×5

Assim como feito para o sistema de Wood & Berry, foi repetido o experimento III do capítulo 4 para o sistema 4×5 para avaliar o funcionamento do algoritmo aplicado ao problema de controle quando o tamanho da base de dados usada pela rede aumenta. Os resultados são apresentados na tabela 5.6 para as funções objetivo (2.4) e (2.5). Devido ao tempo maior necessário para calcular o problema da destilaria 4×5, o número de amostras considerado para o experimento foi de 5 execuções. O tamanho da base de dados varia de 200, 300, 500 e 1000 pontos para treinar a rede neural. Todos os demais parâmetros do algoritmo são iguais aos usados na criação da tabela 5.4.

Tabela 5.6: Impacto do tamanho da base de dados para treinamento da rede neural no algoritmo DE-NN - Destilaria 4×5

Função baseada em norma de sistema (H_2)				
Número de vetores da base de dados	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**solução final**	**Número de execuções que não convergiram**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	
200	35,76 ± 2,49	85,62 ± 1,16	$0{,}14 \pm 3{,}8 \times 10^{-3}$	0
300	34,72 ± 1,63	86,91 ± 1,55	$0{,}14 \pm 1{,}5 \times 10^{-3}$	0
500	32,55 ± 1,22	87,07 ± 1,12	$0{,}14 \pm 1{,}1 \times 10^{-3}$	0
1000	34,22 ± 6.33	86,59 ± 0,98	$0{,}14 \pm 2{,}2 \times 10^{-3}$	0
Função baseada em norma de sistema (H_∞)				
Número de vetores da base de dados	**Número de cálculos economizados(%)**	**Percentagem de acerto**	**solução final**	**Número de execuções que não convergiram**
	$\mu \pm \sigma$	$\mu \pm \sigma$	$\mu \pm \sigma$	
200	35,55 ± 1,11	90,92 ± 0,51	0,57 ± 0,13	0
300	35,77 ± 3,91	89,77 ± 1,22	0,36 ± 0,15	1
500	36,08 ± 3,02	91,10 ± 0,62	0,42 ± 0,04	0
1000	37,03 ± 3,42	91,36 ± 0,53	0,40 ± 0,17	0

Embora o sistema 4×5 demore muito mais do que o sistema 2×2 para concluir o processo de otimização, observa-se que o número de vezes que as funções objetivo (2.5) e (2.4) não convergem é muito menor. Apenas a função (2.4) apresentou um caso de não convergência (com base de dados de 300 pontos). Esse fato pode estar relacionado à diferença nos atrasos de transporte de cada sistema. Para a destilaria de Wood & Berry os atrasos são maiores e estão presentes em todos os elementos da matriz de funções de transferência. Para a destilaria 4×5 os atrasos são menores e estão restritos a apenas alguns elementos da matriz de funções de transferência.

5.3 Considerações finais

Tendo em vista os resultados das simulações apresentados, tem-se que o algoritmo DE-NN é uma opção viável e de menor custo computacional para a resolução de problemas de síntese de controladores multivariáveis, principalmente quando a função objetivo utilizada é uma norma de sinal, como a função (2.6). Para a destilaria de Wood & Berry, um sistema 2×2 com alto grau de acoplamento e atrasos de transporte elevados, a aplicação do algoritmo DE-NN se mostrou bastante flexível principalmente para função objetivo (2.6), tendo obtido os mesmos resultados mas com custo computacional reduzido. As funções (2.4) e (2.5) exigem que a base de dados usada para treinar a rede neural seja suficientemente grande para evitar que o algoritmo DE-NN não convirja, sendo necessários ajustes adicionais por parte do projetista para determinar o melhor parâmetro. Não obstante, os resultados mostram que os critérios de desempenho obtidos com o DE-NN são similares aos obtidos com o DE clássico ou com diferenças dentro da variância do conjunto de resultados, mas com custo computacional menor. As duas desvantagens do algoritmo são a convergência mais lenta e o menor grau de repetibilidade, atestado pela alta variância encontrada nos conjuntos de dados analisados.

Já para o sistema 4×5, que possui atrasos de transporte menores, o algoritmo DE-NN funcionou bem em todas as situações, reduzindo o custo computacional em torno de 40% e mantendo os critérios de desempenho similares ou com diferenças dentro do desvio padrão em relação aos obtidos com DE clássico. Tal como para o sistema 2×2, o algoritmo DE-NN apresentou os melhores resultados para a função objetivo baseada em norma de sinal (2.6). Para as funções objetivo baseadas em normas de sistema, os resultados também são similares aos obtidos com o algoritmo DE clássico, com diferenças em torno de 10% no tempo de subida e acomodação, e com custo computacional também cerca de 40% menor. Também para esse caso foi notado que as duas desvantagens do algoritmo são a convergência mais lenta e o menor grau de repetibilidade, atestado pela alta variância encontrada nos conjuntos de dados analisados.

