UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA QUIMÍCA

MODELAGEM DINÂMICA E CONTROLE DE PROCESSOS NÃO LINEARES:

Uma aplicação de algoritmos genéticos para treinamento de redes neurais recorrentes

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia Química do Centro Tecnológico da Universidade Federal de Santa Catarina, como requisito parcial à obtenção do título de Mestre em Engenharia Química.
Orientador. Prof. Nestor Roqueiro

CARLOS ALBERTO CLAUMANN

Florianópolis – SC
1999

MODELAGEM DINÂMICA E CONTROLE DE PROCESSOS NÃO LINEARES:
Uma aplicação de algoritmos genéticos para treinamento de redes neurais recorrentes

por

CARLOS ALBERTO CLAUMANN

Dissertação aprovada como requisito parcial para obtenção do título de mestre no Curso de Pós-Graduação em Engenharia Química, área de concentração Desenvolvimento de Processos Químicos e Biotecnológicos, pela comissão formada por:

Nestor Roqueiro, D. Sc
Orientador

Prof. Humberto Jorge José, Dr. rer. nat.
Coordenador do Curso de Pós-Graduação em Eng. Química

Banca Examinadora:

Ariovaldo Bolzan, D. Sc

Enrique Luis Lima, D. Sc

Florianópolis (SC), Fevereiro de 1999.

AGRADECIMENTOS

Aos professores Nestor Roqueiro e Ariovaldo Bolzan pela orientação, e principalmente pela amizade.

Ao Departamento de Engenharia Química e Alimentos pela oportunidade concedida.

À CAPES pelo apoio financeiro.

Aos colegas do Departamento de Engenharia Química, pelo imenso auxílio e paciência.

Aos amigos do laboratório pelo auxílio e ensinamentos.

A meus pais.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Em processos químicos é comum a presença de não linearidades que estão associadas principalmente às reações químicas e ao grande número e interação entre variáveis. Para simular, otimizar ou ampliar tais processos torna-se indispensável a obtenção de um modelo. Entretanto, a determinação de um modelo baseado nos princípios físicos pode ser difícil ou mesmo muito trabalhosa em tais casos.

Uma abordagem alternativa é a utilização de modelos empíricos ou semi-empíricos obtidos a partir de dados experimentais, sendo que, as redes neurais artificiais se constituem na técnica mais utilizada ultimamente. Uma rede neural artificial é um sistema de processamento de informações que possui semelhanças com o sistema nervoso biológico.

Na modelagem e controle processos podem ser utilizadas redes estáticas ou dinâmicas. As redes dinâmicas possuem um capacidade maior de identificação devido à presença de realimentações, o que inclusive, torna esta classe de redes mais semelhantes às equações diferenciais, se comparadas às redes estáticas. A maior dificuldade encontrada na utilização de redes dinâmicas para identificação de processos está na determinação dos pesos, que não pode, em geral, ser realizada utilizando métodos tipo gradiente devido a realimentação das saídas. Por esta razão técnicas provenientes da Inteligência Artificial tem recebido recentemente uma considerável atenção. Dentre estas técnicas podem-se ressaltar os algoritmos genéticos, conhecidos como *GAs*, que são algoritmos de otimização estocásticos baseados em mecanismos simplificados de adaptação e evolução das espécies.

A área de controle de processos melhor desenvolvida é a de sistemas lineares, onde há metodologias estabelecidas para projeto de controladores. No entanto, um controlador linear pode apresentar um desempenho insatisfatório no controle de processos não lineares. O sucesso do controle preditivo baseado em modelos lineares, para processos que podem ser descritos por tais modelos, tem motivado a extensão dessa metodologia para problemas de controle não linear.

A formulação do problema de controle é análoga a do caso linear, exceto que um modelo dinâmico não linear é utilizado para predizer o comportamento futuro do processo.

Neste trabalho utilizou-se um algoritmo genético para o treinamento de uma de rede neural recorrente. Esta, por sua vez, foi aplicada na modelagem de dois *benchmarks* de identificação em Engenharia Química: um *CSTR* e um Bioreator e, na obtenção de um modelo para controle preditivo de nível de um tanque cônico-cilíndrico.

ABSTRACT

In chemical processes, non-linear behavior is quite common, mostly associated to chemical reactions, to the great number of variables and to the interactions among them. In order to simulate, optimize or scale up processes, the development of a model is essential. However, the proposition of a model based on physical principles may be difficult or too laborious.

An alternative approach is the use of empirical or semi-empirical models obtained from experimental data. In this context, artificial neural networks have been one the most employed from the available techniques. An artificial neural network is an information processing system that possesses some similarities with the biological neural system.

Both dynamic and static networks can be used in process modeling and control. Dynamic networks have a larger identification capacity due to the feedback, what makes them more similar to differential equations than static networks. The major difficulty found in the use of dynamic networks to process identification is the determination of the weights, that generally, can not be done by means of gradient methods due the feedback. For that reason, techniques coming from Artificial Intelligence have been receiving considerable attention recently. Among those techniques, emphasis should be given to genetic algorithms, known as *GAs*, that are stochastic optimization algorithms based on simplified mechanisms of adaptation and evolution of the species.

The better develop area in process control is the one dedicated to linear systems, presenting well established methodologies to the project of controllers. However, a linear controller may present a poor performance in the control of non-linear processes. The success of the application of predictive control based on linear models in processes that can be described by these models has been motivating the extension of this methodology to non-linear control problems. The formulation of the control problem is analogous to the formulation of the linear case, except for the use of a non-linear dynamic model to predict the future behavior of the process. In this work, a genetic algorithm was used to the training of a recurrent neural network.

This network was used in the modeling of two identification benchmarks in Chemical Engineering: A CSTR and a Bioreactor, and to the predictive control of the level of a conical-cylindrical tank. The results obtained confirm the viability of the use of genetic algorithms in control problems and in the empirical modeling of non-linear processes.

1 INTRODUÇÃO

Muitos dos processos industriais, incluindo colunas de destilação de alta pureza, reatores e sistemas em batelada, podem exibir comportamento bastante complexo. Alguns dos principais fatores da complexidade nos processos são: não linearidades, incertezas, grande número de variáveis e interação entre essas. Problemas deste tipo são típicos da área de Engenharia Química, onde são necessários bons modelos para serem utilizados em simulação, otimização, projeto de equipamentos e análise de estratégias de controle.

O modelo de um processo pode ser obtido a partir dos princípios físicos (balanços de massa, energia e quantidade de movimento). Neste caso, os modelos podem ser utilizados em uma faixa de operação relativamente grande, entretanto, pode ser difícil ou mesmo muito trabalhoso obtê-los.

Uma abordagem alternativa é a utilização de modelos empíricos ou semi-empíricos obtidos a partir de dados experimentais, ou seja, determinados das relações entre as entradas e as saídas. Das várias técnicas de modelagem empírica disponíveis para processos não lineares, a que utiliza redes neurais artificiais é uma das mais utilizadas atualmente. Estas constituem uma ferramenta poderosa de modelagem, quando se dispõe de um quantidade suficiente de informações, devido ao maciço paralelismo presente em sua estrutura, rápida adaptação e inerente capacidade de aproximação. Na modelagem e controle de processos podem ser utilizadas redes estáticas ou dinâmicas. Entretanto, a presença de realimentação das saídas nas redes dinâmicas proporciona uma capacidade de modelagem superior, quando comparadas às redes estáticas.

O algoritmo de aprendizagem mais utilizado no treinamento de redes estáticas é o *backpropagation* descrito por RUMELHART *et alli* (1986) [1]. Este algoritmo necessita, em muitos casos, um grande número de iterações para convergir e apresenta problemas de estagnação em mínimos locais. No caso das redes dinâmicas a determinação dos pesos não pode, em geral, ser realizada utilizando métodos tipo gradiente devido à realimentação das saídas. Por esta razão, técnicas provenientes da Inteligência Artificial tem recebido recentemente uma

considerável atenção. Dentre estas técnicas podem-se ressaltar os algoritmos genéticos, conhecidos como *GAs* (*Genetic Algorithms*), que são algoritmos de otimização estocásticos baseados em mecanismos simplificados de adaptação e evolução das espécies. Os *GAs*, que são algoritmos com paralelismo intrínseco, estão sendo aplicados com sucesso onde outros métodos tradicionais falham. Isto só foi possível devido ao advento dos computadores digitais, que tem se tornado cada vez mais rápidos, baratos e compactos.

Os *GAs* tem provado ser bastante úteis na resolução de problemas complexos, podendo inclusive ser aplicados na síntese (determinação tanto dos parâmetros quanto da arquitetura) de redes neurais.

A área de controle de processos melhor desenvolvida é a dos sistemas lineares, onde há uma metodologia bem estabelecida para projeto de controladores. De fato, muitos dos controladores utilizados na indústria moderna, tal como o PI ou PID, pertencem a essa classe. Nos últimos anos, as estratégias de controle baseadas em modelos (*Model Predictive Control* (*MPC*)) estão sendo muito utilizadas para o controle de processos multivariáveis, tais como os encontrados na indústria petroquímica. Porém, um *MPC* baseado em um modelo linear deve ser ajustado conservativamente para controlar processos não lineares, causando uma degradação do desempenho do sistema de controle. Decorrente dos problemas envolvidos na utilização de controladores lineares para processos não lineares, o interesse no desenvolvimento de estratégias que incorporam o conhecimento de características não lineares tem aumentado nos últimos anos. O sucesso do controle preditivo utilizando modelos lineares tem motivado a extensão dessa metodologia para problemas de controle não linear. A formulação do problema de controle é análoga a do caso linear usando um modelo dinâmico não linear para predizer o comportamento futuro do processo.

Neste trabalho será utilizado um algoritmo genético para o treinamento de uma arquitetura de rede neural recorrente. Esta por sua vez será aplicada na modelagem e controle de processos.

Os objetivos deste trabalho são:

• *Estudo e melhorias dos algoritmos genéticos*: Será efetuado um estudo comparativo entre os principais operadores genéticos encontrados na literatura e outros propostos neste trabalho, baseado na otimização de alguns problemas selecionados. O estudo tem por objetivo determinar o melhor conjunto de operadores, entre todos os testados, e também quais as características presentes nos operadores que aumentam a eficiência do algoritmo genético. Por último, o melhor *GA* determinado será comparado a um *software* de domínio público na otimização de alguns *benchmarks*.

• *Estudo e aplicação dos algoritmos genéticos no treinamento de redes recorrentes*: O melhor algoritmo genético determinado na primeira parte deste trabalho será utilizado no treinamento de uma neural recorrente completamente interconectada. Alguns testes serão efetuados para verificação da capacidade de predição da rede em dois sistemas reconhecidos como *benchmarks* de identificação em Engenharia Química: Um *CSTR* encamisado com três estados estacionários e um bioreator.

• *Implementação de um controlador preditivo não linear baseado uma rede recorrente*: Uma rede recorrente será treinada com o algoritmo genético e utilizada como modelo para controle preditivo de nível de um tanque cônico-cilíndrico.

• *Otimização da função custo do controlador em tempo real*: O algoritmo genético será utilizado como otimizador, em tempo real, da função custo do controlador preditivo baseado em uma rede recorrente.

Este trabalho foi dividido da seguinte forma: No capítulo 2 apresenta-se uma revisão de métodos de otimização para treinamento de redes neurais, e descrevem-se aplicações de redes recorrentes e algoritmos genéticos à modelagem e controle de processos. No capítulo 3 faz-se uma revisão de redes estáticas e recorrentes, mostrando sua utilização na modelagem de sistemas dinâmicos. Os principais fundamentos e o funcionamento dos algoritmos genéticos são descritos no capítulo 4. No capítulo 5 são apresentados os operadores genéticos mais freqüentemente

encontrados na literatura e os propostos neste trabalho. No capítulo 6 faz-se um estudo comparativo de desempenho de operadores genéticos descritos na literatura e os propostos no trabalho. No capítulo 7 apresenta-se a implementação de algoritmos genéticos e redes recorrentes para identificação de um *CSTR* e de um bioreator. No capítulo 8 mostra-se uma aplicação de controle de nível de um tanque cônico-cilíndrico. No capítulo 9 são apresentadas as conclusões deste trabalho.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo foi dividido da seguinte forma: Na primeira seção são descritos os principais algoritmos, baseados em gradiente e estocásticos, utilizados para treinamento de redes neurais. Na segunda seção mostra-se a utilização de redes neurais recorrentes na identificação e controle de processos. Por último, são ilustradas algumas aplicações de algoritmos genéticos também relacionadas às áreas de modelagem e controle de processos.

2.1 Algoritmos para Treinamento de Redes Neurais

Conforme comentado, o algoritmo de aprendizagem mais utilizado no treinamento de redes estáticas é o *backpropagation* baseado na minimização do gradiente do erro quadrático. Entretanto sua convergência é lenta devido ao fato do *backpropagation* ser um método do tipo *steepest descent*. Vários pesquisadores tem utilizado outros métodos que apresentam convergência mais rápida, como por exemplo, métodos baseados em gradiente conjugado, Newton, quasi-Newton etc.

VAN DER SMAGT (1994) [2] apresenta uma comparação de métodos para treinamento de redes *feedforward*. Os métodos de quasi-Newton e gradiente conjugado são revisados, sendo que, o último é mostrado ser um caso especial do método *backpropagation* com um termo adicional de momento. Três problemas envolvendo o treinamento de redes *feedforward* são testados utilizando-se cinco métodos de otimização. É mostrado, devido ao passo de tamanho fixo, que o clássico método *backpropagation* evita razoavelmente bem os mínimos locais. No entanto, a utilização da informação da derivada segunda do erro pode, adicionalmente ao gradiente local, diminuir muito o tempo de treinamento.

ROBITAILLE *et alli* (1996) [3] fazem modificações no método quasi-Newton para evitar o cálculo da matriz *Hessiana* completa, presente em métodos de segunda ordem. As modificações propostas eliminam algumas interações de segunda ordem e o tamanho resultante da matriz *Hessiana* aproximada não é proporcional ao quadrado do número de pesos da rede, e sim ao número de neurônios. O método quasi-Newton modificado foi comparado em dois exemplos ao método quasi-Newton original, e aos métodos *backpropagation* e gradiente conjugado. Os resultados numéricos mostram que o método quasi-Newton modificado apresentou um ganho, em termos do tempo computacional, para problemas em grande escala em relação aos métodos tradicionais.

No caso das rede neurais recorrentes, conhecidas por *RNNs* (*Recurrent Neural Networks*), a maior parte dos algoritmos são modificações do método *backpropagation.* Os primeiros algoritmos baseados em gradiente para treinamento de *RNNs* foram: o *backpropation in time* e o *recursive backpropagation.*

O método *backpropation in time* RUMELHART & McCLELLAND (1986) [4] aproxima uma *RNN* a uma rede estática equivalente de maior dimensão e, utiliza o algoritmo *backpropagation* no treinamento. Deve-se lembrar entretanto que quanto maior a capacidade de predição desejada para a *RNN* maior também será o tamanho da rede estática expandida, tornando o método limitado pela excessiva quantidade de cálculos efetuada.

O algoritmo *recursive backpropagation* é baseado no cálculo das derivadas do erro quadrático em função de cada parâmetro da *RNN.* Este algoritmo foi proposto para o treinamento de *RNN* por WILLIAMS & ZIPSER (1989) [5]. A maior limitação do método é o crescimento exponencial da quantidade de cálculos efetuados em função do número de parâmetros. Para cada parâmetro da rede, deve-se calcular a derivada cruzada do erro em relação a esse e a todos os outros parâmetros presentes.

Além do *backpropation in time* e do *recursive backpropagation* muitos outros algoritmos baseados em gradiente têm sido desenvolvidos recentemente. Uma revisão de métodos baseados em gradiente para treinamento de redes neurais recorrentes pode ser encontrada em PEARLMUTTER (1995) [6].

A aplicação de algoritmos estocásticos no treinamento de redes neurais é uma área de pesquisa que está crescendo rapidamente nos últimos anos. A principal razão é a dificuldade de utilizar métodos baseados em cálculo a medida que crescem as dimensões e a complexidade da topologia da rede. Neste caso, as técnicas baseadas em gradiente possuem problemas de convergência para mínimos locais. Outra razão importante é que os algoritmos estocásticos são de uso mais geral, sendo que, pode-se normalmente treinar qualquer topologia de rede neural, recorrente ou estática, com pequenas modificações no algoritmo.

Os primeiros métodos estocásticos utilizados no treinamento de redes neurais foram a *Máquina de Boltzmann* e o *Simulated Annealing.*

PETERSON & ANDERSON (1987) [7] utilizaram um algoritmo de treinamento baseado na *máquina de Boltzmann*, onde o erro de aproximação da rede foi tratado como a função energia a ser minimizada. O método foi dividido em dois procedimentos aplicados simultaneamente a cada iteração: o primeiro gerava um nova solução do problema baseada na solução atual e, o outro decidia se a antiga solução deveria ser substituída pela nova gerada. Inicialmente, o procedimento de decisão escolhia aleatoriamente uma entre as duas soluções com igual probabilidade de sorteio. A medida que o treinamento avançava, a melhor solução, entre a antiga e nova, recebia maior probabilidade de ser sorteada. Este efeito, conhecido como **resfriamento** da *máquina de Boltzmann*, continuava até o procedimento de decisão se tornar completamente determinístico, ou seja, a solução com menor erro quadrático sempre passaria para a próxima iteração. O método apresenta duas limitações: a primeira é que o método tem garantia de convergir para o mínimo global. Entretanto, o tempo requerido para encontrá-lo pode ser inaceitável mesmo para redes de dimensões pequenas. A segunda é a determinação da função de resfriamento da máquina que depende do problema a ser otimizado.

VAN DEN BOUT & MILLER (1989) [8] aplicaram o *Simulated Annealing* no treinamento de uma *RNN*. O *Simulated Annealing* é uma técnica de otimização bastante semelhante a *máquina de Boltzmann* mas utiliza um procedimento diferente de geração de novas soluções.

Dentre os métodos de otimização estocásticos, os *GAs* tem se destacado. Estes são uma classe de algoritmos de pesquisa baseados em mecanismos simplificados da seleção natural e da adaptação dos seres vivos. A medida que a complexidade do espaço de pesquisa aumenta, os algoritmos genéticos são uma alternativa atraente aos métodos baseados em gradiente. Nesta área, alguns trabalhos envolvendo topologias tal como a *feedforward*, de base radial ou recorrente podem ser encontrados.

SAYED (1994) [9] empregou um algoritmo genético como método de treinamento para redes neurais recorrentes. No trabalho são descritas aplicações a problemas tais como o *XOR* e o controle de um pêndulo invertido.

MONTANA (1995) [10] utilizou um algoritmo genético com várias combinações de operadores genéticos para o treinamento de redes do tipo base radial e *feedforward*. No último caso, obteve um erro quadrático de treinamento menor do que o método *backpropagation*.

MANDISCHER *et alli* (1998) [11] utilizaram redes neurais *feedforward* e algoritmos genéticos para predição de propriedades termodinâmicas. Dados experimentais foram obtidos da literatura, e utilizados para treinamento das redes. O objetivo foi prever a entalpia de vaporização dado um conjunto de parâmetros que identificavam a estrutura molecular das moléculas e a temperatura. A predição das redes treinadas também foi comparada com modelos físicos, e os resultados mostraram a boa capacidade das redes neurais em correlacionar e predizer o valor de propriedades termodinâmicas.

Possivelmente, o mais importante tipo de generalização permitida pelos algoritmos genéticos seja a utilização de uma função de avaliação arbitrária, sendo que, os métodos baseados em gradiente estão presos ao erro quadrático. A capacidade de lidar com funções de avaliação arbitrárias é importante em problemas relacionados a controle e modelagem de processos, onde o aprendizado não está relacionado apenas a um simples somatórios de erros quadráticos. Funções do tipo do *Minimum Description Length* (*MDL*) RISSANEN (1989) [12], que envolve um custo pelo aumento da complexidade da rede (número de pesos e bias) adicionalmente ao erro quadrático, podem ser utilizadas facilmente. Neste caso o algoritmo genético poderia ser utilizado para o projeto de redes.

ANGELINE *et alli* (1994) [13] utilizaram um algoritmo evolucionário que simultaneamente constrói a topologia e determina os parâmetros de redes neurais recorrentes. O algoritmo desenvolvido é um método empírico que permite a representação de comportamentos complexos que poderiam não ser observados quando se admite uma topologia de rede fixa.

MANDISCHER (1995) [14] desenvolveu um algoritmo evolucionário para projeto de redes neurais *feedforward* e recorrentes. O método constrói redes a partir das informações contidas no grupo de dados. Foram apresentados resultados relacionados às áreas de classificação, aproximação de funções e predição de séries temporais.

SALUZTOWICZ (1995) [15] mostra um estudo de otimização da topologia de redes neurais *feedforward* utilizando algoritmos genéticos. Tarefas como a determinação dos parâmetros da rede, a seleção do grupo de dados e da topologia da rede não são triviais, sendo que, uma escolha errada em qualquer um destes pontos poderia produzir redes sem capacidade de generalização. O estudo feito foi focalizado na otimização da topologia com o objetivo de sintetizar redes com pequena complexidade (quantidade de pesos e bias) e boa capacidade de generalização.