Graficamente, tanto para o sistema 2×2 como para o sistema 4×5, as diferenças entre o DE clássico e o DE-NN são difíceis de serem notadas, tendo todos os gráficos um perfil similar.

Capítulo 6

Considerações Finais

Neste capítulo são apresentadas as conclusões finais, tendo em vista o conjunto de dados dos capítulos 4 e 5, bem como as ideias para continuidade da pesquisa.

6.1 Conclusões

Neste trabalho foi apresentado uma nova versão do algoritmo Evolução Diferencial que lança mão de redes neurais artificiais para classificar soluções candidatas e prever se vale a pena ou não calcular a função objetivo.

O algoritmo é apresentado no capítulo 4 junto a uma série de experimentos para validar seu funcionamento e obter conhecimentos sobre o seu comportamento. Com base nesses experimentos, conclui-se que o algoritmo DE-NN funciona como esperado em relação ao seu objetivo de diminuir o número total de cálculos da função objetivo ao mesmo tempo que não muda significativamente o resultado final do processo de otimização, em média, em comparação ao DE clássico para os tipos de problemas para os quais o mesmo foi desenvolvido. Não obstante, a convergência do algoritmo é mais lenta quando comparado ao DE clássico e o nível de repetibilidade é mais baixo.

A partir dos experimentos feitos foi extraído um conjunto de apontamentos que serve de guia para o ajuste e funcionamento do algoritmo, apresentado em detalhes no capítulo 4 e repetido resumidamente a seguir:

- O tempo de execução da rede neural aumenta com número de neurônios da camada intermediária, e o tempo de treinamento cresce com aumento da base de dados usada e também com o número total de neurônios e camadas.

- O número de 10 neurônios para a camada intermediária tende a ser uma boa escolha, haja visto que números maiores que isso não implicam em aumento significativo na precisão da rede (experimento II).

- A economia do custo computacional tende a aumentar usando-se base de dados menores (reinicializando a rede neural mais vezes) e aumentando o número de classes. Por outro lado, a solução final tende a piorar caso esses dois parâmetros sejam levados a extremos, sendo necessário uma solução intermediária.

- Pelo experimento V, vale a pena começar o algoritmo como DE clássico e chavear para a versão modificada apenas após a primeira base de dados ter sido armazenada, evitando-se assim a etapa de criação de uma base de dados exclusiva para a rede neural, que adicionaria custo computacional ao algoritmo.

- Pelo experimento VII, é fundamental a reinicialização da rede neural com novos conjuntos de soluções para que a diminuição do custo computacional seja satisfatória.

A partir desses apontamentos, o algoritmo foi aplicado a dois problemas de controle de síntese de controladores PI centralizados multivariáveis.

Tanto para o sistema 2×2 como para o sistema 4×5 o algoritmo DE-NN reduziu consideravelmente o custo computacional. Graficamente, as diferenças são pequenas e difíceis de serem notadas. Em geral os critérios de desempenho obtidos com o novo algoritmo, principalmente o índice ISE, são similares aos obtidos com o DE clássico. Observou-se que a variância do algoritmo DE-NN é maior com relação ao DE clássico, e que o tempo de acomodação e tempo de subida são os critérios de desempenho que mais sofrem variações.

Do ponto de vista das três funções objetivo utilizadas, tem-se que a função (2.6) (índice ISE) é a que resulta no melhor comportamento em malha fechada, mas ao mesmo tempo é a função com maior custo computacional, com tempo de execução total mais de cinco vezes maior do que as funções baseadas em normas de sistema. Além do melhor desempenho para o sistema em malha fechada, a abordagem com norma de sinal também possui o benefício de poder ser aplicada a problemas em que se deseja obter o melhor controlador para os sinais de referência típicos da planta, sem ter a necessidade de modelar uma matriz de funções de transferência que traduza o comportamento esperado.