KWOK & YEUNG (1995) [16] apresentam uma revisão de procedimentos para construir redes neurais *feedforward*. Enquanto o clássico método *backpropagation* realiza uma pesquisa no espaço dos pesos para uma topologia de rede fixa, os procedimentos construtivos começam com uma pequena rede que é aumentada pela adição de neurônios na(s) camada(s) escondida(s). Os procedimentos construtivos são classificados de acordo com a arquitetura de rede resultante e o algoritmo de aprendizado utilizado para determinação dos pesos.

RUDOLPH (1996) [17] empregou um algoritmo genético para projetar a topologia de redes neurais *feedforward*. Estas por sua vez foram utilizadas na predição de relações entre variáveis para problemas multivariáveis. Uma teoria geral para o projeto de topologias, de forma a aumentar a capacidade de generalização de uma rede *feedforward*, é desenvolvida, matematicamente provada e mostrada em algumas simulações.

Também podem-se utilizar algoritmos estocásticos conjuntamente com métodos baseados em cálculo (métodos híbridos) para treinamento de redes neurais.

FREITAS *et alli* (1998) [18] derivaram um algoritmo estocástico para treinamento de redes neurais *feedforward* baseado no princípio da maximização da expectação (EM). O método utiliza o conhecimento das derivadas de cada peso da rede, sendo que, estas são obtidas pela aplicação do *backpropagation*. O método foi empregado para identificação de sistemas não lineares.

FREITAS *et alli* (1998) [19] empregaram uma nova estratégia para treinar redes neurais *feedforward* usando o método de Monte Carlo para pesquisa global e o *backpropagation* para pesquisa local. Os resultados obtidos, em termos do tempo computacional e de convergência, com a técnica híbrida são claramente superiores aos conseguidos pela utilização apenas do método de Monte Carlo. Por último, uma rede neural treinada com o método híbrido foi utilizada na predição de preços de contratos no mercado financeiro.

Métodos híbridos envolvendo algoritmos genéticos também são encontrados:

PACHECO & THOME (1997) [20] apresentam os resultados obtidos com o uso de algoritmo híbridos neuro-genéticos para treinamento de redes neurais *feedforward*. O método "Turbo-shake", desenvolvido no trabalho, é comparado a outros algoritmos híbridos e com o método *backpropagation* em relação a velocidade de treinamento, acurácia e taxa de convergência. Os resultados obtidos com o novo método foram bastante promissores.

RUDOLPH (1998) [21] mostra as principais deficiências da utilização da minimização do erro quadrático de aproximação como índice de desempenho no treinamento de redes *feedforward*. É proposta uma nova função objetivo que é otimizada por um algoritmo genético e aplica-se, simultaneamente, o *backpropagation* na minimização do erro quadrático de aproximação. Dessa forma é feita uma pesquisa global otimizando a nova função objetivo proposta e uma pesquisa local baseada na minimização do erro quadrático de aproximação.

2.2 Aplicação de Redes Neurais Recorrentes na Modelagem e Controle de Processos

Redes neurais estáticas e recorrentes tem sido utilizadas com sucesso na identificação de sistemas. No entanto, em vários estudos encontrados na literatura mostra-se a capacidade superior das redes recorrentes na representação de sistemas dinâmicos.

BURROWS & NIRANJAN (1993) [22] compararam a predição efetuada por redes *feedforward* e recorrentes na predição de séries temporais. As redes recorrentes apresentaram desempenho superior de predição, sendo que, o resultado foi atribuído a presença de recorrências que as torna mais semelhantes aos sistemas dinâmicos quando comparadas as redes *feedforward*.

ROISENBERG *et alli* (1997) [23] apresentam um estudo sobre a complexidade de diferentes topologias de rede neural. É mostrado, usando um exemplo concreto, como um dado problema é considerado e resolvido por uma rede *feedforward* e por uma rede recorrente. No trabalho também são feitas considerações relativas a importância da escolha adequada da estrutura em função do tipo de problema a ser resolvido.

Algumas das principais aplicações de redes neurais recorrentes estão na área de identificação de sistemas complexos, como por exemplo, tanques de *pH*, *CSTR* e bioreatores.

YOU & NIKOLAOU (1993) [24] estudaram a aplicação de redes neurais recorrentes na modelagem de sistemas dinâmicos. Na identificação de um sistema *SISO*, utilizaram um tanque agitado onde a variável de interesse era o pH. O bom desempenho do modelo fornecido pela rede pode ser observado nos resultados obtidos. Além desse, dois sistemas *MIMO* foram estudados: Um *CSTR* com reação exotérmica de primeira ordem e um bioreator em regime de batelada.

PARK *et alli* (1993) [25] utilizaram uma rede neural recorrente como um preditor *on-line* de *runaway* em reatores. *Runaway* em um reator exotérmico pode levar a situações perigosas devido ao aumento descontrolado da temperatura e pressão do reator. HUB & JONES (1986) [26] desenvolveram o *OLIWA* (*On-Line Warning*) que é um algoritmo para predição *on-line* do *runaway*. Entretanto, este algoritmo possui limitações para processos com ruído. O desempenho e

robustez do preditor baseado na rede neural recorrente foi demonstrado adicionando-se ruído nas medições e perturbações no processo.

SU *et alli* (1998) [27] empregaram um rede neural recorrente em um *soft-sensor* de um processo industrial de cura de fibras epóxi/grafite. O monitoramento *on-line* do processo de cura pode ser realizado pela avaliação da viscosidade, conteúdo de resina e grau de cura (*DOC*) através de sensores espectroscópicos e dielétricos. No entanto, estes sensores são muitos caros e ainda não fornecem uma precisa história do processo de cura. Foi construído um sensor integrado composto de duas partes: o primeiro é um sensor que fornece o número de *Damkoler* (*DA*) em função do fluxo de calor e o segundo é uma rede neural recorrente que atua como *soft-sensor* predizendo o *DOC* em função do *DA* obtido. A *RNN* foi inicialmente ajustada e validada através de dados experimentais, sendo capaz de um efetivo monitoramento *on-line* do *DOC* de um composto comercial de fibras de epóxi/grafite.

Redes recorrentes estão sendo utilizadas em controle de processos da várias formas: Pode-se empregar redes recorrentes diretamente como modelos não lineares de processos em controladores preditivos. Também pode-se utilizar uma rede recorrente como elemento linearizador de um planta através da realimentação dos estados. A planta, neste caso, seria linearizada e poderia ser controlada por um controlador linear tal como o PID. Outra maneira comum de utilizar redes recorrentes em controle de processos é treinar a rede em tempo real de forma a minimizar o erro entre a saída do sistema e a do modelo de referência. Neste caso, a rede recorrente é encontrada em alguns trabalhos com o nome de neurocontrolador.

ROVITHAKIS & CHRISTODOULOU (1994) [28] tratam do controle de sistemas não lineares com dinâmica desconhecida. Eles propõe um algoritmo dividido em duas fases. Primeiro, a rede neural recorrente é utilizada na identificação, e então, utiliza-se a realimentação de estados para controlar o sistema. O algoritmo foi aplicado com sucesso no controle de velocidade de um motor.

VENUGOPAL *et alli* 1993 [29] propuseram um algoritmo de controle *on-line* para submarinos autônomos. No algoritmo proposto, o controlador consistia de uma rede recorrente de 3 camadas. O algoritmo utilizado no treinamento da *RNN* foi o *Alopex*, HARTH & PANDYA

(1987) [30] que é um método estocástico derivado do *Simulated Annealing*. Os resultados simulados mostraram que o algoritmo de otimização utilizado e o controlador (rede neural recorrente) resultaram em uma excelente estratégia de controle *on-line*. Foram realizados testes para verificar a robustez do controlador baseado na *RNN* e concluiu-se que esse é capaz de rejeitar perfeitamente as perturbações aplicadas.

DELGADO *et alli* (1995) [31] propuseram uma rede neural recorrente dinâmica (*DRNN*), que pode ser vista como uma generalização de rede de *Hopfield*, para identificação e controle de sistemas *afim*. Para controle utilizou-se a realimentação dos estados da rede para cancelar os termos não lineares presentes no modelo da planta.

AHMED & TASADDUQ (1998) [32] apresenta o projeto de um neurocontrolador para uma planta MIMO. O controlador é treinado para minimizar um critério quadrático baseado no erro relativo ao *set-point*. Além do erro, o neurocontrolador também utiliza o valor das variáveis presentes no processo.

BRDYS *et alli* (1998) [33] desenvolveram um controlador adaptativo para plantas não lineares utilizando uma rede neural recorrente. Empregando o modelo dinâmico, calculou-se as ações de controle a serem aplicadas na planta. Estas foram determinadas pela linearização da planta pelo realimentação dos estados. Os parâmetros da rede foram atualizados *on-line*. A estabilidade do algoritmo foi provada para o caso de referência constante e, o desempenho do controlador testado em algumas simulações.

2.3 Aplicação de Algoritmos Genéticos na Modelagem e Controle de Processos

O treinamento e síntese de redes neurais não são as únicas contribuições dos algoritmos genéticos na área de modelagem e controle de processos. Estes podem ser utilizados diretamente na determinação dos parâmetros de controladores. Também, podem-se utilizar os algoritmos

genéticos na otimização da função custo de controladores. Abaixo, citam-se algumas contribuições de algoritmos genéticos em modelagem e controle de processos.

LINKENS & NYONGESA (1995) [34] desenvolveram uma técnica para ajuste de controladores difusos adaptativos baseada em algoritmos genéticos, sendo que, esse foi utilizado para aquisição de regras otimizando critérios em constante alteração. As simulações efetuadas mostraram um desempenho de controle satisfatório para um processo não linear.

LINKENS & NYONGESA (1995) [35] discutem o projeto de controladores inteligentes para sistemas dinâmicos complexos que não podem ser totalmente resolvidos pelos métodos da teoria de controle convencional. Isto acontece principalmente devido a incertezas devido a complexidade e a necessidade de tomada de decisão requerendo o uso de raciocínio heurístico. No trabalho são revisadas as principais técnicas utilizadas no projeto de controladores inteligentes, em particular, a lógica difusa, as redes neurais e os algoritmos genéticos.

ZUO (1995) [36] apresenta um método para projeto adaptativo de controladores PID para sistemas multivariáveis utilizando algoritmos genéticos. O método é aplicado para determinar a atitude de controle e gerenciamento do momento (ACMM) de uma estação espacial. O projeto do controlador é dividido em duas partes. Na primeira, é feita uma identificação periódica da planta utilizando-se o método dos mínimos quadrados com um fator de esquecimento variável. Na segunda, um algoritmo genético é utilizado para ajuste *on-line* da matriz de parâmetros do controlador. As simulações efetuadas mostram a eficácia da estratégia adotada no controle do ACMM.

FOGARTY & BULL (1995) [37] utilizaram um algoritmo genético para otimização de regras de um controlador baseado em um sistema especialista. O controlador foi utilizado para controle da trajetória de um veículo e na combustão de um queimador.

DOWNING *et alli* (1996) [38] aplicaram um algoritmo genético para ajustar os parâmetros de um *PID* e estimar os parâmetros de um sistema linear discreto. O *PID* foi ajustado com sucesso pela otimização de um índice de desempenho estabelecido e o mesmo aconteceu

para a estimação dos parâmetros do sistema discreto. Os resultados obtidos neste último caso foram comparáveis aos atingidos pelo método dos mínimos quadrados.

TREBI-OLLENU & WHITE (1997) [39] discutem uma técnica para ajustar controladores com uma grande número de parâmetros. É definida uma função multiobjetivo difusa que fornece uma maneira eficiente e intuitiva de selecionar os parâmetros do controlador, sendo que, esta função é otimizada por um algoritmo genético. A técnica é aplicada para seleção dos parâmetros de um *slide mode control* utilizado no controle de profundidade de um ROV (veículo submarino controlado remotamente).

DELGADO *et alli* (1997) [40] estudaram a otimização dinâmica de processos variantes no tempo modelados por redes *feedforward*. No trabalho foi proposto um método heurístico cuja função é parar o treinamento quando o melhor modelo tiver sido determinado. Para encontrar a melhor estrutura da rede neural utilizou-se um algoritmo genético, e o método *backpropagation* foi empregado para treinamento dessa. Na predição de sistemas não lineares variantes no tempo, onde a modelagem tradicional falha, o método proposto mostrou-se bastante eficiente.

CONG *et alli* (1998) [41] apresentaram um algoritmo de controle baseado em uma rede neural recorrente para sistemas com mais entradas do que saídas. Este algoritmo combina otimização da lei de controle com minimização dos recursos de controle. Dessa forma, tanto as variáveis controladas quanto as manipuladas foram mantidas em suas regiões desejadas se possível porém, as variáveis controladas possuíam prioridade para atingir o *set point*. Um estudo de caso do controle de temperatura de um *FCCU* (*Fluid Catalytic Cracking Unit*) foi ilustrado e mostrou o bom desempenho do algoritmo.

PHIMISTER *et alli* 1998 [42] utilizaram um algoritmo genético para ajustar quatro controladores *PI* de um coluna de destilação binária. O ajuste de controladores de uma coluna de destilação é procedimento complicado pelos reciclos presentes no processo e pelas interações entre controladores. Existem, neste caso, múltiplos objetivos, tal como eliminar *overshoot*, *off-set* e oscilações. Ajustar cada controlador independentemente baseado em um único objetivo pode levar a desempenhos insatisfatórios. Para resolver este problema foi utilizado um algoritmo genético que emprega regras evolucionárias. A população de soluções é ordenada em função dos

valores dos múltiplos objetivos definidos e as regras evolucionárias são utilizadas para determinar soluções ótimas.

3 FUNDAMENTOS DE REDES NEURAIS

3.1 Introdução

Neste capítulo será feita uma breve descrição sobre redes neurais. Descrições mais detalhadas podem ser encontradas em ROQUEIRO & LIMA (1997) [43], MAZZUCO (1996) [44] e CANCELIER (1997) [45].

Uma rede neural artificial é um sistema para processamento de informações que possui características em comum com o sistema nervoso biológico. As redes neurais artificiais têm sido desenvolvidas como generalizações de modelos matemáticos dos neurônios biológicos baseados nas seguintes hipóteses:

- O processamento de informações ocorre em elementos simples, chamados neurônios;
- Os sinais são passados entre neurônios biológicos através de conexões;
- Cada conexão tem um peso ou força associado;
- Cada neurônio aplica uma função de ativação para determinar o seu sinal de saída.

Uma rede neural é caracterizada por:

- Uma função de ativação;
- Um padrão de conexão entre os neurônios (Arquitetura ou Topologia);
- Um método de determinação dos pesos associados as conexões (Treinamento).

3.2 Função de Ativação Neural

O processamento das informações recebidas por um neurônio é normalmente efetuado em duas etapas:

1) Geração de um único sinal representativo formado a partir de todas as entradas aplicadas ao neurônio: Um neurônio pode emitir apenas um sinal (saída). No entanto, o mesmo sinal pode ser enviado para muitos outros neurônios. Dessa forma, cada neurônio pode receber uma grande quantidade de informações simultaneamente. A maneira mais simples de gerar um único sinal representativo é fazer um somatório ponderado de todas as entradas recebidas. Matematicamente, esta operação é descrita pela equação 3.1:

$$S_t = \sum_{i=1}^{n} w_i . x_i \quad (3.1)$$

Onde:

S_t = Somatório ponderado das entradas;

w_i = i-ésimo peso;

x_i = i-ésima entrada aplicada ao neurônio.

2) *Formação do sinal de saída:* A saída de um neurônio, também chamada de ativação ou nível de atividade, é calculada, em geral, a partir de uma transformação não linear do sinal resultante obtido na etapa 1. O processamento efetuado por um neurônio é ilustrado na figura 3.1.

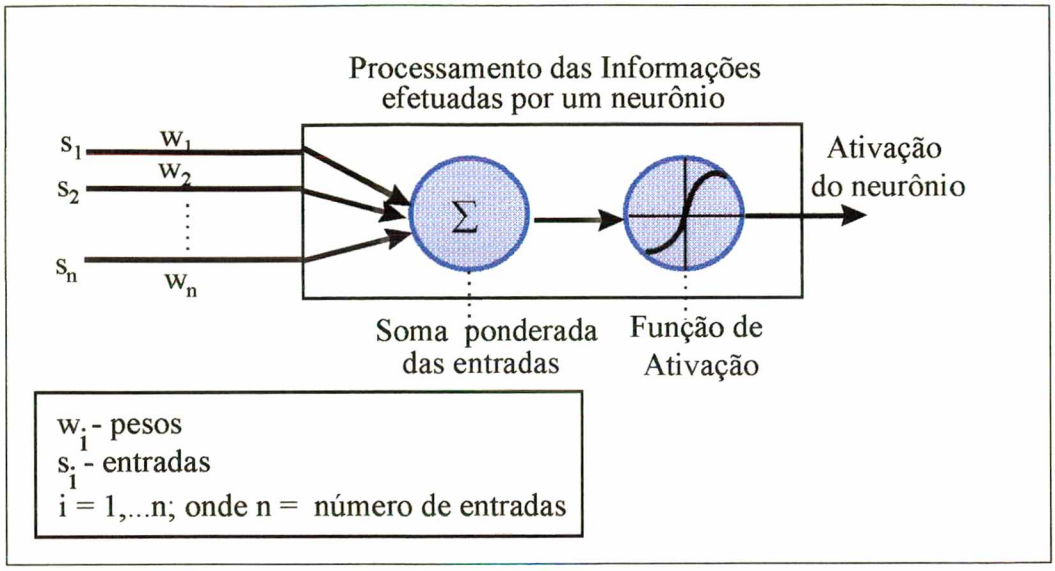

Figura 3. 1 - Processamento das informações efetuadas por um neurônio

Normalmente, a função de ativação deve possuir as seguintes características:

- Não linearidade;
- Saída limitada a um intervalo especificado. Por exemplo, entre 0 e 1;
- De modo semelhante a um neurônio biológico, um neurônio artificial só deve ser ativado quando o sinal resultante de todas as entradas aplicadas for superior a um limite mínimo. Este limite pode ser alterado pela inclusão de um bias em cada neurônio. O bias é um peso alimentado com uma entrada constante, normalmente *um*, que pode ser ajustado por treinamento como qualquer outro peso da rede.

As funções de ativação mais empregadas podem ser vistas na tabela 3.1 e na figura 3.2.

Tabela 3.1 - Funções de Ativação mais empregadas

Nome da Função	Equação
Sinal	$f(x) = \begin{cases} 1 \, se \, x > 0 \\ -1 \, se \, x < 0 \end{cases}$
Degrau	$f(x) = \begin{cases} 1 \, se \, x > 0 \\ 0 \, se \, x < 0 \end{cases}$
Rampa	$f(x) = \begin{cases} -1 \, se \, x \leq -1 \\ x \;\; \text{se -1} < x < 1 \\ 1 \, \text{se} \, x \geq 1 \end{cases}$
Sigmóide	$f(x) = \dfrac{1}{1 + \exp(-x)}$
Tangente hiperbólica	$f(x) = \dfrac{1 - \exp(-2x)}{1 + \exp(-2x)}$
Gaussiana	$f(x) = \exp(-x^2)$

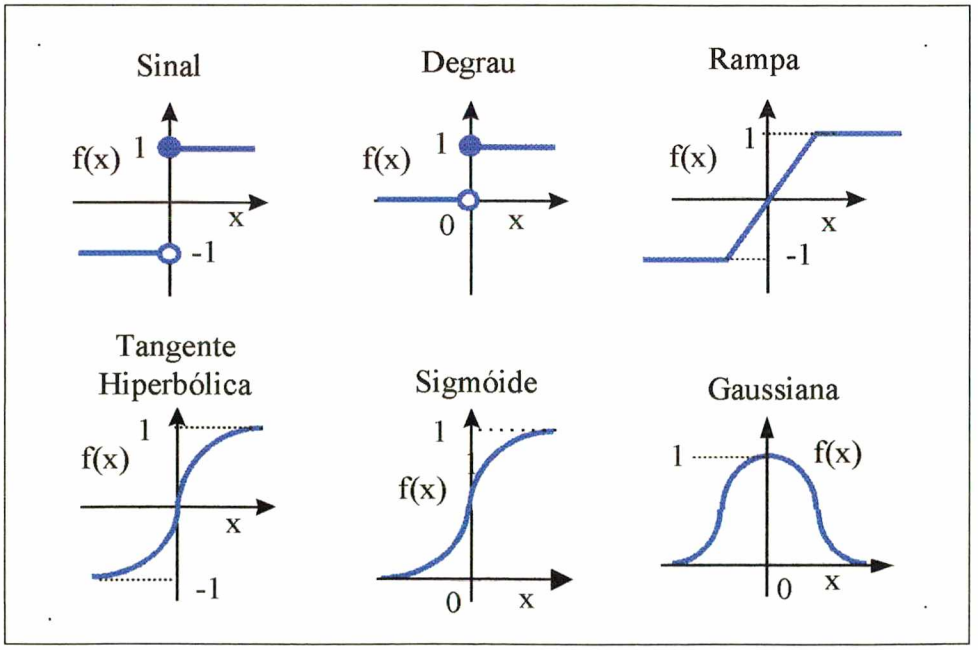

Figura 3. 2- Gráficos de algumas das funções de ativação mais utilizadas

Nas simulações apresentadas nos próximos capítulos, a função sigmóide foi utilizada como função de ativação dos neurônios. Esta foi escolhida pois, além de ser contínua tal como os processos tratados neste trabalho, é a mais utilizada na modelagem e controle de processos, ZUPAN & GASTEIGER 1993 [46]. Cabe ressaltar que o objetivo deste trabalho não é o estudo de diferentes funções de ativação em redes neurais.