A função objetivo (2.4) (baseada na norma H_∞) possui custo computacional maior em relação à função (2.5) (baseada em norma H_2). O índice ISE, tempo de acomodação e sobressinal são melhores com a função (2.4). O tempo de subida varia com cada função dependendo do problema. Com o sistema 4×5 a função (2.5) apresentou índice ISE e tempo de acomodação significativamente maior do que as outras duas funções. Dessa forma, embora o custo computacional seja maior, para as formulações baseadas em modelo de referência testadas, a função objetivo (2.4) resultou em melhor desempenho do que a (2.5).

A utilização de funções objetivo baseadas em normas de sistemas exige que o projetista trate matematicamente os atrasos de transporte da planta (aproximações de Padé) e modele uma função de transferência de referência satisfatória, o que em geral pode demandar testes e nem sempre é trivial. Já a utilização de funções objetivo baseadas em normas de sinais, como a função (2.6), é mais prática e simples de ser usada, pois basta que o projetista disponha do sinal de *setpoint* típico da planta. Dessa forma, a escolha da função objetivo deve estar de acordo com as informações disponíveis e com a relação custo computacional × desempenho. A função (2.6) apresenta o melhor desempenho, e a função (2.4) o melhor custo computacional.

6.2 Propostas de continuidade

A presente pesquisa introduziu e apresentou o novo algoritmo DE-NN e o aplicou à síntese de controladores multivariáveis PI centralizados. Uma primeira linha de continuidade para a aplicação do algoritmo é a sua utilização e validação em problemas de controle robusto, onde sistemas com incertezas nos parâmetros implicam em processos de otimização com alto custo computacional, sendo portanto um tipo de problema onde o novo algoritmo proposto tem alto potencial de economia de tempo e recursos.

Uma segunda linha de continuidade é o estudo aprofundado de como diferentes estruturas da rede neural usada para classificar soluções candidatas afeta o funcionamento do algoritmo. Para esse trabalho, foi usada uma rede simples com 10 neurônios na camada intermediária e tantos neurônios na camada de saída quanto o número de classes utilizadas.

Além disso, como caminho natural para o algoritmo proposto, é de interesse da pesquisa que o mesmo seja aplicado a outros tipos de problemas que envolvam funções objetivo complexas e de alto custo computacional, como projeto de antenas, modelagem de sistemas, modelagem eletromagnética e outros problemas de engenharia.

Foram observadas limitações da aplicação do algoritmo em relação à taxa de convergência e nível de repetibilidade, sendo importante estudar os ajustes necessários para aprimorar o algoritmo e melhorar seu desempenho quanto a esses dois aspectos.

Por fim, é interessante estudar a aplicação da classificação de soluções no caso do DE multiobjetivo, onde as classificações seriam associadas a qual fronteira de Pareto a solução pertence.

Referências Bibliográficas

AKOJWAR, S. G.; KSHIRSAGAR, P. R. Performance evolution of optimization techniques for mathematical benchmark functions. *International Journal of Computers*, v. 1, p. 231–236, 2016. Citado na página 37.

ALBERTOS, A. S. P. Decentralised and decoupled control. In: *Multivariable Control Systems: An engineering approach*. [S.l.]: Springer-Verlag London Limited, 2004. p. 125–164. Citado na página 13.

ALBERTOS, A. S. P. Fundamentals of centralised closed-loop control. In: *Multivariable Control Systems: An engineering approach*. [S.l.]: Springer-Verlag London Limited, 2004. p. 165–188. Citado na página 13.

Arvani, A.; Teshnehlab, M.; Aliyari Sh., M. Robust h_∞ controller design for distillation column based on multi-objective optimization and genetic algorithms. In: *2009 IEEE Symposium on Industrial Electronics Applications*. [S.l.: s.n.], 2009. v. 2, p. 773–777. ISSN null. Citado na página 18.

AYTUG, H.; SAYIN, S. Using support vector machines to learn the efficient set in multiple objective discrete optimization. *European Journal of Operational Research*, v. 193, n. 2, p. 510 – 519, 2009. ISSN 0377-2217. Citado na página 20.

BACHUR, W. E. G. et al. A multiobjective robust controller synthesis approach aided by multicriteria decision analysis. *Applied Soft Computing*, v. 60, p. 374 – 386, 2017. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494617303708>. Citado na página 17.

Balestrino, A. et al. Performance indices and tuning in process control. In: *2006 14th Mediterranean Conference on Control and Automation*. [S.l.: s.n.], 2006. p. 1–6. ISSN null. Citado na página 18.

BANDARU, S.; NG, A.; DEB, K. On the performance of classification algorithms for learning pareto-dominance relations. In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2014. p. 1139–1146. ISSN 1089-778X. Citado na página 20.