3.3 Arquitetura

A arquitetura de uma rede é o padrão de conexão entre os neurônios. Uma maneira conveniente de analisar a arquitetura de uma rede neural é através de camadas. Tipicamente, os neurônios que pertencem a uma camada comportam-se de maneira semelhante, ou seja, tem o mesmo padrão de conexão em relação a neurônios de outras camadas e a mesma função de ativação.

Redes neurais são classificadas em mono ou multicamadas. Muitas redes neurais, tal como a *feedforward*, são organizadas da seguinte maneira:

- *Camada de entrada*: É responsável por receber os padrões de entrada e repassá-los para a próxima camada. Não efetua qualquer forma de processamento;

- *Uma ou mais camadas escondidas*: Estas camadas fazem a maior parte do processamento de informações de uma rede neural. Cada neurônio em uma camada escondida recebe as saídas da camada anterior, efetua o processamento das informações recebidas e envia sua ativação (saída) para a próxima camada;

- *Camada de saída*: Na camada de saída são realizados os últimos passos do processamento e são obtidas as saídas da rede neural.

Como exemplo de uma rede multicamadas, mostra-se na figura 3.3 uma rede *feedforward* composta por 2 neurônios na camada de entrada, 3 na camada escondida e 2 na de saída.

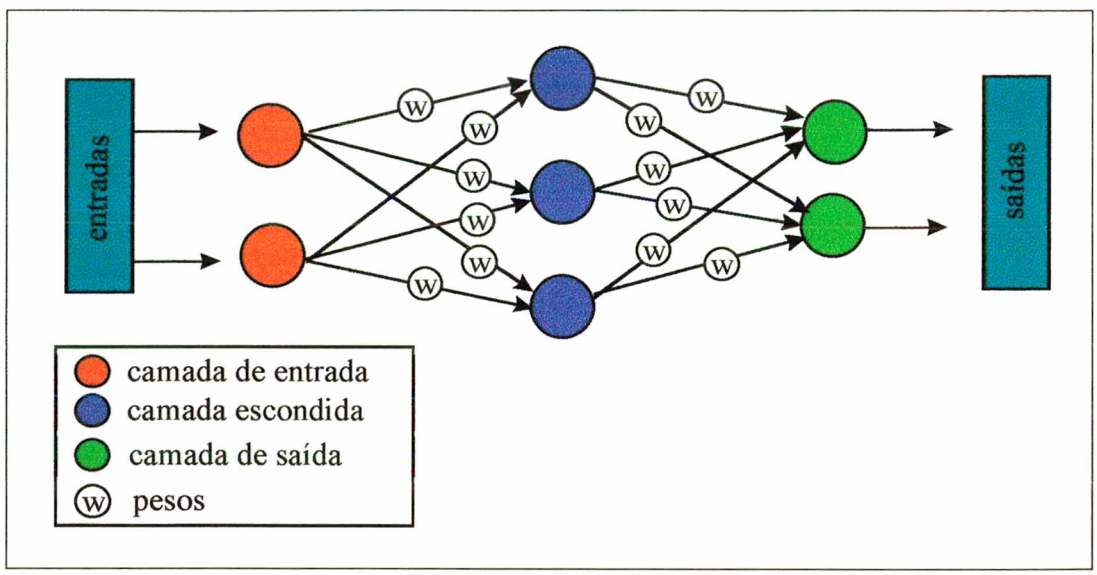

Figura 3. 3 -Rede neural *feedforward* multicamadas

As redes neurais também são classificadas pelo tipo de conexão entre os neurônios. Nas redes estáticas, como por exemplo a *feedforward*, o fluxo de informações passa apenas no sentido das entradas para as saídas. Em redes recorrentes, são possíveis conexões no sentido inverso, ou seja, das saídas paras as entradas (*feedback*), ou mesmo entre neurônios pertencentes a mesma camada (conexões laterais).

Os neurônios em uma *RNN* são geralmente classificados em três categorias: de entrada, saída e escondidos. Os neurônios de entrada apenas repassam as informações externas aos outros neurônios da rede. Os neurônios escondidos e de saída são as unidades processadoras da rede e recebem as ativações de todos os neurônios, inclusive de si próprios (conexões auto-recorrentes). O processamento efetuado pelos neurônios escondidos é análogo ao efetuado pelos de saída, a única diferença é que as ativações dos neurônios de saída são também saídas da rede. Na figura 3.4 pode ser vista uma arquitetura típica de rede neural recorrente completamente interconectada.

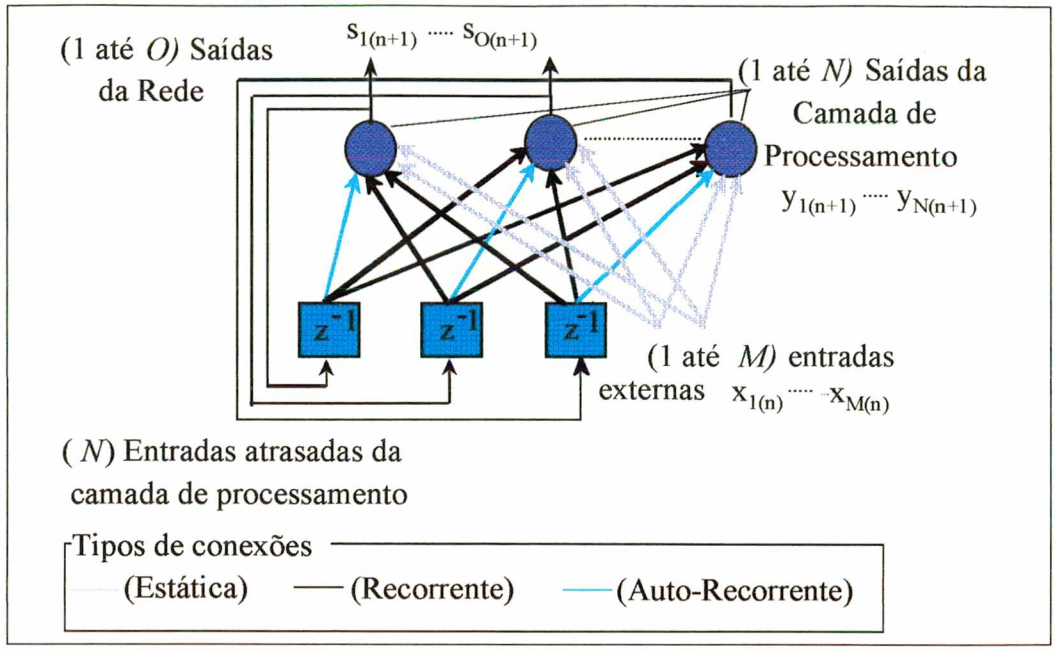

Figura 3. 4- Arquitetura de uma rede neural recorrente

A *RNN* mostrada na figura 3.4 é formada de N neurônios processadores, M entradas externas e O saídas. Na *RNN*, $X_{(n)}$ é um vetor com $(M \times 1)$ entradas externas aplicadas no tempo discreto n. $Y_{(n+1)}$ é o vetor com $(Nx1)$ ativações dos neurônios processadores no tempo $n+1$, das quais $(Ox1)$ ativações compõe o vetor $S_{(n+1)}$ de saídas da *RNN*. As entradas da rede são formadas pela justaposição das ativações atrasadas dos neurônios processadores, ou seja, do vetor $Y_{(n)}$ e das entradas externas contidas em $X(n)$, gerando um vetor de entradas com tamanho $(M+N)x1$ a cada intervalo de amostragem. Dessa forma a rede recorrente possui um total de $(M \times N)$ conexões estáticas e N^2 conexões recorrentes, sendo N das quais auto-recorrentes.

As *RNN* possuem duas características que as distinguem das redes *feedforward*. A primeira é a presença de recorrências, que transformam as *RNN* em modelos capazes de captar a dinâmica de sistemas de equações diferenciais ordinárias (ODE's) não lineares. Em sistemas dinâmicos, o valor de cada variável de estado depende dos seus valores atrasados e das outras variáveis de estado do sistema. Em uma *RNN*, estas informações ficam armazenadas nas auto-recorrências e conexões laterais dos neurônios da camada de processamento. A segunda característica é que a presença das recorrências faz as *RNN* mais similares a sistemas neurais biológicos.

3.4 Treinamento

Adicionalmente à arquitetura, o método de determinação do valor dos pesos (treinamento) é utilizado para diferenciar redes neurais. Há dois principais tipos de métodos de treinamento:

1) Supervisionado: Este tipo de treinamento é utilizado quando se conhecem as saídas de um sistema (alvos) em relação a um determinado conjunto de entradas. O grupo de dados formado pelas entradas e alvos é conhecido por padrões de treinamento. O treinamento supervisionado tem por objetivo determinar um modelo que associe corretamente as entradas aos alvos através de correções aplicadas em cada peso. A intensidade destas correções depende do magnitude do erro de aproximação cometido(diferença entre as saídas da rede e os alvos) e das entradas aplicadas a rede neural.

O método de treinamento supervisionado mais conhecido é o *backpropagation* utilizado em redes estáticas. O motivo da intensa utilização deste método é o conjunto bem estabelecido de equações para a correção dos pesos, sendo que, essa correção pode ser feita após a apresentação de cada padrão ou de todos os padrões (uma época). Uma vez que o erro é conhecido, utilizam-se as equações do método para atualização dos pesos, começando com os da última camada e seguindo seqüencialmente na direção inversa até a primeira camada (daí o nome *backpropagation*). Uma representação esquemática do método *backpropagation* pode ser vista na figura 3.5.

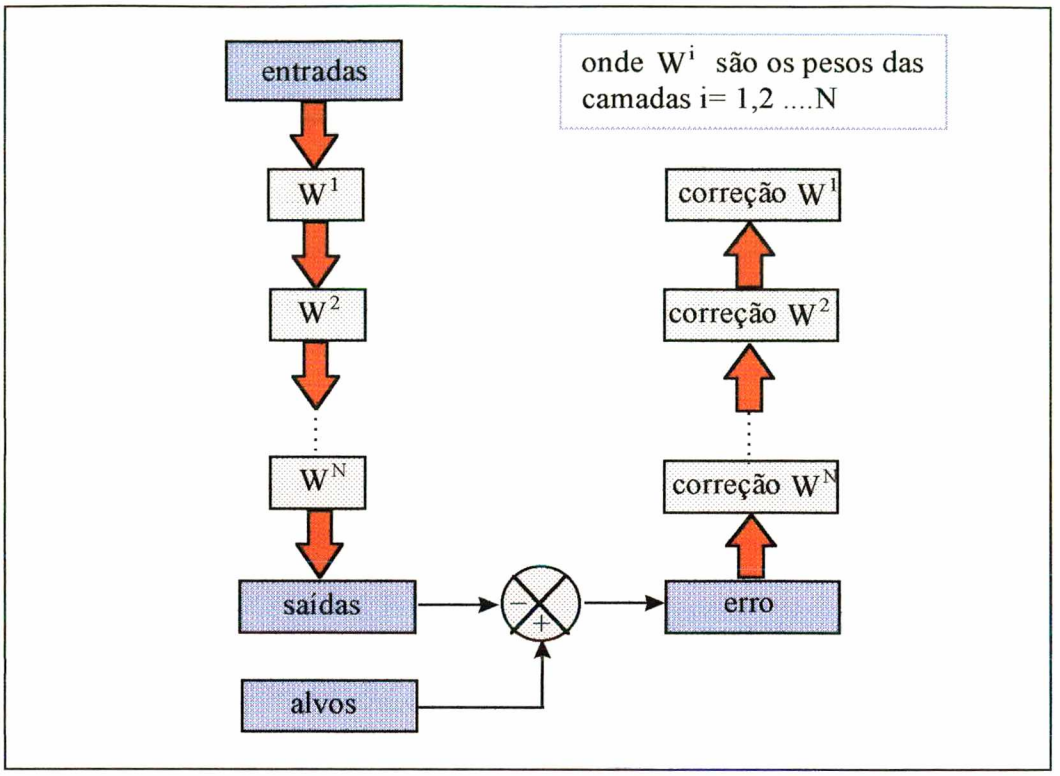

Figura 3. 5 -Treinamento supervisionado (método *backpropagation*)

Um método de treinamento supervisionado para a *RNN* mostrada na figura 3.4 pode ser visto em HAIKIN (1994) [48] e corresponde a uma modificação do método da *backpropagation*. A presença das recorrências aumenta muito a complexidade do método, exigindo para uma *RNN* com N neurônios e M entradas externas, o cálculo de $(N^3 + N^2 \times M)$ derivadas a cada intervalo de tempo;

2) Não Supervisionado. No treinamento não supervisionado uma seqüência de padrões é fornecida à rede, porém, nenhum alvo é especificado. Este treinamento tem por objetivo descobrir relações de similaridade entre os padrões, ou seja, classes. No treinamento não supervisionado determina-se, para cada classe, um exemplar (semelhante a um padrão) que contenha as características mais representativas dessa, sendo que, os padrões serão classificados a partir de sua *distância* em relação aos exemplares característicos. A menor *distância* entre padrões pertencentes a uma mesma classe em relação a padrões de classes diferentes é que possibilita a classificação.

O método de treinamento não supervisionado mais conhecido é a regra de aprendizado de Kohonen, também chamado de *Winner takes it all*. Uma descrição do método, bem como algumas aplicações, podem ser encontradas em FAUSETT (1995) [49].

3.5 Processamento Temporal Utilizando Redes Neurais

O processamento temporal é a predição dos fenômenos que se modificam ao longo tempo. A investigação dos processos dependentes do tempo é um dos principais objetivos da área de controle de processos. Em função das informações disponíveis sobre o comportamento de um processo podem-se utilizar dois métodos de identificação: Em *série-paralelo* e *em paralelo*:

Método de Identificação em série-paralelo: Neste caso deseja-se fazer predições dos valores futuros das saídas de um processo, sendo que, são conhecidos os valores passados e atuais das saídas e as perturbações aplicadas a esse. Para identificar um processo, utilizando o método *série-paralelo* e uma rede neural, deve-se inicialmente formar um grupo de dados cujos alvos são as saídas do processo e, as entradas são os valores atrasados dessas e as respectivas perturbações (a formação de um grupo de dados da maneira descrita é conhecida por *deslocamento temporal*). Os conjuntos de entradas e saídas consecutivos no tempo são chamados respectivamente de *horizontes de aprendizagem passado* e *futuros*. Um exemplo de formação de um grupo dados cujas entradas e saídas são deslocadas no tempo é mostrado na figura 3.6.

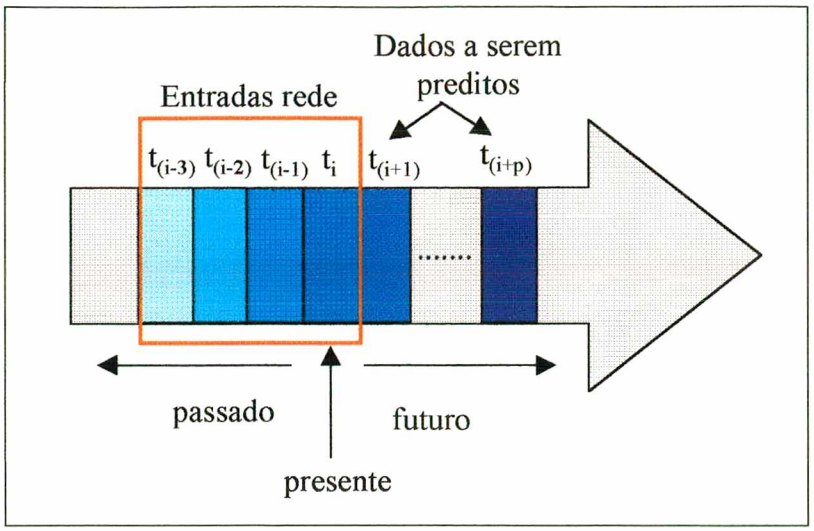

Figura 3. 6 – Dependência temporal dos padrões de entrada e saída (deslocamento temporal)

Na figura 3.6 pode-se observar que o *horizonte de aprendizagem passado* é composto por 4 eventos consecutivos no tempo, ou seja, nos tempos $t_{(i-3)}$, $t_{(i-2)}$, $t_{(i-1)}$ e $t_{(i)}$ (o valor presente). O *horizonte futuro* começa no tempo $t_{(i+1)}$ e acaba em $t_{(i+P)}$, onde p é o número de intervalos do *horizonte futuro*. Após o término de cada intervalo de tempo as entradas da rede devem ser atualizadas. Dessa forma os valores mais atrasados, neste caso os ocorridos em $t_{(i-3)}$, são descartados do vetor de entrada e, os valores reais do processo mais as entradas externas no tempo $t_{(i+1)}$ tornam-se os valores de entrada da rede correspondentes ao instante $t_{(i)}$.

Após obtido o grupo de dados treina-se a rede neural. Como último passo, valida-se o modelo obtido com outro grupo de dados diferente daquele utilizado para treinamento. Se a rede passou no teste pode ser utilizada para efetuar predições.

Cabe ressaltar que no caso da identificação em *série-paralelo* são alimentadas à rede as saídas **reais** atrasadas do processo e não as saídas **preditas** por essa;

Método de Identificação em paralelo: Da mesma forma que o caso anterior, também deseja-se fazer predições dos valores futuros das saídas do processo. No entanto, apenas os valores atrasados e atuais das perturbações aplicadas são conhecidos. Neste caso não se pode utilizar uma rede *feedforward* pois essa apresenta um pobre desempenho de predição quando os valores atrasados e presentes das saídas reais não são conhecidos, e se realimentam as saídas preditas pela

própria rede, HENSON & SEBORG (1997) [47]. As redes *feedforward* podem somente efetuar um mapeamento estático entre as entradas e saídas, HAIKIN (1994) [48], sendo que, o comportamento dinâmico não está dentro da rede *feedforward* mas no *deslocamento temporal* dos padrões. Nesta situação as redes neurais recorrentes podem ser utilizadas.

Para identificar um processo, utilizando o método em *paralelo* e uma *RNN*, deve-se inicialmente formar um grupo de dados cujos alvos são as saídas do processo e, as entradas são os valores atuais e atrasados das perturbações aplicadas, mais as saídas preditas pela *RNN*. A formação do grupo de dados é semelhante a utilizada para redes *feedforward*, exceto que, os valores atrasados das saídas do processo não são aplicados à *RNN*. O próximo passo é treinar a *RNN* com o grupo de dados formado. Por último, valida-se o modelo obtido. Se a *RNN* tiver um bom desempenho na etapa de validação pode ser utilizada como modelo de predição.

Neste trabalho a *RNN* descrita na seção 3.3 será utilizada para modelagem e controle dos processos descritos nos capítulos 7 e 8. As razões mais importantes de escolher uma *RNN* ao invés de uma rede *feedforward* são:

1) Neste trabalho tem-se interesse na obtenção *off-line* de modelos empíricos cuja predição não dependa dos valores reais atrasados do processo e portanto, as redes estáticas não podem ser empregadas. Para o desenvolvimento de trabalhos futuros, uma *RNN* poderia ser utilizada na predição de variáveis de um processo que não podem ser medidas ou que são amostradas ocasionalmente. Neste caso a *RNN* atuaria como modelo de predição em um sensor por *software*;

2) Em aplicações de controle os valores atrasados das saídas de um processo, em geral, são conhecidos possibilitando a utilização de redes *feedforward* como modelos para controladores preditivos. No entanto, as redes dinâmicas são mais adequadas a estas aplicações devido a sua melhor capacidade de generalização, ou seja, conseguem aprender melhor as informações contidas em um grupo de dados e decoram menos os padrões de treinamento, quando comparadas às redes estáticas.

4 TEORIA BÁSICA DE ALGORITMOS GENÉTICOS

4.1 Introdução

Os métodos de otimização podem ser divididos em três principais grupos: os baseados em cálculo, os enumerativos e os algoritmos evolucionários. Os métodos baseados em cálculo dividem-se em duas principais classes: de primeira e segunda ordem. Os métodos de primeira ordem, como por exemplo o passo descendente, procuram o ótimo a partir da informação obtida do gradiente local (utilizam informação da função objetivo e das derivadas de primeira ordem). Os métodos de segunda ordem, tal como o de Newton, utilizam adicionalmente derivadas de segunda ordem. Os métodos baseados em cálculo tem sido extensivamente estudados e melhorados. Mesmo assim, apresentam os seguintes problemas:

• *Sensibilidade às condições iniciais*: O valor determinado para o ótimo depende das condições iniciais indicando a sensibilidade do método à presença de mínimos locais;

• *Tratamento de restrições*: A imposição de restrições no valor da função objetivo e/ou no domínio das variáveis do problema aumentam significativamente a complexidade da otimização;

• *Descontinuidades da função objetivo e de suas derivadas*: Os métodos que utilizam informação de derivadas da função objetivo não podem ser aplicados a problemas definidos por funções descontínuas.