BOETTCHER, S.; PERCUS, A. Optimization with extremal dynamics. *Physical review letters*, v. 86, p. 5211–4, 07 2001. Citado na página 16.

Bristol, E. On a new measure of interaction for multivariable process control. *IEEE Transactions on Automatic Control*, v. 11, n. 1, p. 133–134, January 1966. ISSN 0018-9286. Citado na página 14.

BUJOK, P.; TVRDIK, J.; POLAKOVA, R. Differential evolution with rotation-invariant mutation and competing-strategies adaptation. *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, p. 2253–2258, 09 2014. Citado na página 34.

CAI, W.-J. et al. Normalized decoulpling: A new approach for mimo process control system design. *Industrial & Engineering Chemistry Research - IND ENG CHEM RES*, v. 47, 08 2008. Citado na página 15.

CAMPESTRINI, L.; FILHO, L. C. S.; BAZANELLA, A. S. Tuning of multivariable decentralized controllers through the ultimate-point method. *IEEE Transactions on Control Systems Technology*, v. 17, n. 6, p. 1270–1281, Nov 2009. ISSN 1063-6536. Citado na página 14.

CHUGH, T. et al. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing*, v. 23, n. 9, p. 3137–3166, May 2019. ISSN 1433-7479. Citado 3 vezes nas páginas 19, 20 e 23.

COELHO, L.; MARIANI, V. Firefly algorithm approach based on chaotic tinkerbell map applied to multivariable pid controller tuning. *Computers & Mathematics with Applications*, v. 64, p. 2371–2382, 10 2012. Citado na página 16.

COELHO, L.; PESSOA, M. W. A tuning strategy for multivariable pi and pid controllers using differential evolution combined with chaotic zaslavskii map. *Expert Syst. Appl.*, v. 38, p. 13694–13701, 05 2011. Citado na página 16.

COMMAULT, C.; DION, J. Transfer matrix approach to the triangular block decoupling problem. *Automatica*, v. 19, p. 533–542, 09 1983. Citado na página 15.

DAS, S.; MULLICK, S. S.; SUGANTHAN, P. Recent advances in differential evolution - an updated survey. *Swarm and Evolutionary Computation*, v. 27, p. 1 – 30, 2016. ISSN 2210-6502. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2210650216000146>. Citado 2 vezes nas páginas 21 e 33.

DAS, S.; SUGANTHAN, P. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, v. 15, p. 4 – 31, 03 2011. Citado na página 21.

DEVIKUMARI, A.; VELAPPAN, V. Decentralized pid controller design for 3x3 multivariable system using heuristic algorithms. *Indian Journal of Science and Technology*, v. 8, 07 2015. Citado na página 17.

DRAA, A.; BOUZOUBIA, S.; BOUKHALFA, I. A sinusoidal differential evolution algorithm for numerical optimisation. *Applied Soft Computing*, v. 27, p. 99 – 126, 2015. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494614005559>. Citado na página 34.

DUCHEYNE, E.; BAETS, B. D.; WULF, R. D. Fitness inheritance in multiple objective evolutionary algorithms: A test bench and real-world evaluation. *Applied Soft Computing*, v. 8, n. 1, p. 337 – 349, 2008. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S156849460700021X>. Citado na página 20.

Eita, M. A.; Shoukry, A. A. Constrained dynamic differential evolution using a novel hybrid constraint handling technique. In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.: s.n.], 2014. p. 2421–2426. ISSN 1062-922X. Citado na página 34.

FERREIRA, P. M. G. The exact model matching revisited. *SBA Controle & Automacao*, v. 10, p. 1–6, 09 1999. Citado na página 18.

GAGNON, E.; POMERLEAU, A.; DESBIENS, A. Simplified, ideal or inverted decoupling? *ISA Transactions*, v. 37, p. 265–276, 09 1998. Citado na página 15.

GAMPERLE, R.; MULLER, S.; KOUMOUTSAKOS, A. A parameter study for differential evolution. *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, v. 10, p. 293–298, 08 2002. Citado na página 36.

GRANING, L.; JIN, Y.; SENDHOFF, B. Individual-based management of meta-models for evolutionary optimization with application to three-dimensional blade optimization. In: YANG, S.; ONG, Y.-S.; JIN, Y. (Ed.). *Evolutionary Computation in Dynamic and Uncertain Environments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 225–250. ISBN 978-3-540-49774-5. Citado na página 19.

HAJI, V.; MONJE, C. Fractional-order pid control of a mimo distillation column process using improved bat algorithm. *Soft Computing*, v. 23, p. 1–20, 08 2018. Citado na página 18.