A robustez de um método de otimização pode ser medida por sua capacidade em determinar o ótimo de problemas com as mais variadas características, ou seja, problemas unimodais, multimodais, descontínuos nas derivadas e/ou na função objetivo, combinatoriais etc. Também, é desejável que a determinação do ótimo não dependa das condições iniciais e o método

deve permitir a inclusão de restrições sem aumentar em demasia a complexidade da otimização. Por estas razões, métodos baseados em cálculo não são robustos.

Os métodos enumerativos são baseados em uma pesquisa exaustiva dentro do espaço das variáveis de interesse. O funcionamento básico de um método enumerativo é muito simples: calcula-se a função objetivo para um grande número de combinações válidas entre as variáveis do problema, ou seja, combinações de variáveis que atendam as restrições, e assume-se que o ótimo é o melhor resultado encontrado entre todas as avaliações da função objetivo. As combinações de variáveis utilizadas são distribuídas aleatoriamente dentro do espaço de pesquisa. O método garante a determinação do ótimo global, porém, a função objetivo deve ser avaliada para todas combinações possíveis de variáveis. A simplicidade deste tipo de algoritmo é bastante atrativa (a enumeração é um tipo de pesquisa muito usada quando o número de possibilidades é pequeno). Tais esquemas são robustos conforme a definição anterior, no entanto, os métodos enumerativos apresentam baixa eficiência (excessiva quantidade de avaliações da função objetivo) à medida que o espaço de pesquisa aumenta. Por esta razão pode-se concluir que métodos enumerativos são eficientes apenas para problemas de pequeno porte. Problemas de médio e grande porte exigiriam um grande número de avaliações da função objetivo e, consequentemente, um dispendioso tempo computacional.

Os algoritmos evolucionários, entre os quais estão os Algoritmos Genéticos, a Programação Genética e as Estratégias Evolucionárias, estão sendo muito utilizados pois não apresentam os problemas dos métodos baseados em gradientes (Uma revisão de Algoritmos Evolucionários pode ser encontrada em Bäck *et alli* (1997) [50]). Basicamente, os algoritmos evolucionários avaliam a função objetivo um especificado número de vezes e, destas avaliações, determinam regiões do espaço onde há maior possibilidade que o ótimo seja encontrado. As próximas avaliações serão concentradas preferencialmente em torno destas regiões. Este processo é repetido até reduzir a região de busca à vizinhança de um ponto no espaço das variáveis. Neste momento é dito que o método convergiu. A utilização dos algoritmos evolucionários em uma otimização oferece uma solução mais rápida que os métodos enumerativos pois utiliza a função objetivo para direcionar a pesquisa. Também, a possibilidade de convergência de um algoritmo evolucionário para ótimos locais é menor que os métodos baseados em cálculo.

Para ter uma idéia da eficiência dos métodos descritos, pode-se observar a figura 4.1. Nesta figura, mostra-se qualitativamente a eficiência versus o tipo de problema para um método baseado em cálculo, um método enumerativo e um algoritmo evolucionário. O método baseado em cálculo (esquema especializado) é muito eficiente quando está próximo do ótimo global, funcionando bem somente em uma estreita faixa de problemas (unimodais). Os esquemas enumerativos tornam-se igualmente ineficientes ao longo de todo espectro de problemas (combinatoriais, unimodais, multimodais), como indicado pela curva de baixo desempenho. O comportamento mais desejável seria a curva do algoritmo evolucionário (esquema robusto), não tão eficiente quanto os esquemas especializados na região próxima ao ótimo global porém, com uma alta eficiência em todo o espectro de problemas. Poder-se-ia pensar também em um esquema híbrido, utilizando o algoritmo evolucionário para encontrar a região próxima ao ótimo global e empregar a solução obtida como valor inicial para o esquema especializado. É possível, dessa forma, acelerar a convergência e aumentar a precisão da solução obtida com menor possibilidade de estagnar o processo de busca em ótimos locais.

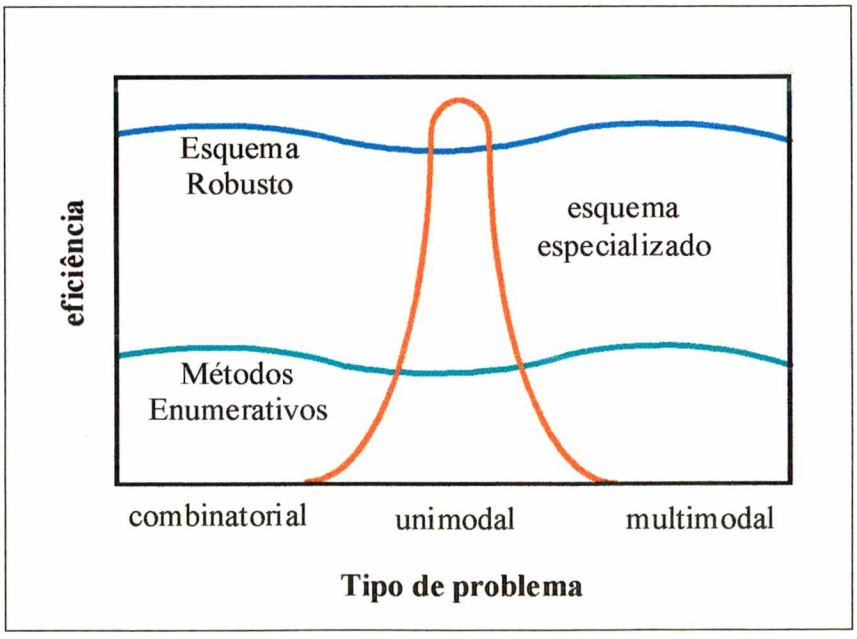

Figura 4. 1- Comparação da eficiência dos métodos de otimização para diversos tipos de problemas

O objetivo desse capítulo é apresentar uma revisão de algoritmos genéticos. Serão descritas as principais diferenças entre os algoritmos genéticos e os métodos clássicos de otimização, o funcionamento básico de um algoritmo genético, o tratamento de restrições e a resolução de problemas multivariáveis utilizando algoritmos genéticos.

4.2 Definição de Algoritmos Genéticos

Os Algoritmos Genéticos (*GAs*) são algoritmos de pesquisa baseados nos mecanismos de seleção natural e adaptação. Estes algoritmos operam com um conjunto de possíveis soluções, denominado "população", para determinação do ótimo. Os elementos da população são combinados através de uma troca de informações de modo a encontrar melhores soluções que as da população original. A troca de informações para determinação da nova população depende do valor da função objetivo, de tal forma que melhores soluções possuem maior possibilidade de serem combinadas. Desse processo obtém-se, em geral, soluções melhores que as originais e toda a população desloca-se em direção ao ótimo. O primeiro Algoritmo Genético foi descrito por HOLLAND (1975) [51] e, foi inspirado em um mecanismo simplificado da adaptação natural dos seres vivos ao ambiente. A adaptação pode ser considerada um processo de modificação progressiva de uma população promovendo um melhor desempenho no ambiente. O ambiente, neste caso, é a função objetivo a ser otimizada.

4.3 Diferenças entre os Algoritmos Genéticos e os Métodos Tradicionais de Otimização

Os algoritmos genéticos diferem dos métodos de otimização baseados em cálculo nos seguintes aspectos:

• As variáveis de um problema a ser otimizado podem ser codificadas em algum alfabeto, tal como o binário. Neste caso, as operações efetuadas resumem-se a cópias e trocas de *bits*;

• *GAs* operam com uma população de possíveis soluções da otimização (indivíduos ou elementos) não com apenas uma solução. A robustez dos *GAs* à estagnação em mínimos locais deve-se a utilização de uma população cujos indivíduos são espalhados inicialmente por todo o espaço de pesquisa. Eventualmente alguns indivíduos da população poderão convergir para ótimos locais, porém, isso dificilmente acontecerá com todos ao mesmo tempo;

• *GAs* utilizam apenas informação das avaliações da função objetivo, não empregando qualquer outro tipo de conhecimento tal como o de derivadas;

• A geração de novos indivíduos nos *GAs* não é realizada de forma determinística mas, através de modificações aleatórias nos indivíduos da população atual.

4.4 Termos Técnicos empregados em Algoritmos Genéticos

Em sistemas naturais um ou mais cromossomos combinam-se para formar a informação genética necessária à formação de um indivíduo. As *strings*, nos sistemas genéticos artificiais, são análogas aos cromossomos nos sistemas biológicos.

Em sistemas naturais a interação entre um organismo e seu ambiente é conhecido como fenótipo. O equivalente ao fenótipo, nos algoritmos genéticos, são as variáveis decodificadas, também chamadas de conjunto de parâmetros, solução alternativa ou ponto no espaço de solução.

Em Genética, os cromossomos dividem-se em genes que são caracterizados por duas propriedades: O alelo (função do gene) e o locus (posição do gene no cromossomo). Na pesquisa genética artificial, as *strings* são formadas por um conjunto de *detetores* ou *características* que

são diferenciados entre si pela função que desempenham e pela posição que ocupam na *string*. Em codificação binária os *bits* são os detetores.

Em populações naturais, o *fitness* é a capacidade das criaturas sobreviverem aos predadores, pestes e outros obstáculos à reprodução. No ambiente artificial, o *fitness* é uma medida relativa de desempenho entre os elementos da população. O *fitness* pode ser determinado diretamente a partir do valor da função objetivo ou de algum critério baseado nela.

4.5 Descrição de uma Geração do *Simple Genetic Algorithm (SGA)*

Nesta seção serão descritos os procedimentos efetuados em uma geração do *Simple Genetic Algorithm (SGA)* que utiliza codificação binária, GOLDBERG (1989) [52], para otimização de um problema monovariável. A resolução de problemas multivariáveis será mostrada posteriormente na seção 4.11. Muitas outras formas de algoritmo genético são possíveis e partem de modificações do *SGA*. Este algoritmo genético pode ser dividido nas seguintes etapas:

1) Inicialização da População: Cada *string* da população é inicializada com uma sequência aleatória de *bits*. As *strings* são limitadas entre $[0_1\ 0_2.....0_n]$ e $[1_1\ 1_2.....1_n]$, onde n é o número de *bits* utilizados na representação da variável a ser otimizada;

2) Decodificação das strings para avaliação do fitness: As *strings* são decodificadas e a função objetivo é calculada para cada uma delas. Em seguida, atribui-se um valor de *fitness* a cada *string* que deve ser, de alguma forma, relacionado ao valor da função objetivo;

3) Realização de operações de troca e cópia de bits de acordo com os operadores genéticos. Selecionam-se aleatoriamente duas *strings* da população (operador de seleção) com probabilidade associada ao *fitness*, e faz-se uma cópia das duas *strings* selecionadas para manipulação de seus detetores, neste caso *bits*. Entre estas duas cópias trocam-se todos os *bits* a partir de uma posição aleatoriamente escolhida na cadeia das *strings* (operador de *crossover*). As *strings* selecionadas

são chamadas de pais e as geradas a partir desta troca de filhas. As *strings* geradas são guardadas separadas da população. Continuando o processo, novos pais são selecionados; estes por sua vez são copiados, sofrem *crossover* e geram filhos até preencher uma nova população, o que equivale a *uma* geração no algoritmo genético. Cabe ressaltar que o *SGA* normalmente trabalha com população de tamanho fixo. A seguir, a nova população substitui a antiga. Para evitar a convergência prematura da população, ou seja, que toda população apresente o mesmo valor para um ou mais detetores, é necessário modificar ocasionalmente o valor dos *bits* (operador de mutação). Isto equivale simplesmente a trocar zeros por uns e vice-versa.