HALEVI, Y.; PALMOR, Z.; EFRATI, T. Automatic tuning of decentralized pid controllers for mimo processes. *Journal of Process Control*, v. 7, p. 119–128, 04 1997. Citado na página 14.

HE, M.-J. et al. Rnga based control system configuration for multivariable processes. *Journal of Process Control*, v. 19, n. 6, p. 1036 – 1042, 2009. ISSN 0959-1524. Citado na página 14.

HOVD, M.; SKOGESTAD, S. Sequential design of decentralized controllers. *Automatica*, v. 30, p. 1601–1607, 10 1994. Citado na página 14.

ICHIKAWA, K. *Control System Design based on Exact Model Matching Techniques*. 1st. ed. Heidelberg, Germany: Springer-Verlag Berlin Heidelberg, 1985. ISBN 9783540157724. Citado na página 18.

IRUTHAYARAJAN, M. W.; BASKAR, S. Evolutionary algorithms based design of multivariable pid controller. *Expert Systems with Applications*, v. 36, n. 5, p. 9159 – 9167, 2009. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417408008920>. Citado na página 16.

IRUTHAYARAJAN, M. W.; BASKAR, S. Covariance matrix adaptation evolution strategy based design of centralized pid controller. *Expert Systems with Applications*, v. 37, n. 8, p. 5775 – 5781, 2010. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417410000709>. Citado na página 16.

ISHIZAKI, M.; YUBAI, K.; HIRAI, J. A design of h infinity model-matching multivariable controller by model-free controller tuning. *Proceedings of the Japan Joint Automatic Control Conference*, v. 54, p. 193–193, 2011. Citado na página 18.

Islam, S. M. et al. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 42, n. 2, p. 482–500, April 2012. ISSN 1083-4419. Citado na página 21.

JAMIL, M.; YANG, X.-S. A literature survey of benchmark functions for global optimization problems. *Int. J. of Mathematical Modelling and Numerical Optimisation*, v. 4, 08 2013. Citado na página 37.

JIN, Y. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, v. 9, p. 3–12, 10 2005. Citado na página 19.

KADHAR, K. M. A.; BASKAR, S.; AMALI, S. M. J. Diversity controlled self adaptive differential evolution based design of non-fragile multivariable pi controller. *Eng. Appl. Artif. Intell.*, v. 46, n. PA, p. 209–222, nov. 2015. ISSN 0952-1976. Disponível em: <http://dx.doi.org/10.1016/j.engappai.2015.09.015>. Citado na página 17.

KARABOGA, D.; BASTURK, B. Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In: MELIN, P. et al. (Ed.). *Foundations of Fuzzy Logic and Soft Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 789–798. ISBN 978-3-540-72950-1. Citado na página 16.

Kennedy, J.; Eberhart, R. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948. ISSN null. Citado na página 16.

KLEIJNEN, J. P. Regression and kriging metamodels with their experimental designs in simulation: A review. *European Journal of Operational Research*, v. 256, n. 1, p. 1 – 16, 2017. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221716304623>. Citado na página 20.

Kramer, A.; Morgado-Dias, F. Applications of artificial neural networks in process control applications: A review. In: *2018 International Conference on Biomedical Engineering and Applications (ICBEA)*. [S.l.: s.n.], 2018. p. 1–6. ISSN null. Citado na página 13.

KUMAR, V. V.; RAO, V. S.; CHIDAMBARAM, M. Centralized pi controllers for interacting multivariable processes by synthesis method. *ISA transactions*, v. 51 3, p. 400–9, 2012. Citado na página 14.

LATTARULO, V.; SESHADRI, P.; PARKS, G. Optimization of a supersonic airfoil using the multi-objective alliance algorithm. In: *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*. [S.l.: s.n.], 2013. p. 1333–1340. Citado na página 19.

LEE, D. et al. Robust design optimisation using multi-objective evolutionary algorithms. *Computers & Fluids*, v. 37, n. 5, p. 565 – 583, 2008. ISSN 0045-7930. Special Issue Dedicated to Professor M.M. Hafez on the Occasion of his 60th Birthday. Citado na página 19.

LEW, T. L. et al. Metamodelling of auxetic cellular solids with differential evolution optimisation. *physica status solidi (b)*, v. 245, n. 11, p. 2433–2439, 2008. Citado na página 22.

LIESLEHTO, J. Mimo controller design using siso controller design methods. *IFAC Proceedings Volumes*, v. 29, n. 1, p. 1152 – 1156, 1996. ISSN 1474-6670. 13th World Congress of IFAC, 1996, San Francisco USA, 30 June - 5 July. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1474667017578209>. Citado na página 14.

LIU, Y.; SUN, F. A fast differential evolution algorithm using k-nearest neighbour predictor. *Expert Systems with Applications*, v. 38, n. 4, p. 4254 – 4258, 2011. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417410010493>. Citado na página 22.

LOSHCHILOV, I.; SCHOENAUER, M.; SEBAG, M. Dominance-based pareto-surrogate for multi-objective optimization. In: *Simulated Evolution and Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 230–239. ISBN 978-3-642-17298-4. Citado na página 20.

LOSHCHILOV, I.; SCHOENAUER, M.; SEBAG, M. A mono surrogate for multiobjective optimization. *Genetic and Evolutionary Computation Conference 2010 (GECCO-2010)*, 07 2010. Citado na página 20.

LU, X.-F.; TANG, K. Classification- and regression-assisted differential evolution for computationally expensive problems. *Journal of Computer Science and Technology*, v. 27, n. 5, p. 1024–1034, Sep 2012. Citado na página 21.

LUYBEN, W. L. Distillation decoupling. *AIChE Journal*, v. 16, p. 198–203, 03 1970. Citado na página 15.

LUYBEN, W. L. Simple method for tuning siso controllers in multivariable systems. *Industrial & Engineering Chemistry Process Design and Development*, v. 25, 07 1986. Citado na página 14.

MELO, V. V. de; DELBEM, A. C. B. Investigating smart sampling as a population initialization method for differential evolution in continuous problems. *Information Sciences*, v. 193, p. 36 – 53, 2012. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025512000266>. Citado na página 21.

MIRJALILI, S.; MIRJALILI, S. M.; LEWIS, A. Grey wolf optimizer. *Advances in Engineering Software*, v. 69, p. 46 – 61, 2014. ISSN 0965-9978. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997813001853>. Citado na página 16.

MLAKAR, M. et al. Gp-demo: Differential evolution for multiobjective optimization based on gaussian process models. *European Journal of Operational Research*, v. 243, n. 2, p. 347 – 361, 2015. ISSN 0377-2217. Citado na página 22.

MUSKE, K. et al. Crude unit product quality control. *Computers & Chemical Engineering*, v. 15, n. 9, p. 629 – 638, 1991. ISSN 0098-1354. Disponível em: <http://www.sciencedirect.com/science/article/pii/0098135491870244>. Citado 6 vezes nas páginas 30, 31, 37, 50, 53 e 74.

MUSTAFA, H. M. J. et al. An improved adaptive memetic differential evolution optimization algorithms for data clustering problems. *PLOS ONE*, Public Library of Science, v. 14, n. 5, p. 1–28, 05 2019. Citado na página 21.

NAKAYAMA, H.; INOUE, K.; YOSHIMORI, Y. Approximate optimization using computaional intelligence and its application to reinforcement of cable-stayed bridges. In: *Proceedings of the 2006 Conference on Integrated Intelligent Systems for Engineering Design*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2006. p. 289–304. ISBN 1-58603-675-0. Citado na página 19.

NERI, F.; TIRRONEN, V. On memetic differential evolution frameworks: A study of advantages and limitations in hybridization. In: *2008 IEEE Congress on Evolutionary Computation, CEC 2008*. [S.l.: s.n.], 2008. p. 2135–2142. Citado na página 36.

PAREEK, M. K. S.; GUPTA, R. Comparison of different performance index factor for abc-pid controller. *International Journal of Electronic and Electrical Engineering*, v. 07, p. 177–182, 2014. Citado na página 18.

Park, B. E.; Sung, S. W.; Lee, I. Design of centralized pid controllers for tito processes. In: *2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)*. [S.l.: s.n.], 2017. p. 523–528. ISSN null. Citado na página 14.

PARK, S.-Y.; LEE, J.-J. An efficient differential evolution using speeded-up k-nearest neighbor estimator. *Soft Computing*, v. 18, n. 1, p. 35–49, Jan 2014. Disponível em: <https://doi.org/10.1007/s00500-013-1030-x>. Citado na página 22.

PIOTROWSKI, A. P. Review of differential evolution population size. *Swarm and Evolutionary Computation*, v. 32, p. 1 – 24, 2017. ISSN 2210-6502. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2210650216300268>. Citado na página 36.

QASEM, S. N. et al. Memetic multiobjective particle swarm optimization-based radial basis function network for classification problems. *Information Sciences*, v. 239, p. 165 – 190, 2013. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025513002107>. Citado na página 20.

Qin, A. K.; Huang, V. L.; Suganthan, P. N. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, v. 13, n. 2, p. 398–417, April 2009. ISSN 1089-778X. Citado na página 21.

QUINN, S.; SANATHANAN, C. Model matching control for multivariable systems - ii. unstable plants. *Journal of the Franklin Institute*, v. 327, n. 5, p. 713–729, 1990. ISSN 0016-0032. Disponível em: <http://www.sciencedirect.com/science/article/pii/001600329090079X>. Citado na página 18.

REYNOSO-MEZA, G. et al. Controller tuning using evolutionary multi-objective optimisation: Current trends and applications. *Control Engineering Practice*, v. 28, p. 58–73, 07 2014. Citado na página 17.

REYNOSO-MEZA, G. et al. Multiobjective evolutionary algorithms for multivariable pi controller design. *Expert Systems with Applications*, v. 39, p. 7985–7907, 07 2012. Citado 2 vezes nas páginas 16 e 17.

Ronkkonen, J.; Kukkonen, S.; Price, K. V. Real-parameter optimization with differential evolution. In: *2005 IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2005. v. 1, p. 506–513. ISSN 1941-0026. Citado na página 36.

SAKTHIVEL, A. et al. Experimental investigations on ant colony optimized pi control algorithm for shunt active power filter to improve power quality. *Control Engineering Practice*, v. 42, p. 153–169, 06 2015. Citado na página 18.

SANTANA-QUINTERO, L. V.; MONTANO, A. A.; COELLO, C. A. C. A review of techniques for handling expensive functions in evolutionary multi-objective optimization. In: TENNE, Y.; GOH, C.-K. (Ed.). *Computational Intelligence in Expensive Optimization Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 29–59. Citado na página 19.

SEAH, C.-W. et al. Pareto rank learning in multi-objective evolutionary algorithms. In: *2012 IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2012. p. 1–8. ISSN 1089-778X. Citado na página 20.

SHAH, G.; ENGELL, S. Tuning mpc for desired closed-loop performance for mimo systems. In: *Proceedings of the 2011 American Control Conference*. [S.l.: s.n.], 2011. p. 4404–4409. ISSN 2378-5861. Citado na página 13.

SHEN, Y.; CAI, W.-J.; LI, S. Multivariable process control: Decentralized, decoupling, or sparse? *Industrial & Engineering Chemistry Research - IND ENG CHEM RES*, v. 49, 12 2009. Citado 2 vezes nas páginas 13 e 14.

SHEN, Y.; SUN, Y.; XU, W. Centralized pi/pid controller design for multivariable processes. *Industrial & Engineering Chemistry Research*, v. 53, p. 10439–10447, 06 2014. Citado 2 vezes nas páginas 14 e 15.

SHINSKEY, F. G. *Process Control Systems: Application, Design and Tuning*. 3rd. ed. New York, NY, USA: McGraw-Hill, Inc., 1990. ISBN 0070569037. Citado na página 15.

SILVA, G. R. Goncalves da; BAZANELLA, A.; CAMPESTRINI, L. On the choice of an appropriate reference model for control of multivariable plants. *IEEE Transactions on Control Systems Technology*, PP, p. 1–13, 06 2018. Citado na página 27.

SILVA, R. P.; LOPES, R.; GUIMARAES, F. Self-adaptive mutation in the differential evolution: Self- * search. In: *Genetic and Evolutionary Computation Conference, GECCO'11*. [S.l.: s.n.], 2011. p. 1939–1946. Citado na página 34.

SIVANANAITHAPERUMAL, S.; SUBRAMANIAN, B. Design of multivariable fractional order pid controller using covariance matrix adaptation evolution strategy. *Archives of Control Sciences*, v. 24, 06 2014. Citado na página 16.

SKOGESTAD S. POSTLETHWAITE, I. *Multivariable feedback control - Analysis and design*. 1st. ed. Chichester, UK: John Wiley & Sons, 1996. ISBN 0470011688. Citado 2 vezes nas páginas 13 e 15.

Soares, R. F.; e Silva, S. J.; Goncalves, E. N. Differential evolution algorithm applied to sparse pi control synthesis for non-square multivariable systems. In: *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*. [S.l.: s.n.], 2018. p. 893–898. ISSN 2576-3555. Citado na página 17.

Srinivas, M.; Patnaik, L. M. Genetic algorithms: a survey. *Computer*, v. 27, n. 6, p. 17–26, June 1994. ISSN 1558-0814. Citado na página 16.

STORN, R.; PRICE, K. V. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optimization*, v. 11, p. 341–359, 1997. Citado 3 vezes nas páginas 16, 33 e 36.

STROFYLAS, G. et al. Using synchronous and asynchronous parallel differential evolution for calibrating a second-order traffic flow model. *Advances in Engineering Software*, v. 125, p. 1 – 18, 2018. ISSN 0965-9978. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997817308645>. Citado na página 22.

THYAGARAJAN, T.; YU, C.-C. Improved autotuning using the shape factor from relay feedback. *Industrial & Engineering Chemistry Research*, v. 42, n. 20, p. 4425–4440, 2003. Citado na página 14.

TUNG, L. S.; EDGAR, T. F. Analysis of control-output interactions in dynamic systems. *AIChE Journal*, v. 27, p. 690 – 693, 07 1981. Citado na página 14.

V, D. R.; MANICKAM, C. Simple method of designing centralized pi controllers for multivariable systems based on ssgm. *ISA Transactions*, v. 56, 12 2014. Citado na página 14.

WANG, Q.; HUANG, B.; GUO, X. Auto-tuning of tito decoupling controllers from step tests. *ISA transactions*, v. 39, p. 407–18, 02 2000. Citado na página 15.

WANG, Q.-G. et al. Auto-tuning of multivariable pid controllers from decentralized relay feedback. *Automatica*, v. 33, p. 319–330, 03 1997. Citado na página 14.

WANG, Y.; CAI, Z.; ZHANG, Q. Enhancing the search ability of differential evolution through orthogonal crossover. *Information Sciences*, v. 185, n. 1, p. 153 – 177, 2012. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025511004518>. Citado na página 21.

WEBER, M.; TIRRONEN, V.; NERI, F. Scale factor inheritance mechanism in distributed differential evolution. *Soft Comput.*, v. 14, p. 1187–1207, 09 2010. Citado na página 36.

WITCHER, M. F.; MCAVOY, T. J. Interacting control systems: steady state and dynamic measurement of interaction. *ISA Transactions*, v. 16, p. 35–41, 01 1977. Citado na página 14.

WOOD, R.; BERRY, M. Terminal composition control of a binary distillation column. *Chemical Engineering Science*, v. 28, n. 9, p. 1707 – 1717, 1973. ISSN 0009-2509. Citado 5 vezes nas páginas 28, 37, 50, 53 e 68.

XIONG, Q.; CAI, W.-J.; HE, M.-J. Equivalent transfer function method for pi/pid controller design of mimo processes. *Journal of Process Control*, v. 17, p. 665–673, 09 2007. Citado na página 14.

YANG, M. et al. An improved adaptive differential evolution algorithm with population adaptation. In: *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*. [S.l.: s.n.], 2013. p. 145–152. Citado na página 21.

YANG, X.; GANDOMI, A. H. Bat algorithm: a novel approach for global engineering optimization. *Engineering Computations*, v. 29, n. 5, p. 464–483, 2012. Citado na página 18.

YANG, X.-S. Firefly algorithms for multimodal optimization. In: *Proceedings of the 5th International Conference on Stochastic Algorithms: Foundations and Applications*. Berlin, Heidelberg: Springer-Verlag, 2009. (SAGA'09), p. 169–178. ISBN 3-642-04943-5, 978-3-642-04943-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1814087.1814105>. Citado na página 16.

YANG, Y.-S. Decoupling control design via linear matrix inequalities. *IEE Proceedings - Control Theory and Applications*, v. 152, p. 357–362(5), July 2005. Citado na página 15.

ZENG, G. et al. Design of multivariable pid controllers using real-coded population-based extremal optimization. *Neurocomputing*, v. 151, p. 1343–1353, 2015. Citado na página 16.

ZHANG, W. Z.; BAO, J.; PETER, L. Control structure selection based on block-decentralized integral controllability. *Industrial & Engineering Chemistry Research - IND ENG CHEM RES*, v. 42, 09 2003. Citado na página 15.

Zhong, J. et al. A differential evolution algorithm with dual populations for solving periodic railway timetable scheduling problem. *IEEE Transactions on Evolutionary Computation*, v. 17, n. 4, p. 512–527, Aug 2013. ISSN 1089-778X. Citado na página 21.

ZHONG, J.; ZHANG, J. Sde: A stochastic coding differential evolution for global optimization. *GECCO'12 - Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*, 07 2012. Citado na página 21.