Com a nova população, o algoritmo retorna ao passo 2 e a partir deste momento o processo se torna iterativo. Os indivíduos mais adaptados possuem maior probabilidade de se combinarem e portanto, os filhos herdarão características predominantes destas soluções na maior parte dos casos. A cada geração, novos indivíduos contendo em média genes melhores que a anterior são produzidos. Deste forma a população prosperará pela substituição dos indivíduos ruins por outros mais adaptados após sucessivas gerações. Quando a população deixar de produzir descendência notoriamente diferente da que a originou é dito que o algoritmo genético convergiu. Um fluxograma do *SGA* pode ser visto na figura 4.2.

No apêndice 1 encontra-se um exemplo ilustrativo de otimização utilizando o *SGA*. Este exemplo foi extraído GOLDBERG (1989) [52].

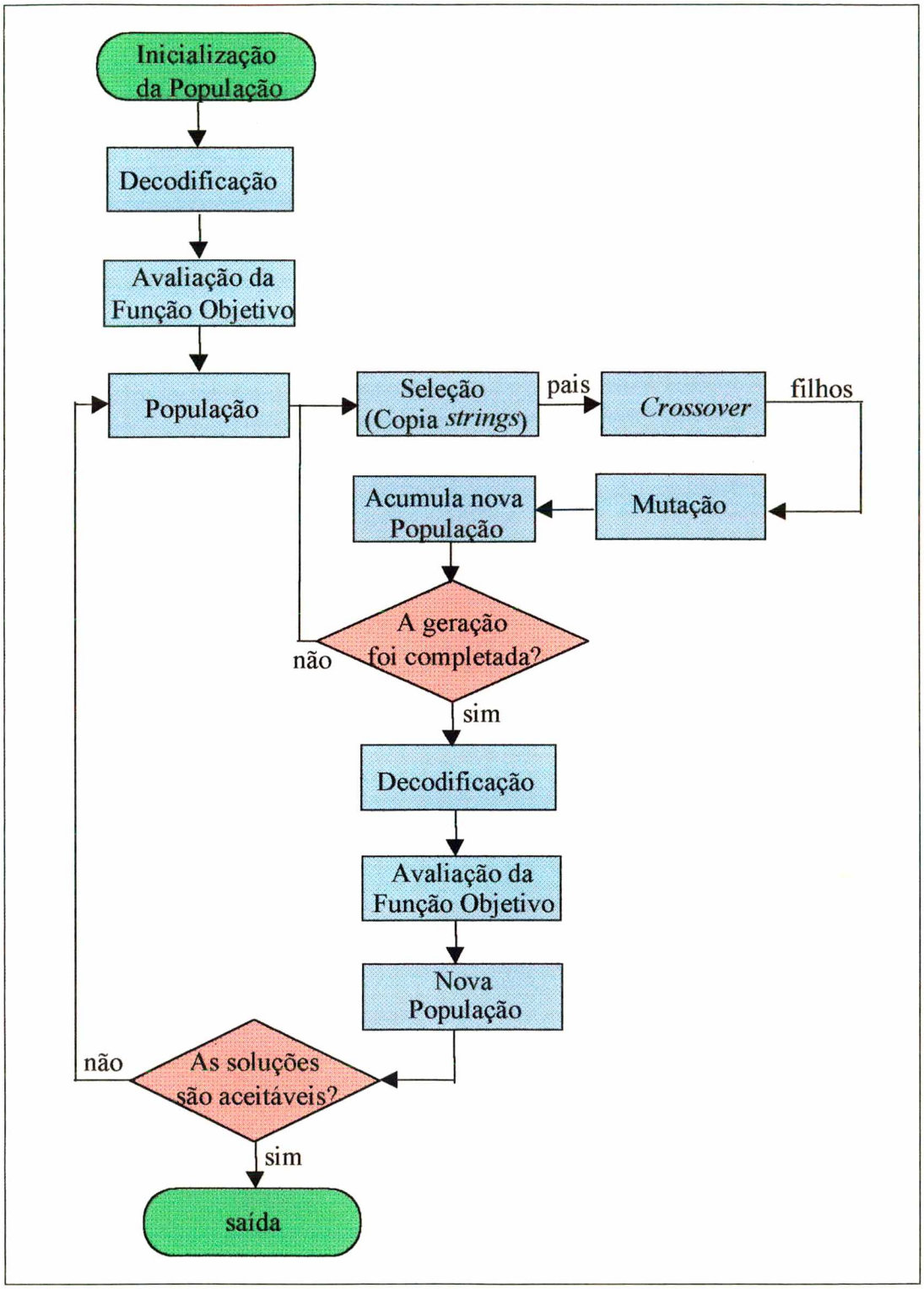

Figura 4. 2- Fluxograma de Cálculo – *SGA*

4.6 Diversidade Genética da População

Um termo bastante conhecido nos algoritmos genéticos é a diversidade da população. Esta é uma medida da diferença, em termos de *bits*, entre as *strings* da população. A forma mais simples de calcular a diversidade, em codificação binária, é verificar o valor de cada detetor em todas as *strings*. Caso algum detetor contenha apenas zeros ou uns (figura 4.3) diz-se que houve perda de diversidade genética naquela posição. A diversidade neste caso pode ser descrita pela equação 4.1

$$D = \frac{N_d}{N_t} \quad (4.1)$$

Onde: D = diversidade;

N_d = número de detetores sem perda de diversidade genética;

N_t = número total de detetores = comprimento da *string*.

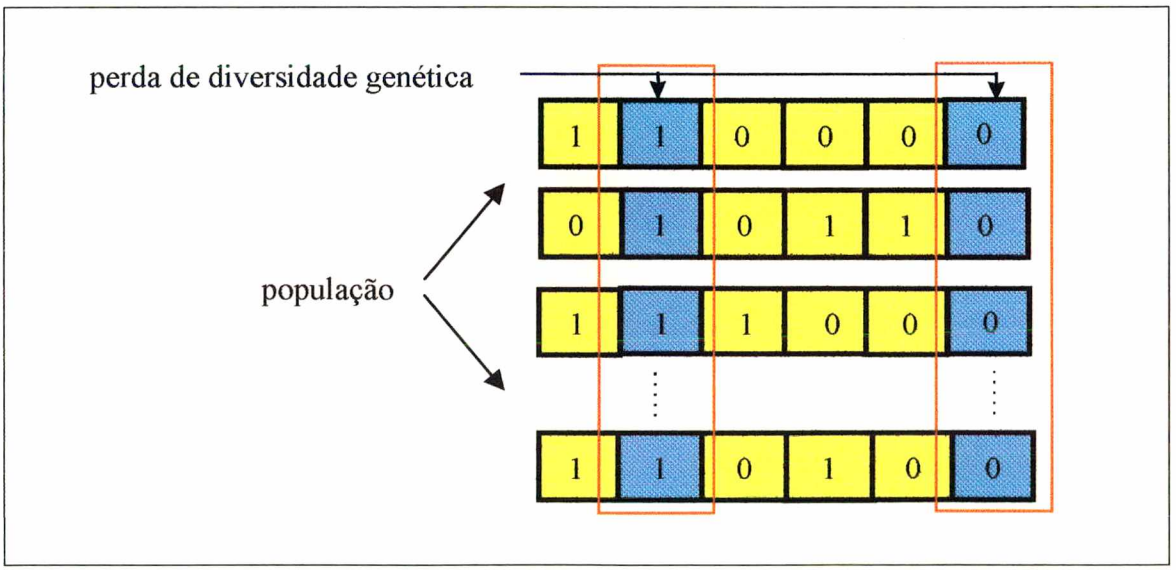

Figura 4. 3- Perda de diversidade em alguns genes dos indivíduos da população

Conforme a equação 4.1 o valor de diversidade pode variar entre 0 e 1. Para D = 1, há diversidade em todos os detetores. Para D = 0, todas as *strings* são iguais.

4.7 Operadores Genéticos Básicos

Os operadores de seleção, *crossover* e mutação, citados no *SGA* (seção 4.5), são considerados operadores fundamentais pois estão presentes em qualquer algoritmo genético. Os mecanismos de seleção, *crossover* e mutação em codificação binária são extremamente simples, exigindo apenas o cálculo de número aleatórios, cópias de *strings* e alguma troca parcial de *bits* entre estas.

4.7.1 Seleção

A seleção é o operador que determina quais *strings* serão escolhidas para posterior troca de *bits* a partir do valor do *fitness*. A seleção pode ser implementada de várias formas, no entanto, os melhores indivíduos sempre devem possuir maior possibilidade de serem selecionados em relação aos piores, permitindo que os genes dos melhores passem preferencialmente às próximas gerações.

O operador de seleção utilizado no *SGA* é a *rolleta*. Inicialmente, este operador atribui para cada indivíduo da população uma probabilidade de seleção proporcional ao valor do seu *fitness*. Para transformar o *fitness* em probabilidade de seleção simplesmente calcula-se o somatório do *fitness* de todos os elementos e efetua-se uma normalização. O *fitness*, no mais simples dos casos, é igualado a função objetivo, porém, em muitos situações isto não é possível (ver seção 4.12 que trata do mapeamento do valor da função objetivo para o *fitness*), ou mesmo o mais adequado a ser feito (ver seção 4.9.1 sobre o operador de escalonamento que manipula o valor do *fitness* para aumentar a eficiência do algoritmo genético). Em seguida, escolhe-se aleatoriamente um indivíduo da população, sendo que, a chance de cada um depende de sua probabilidade de seleção. Como a *rolleta* só seleciona um indivíduo por vez, este operador deve ser aplicado um número de vezes igual ao número de indivíduos da população. Devido à natureza

estocástica da *rolleta*, os elementos mais adaptados da população nem sempre serão selecionados, no entanto, é mais provável que o sejam. Uma ilustração da *rolleta*, para uma população de quatro indivíduos, pode ser vista na figura 4.4

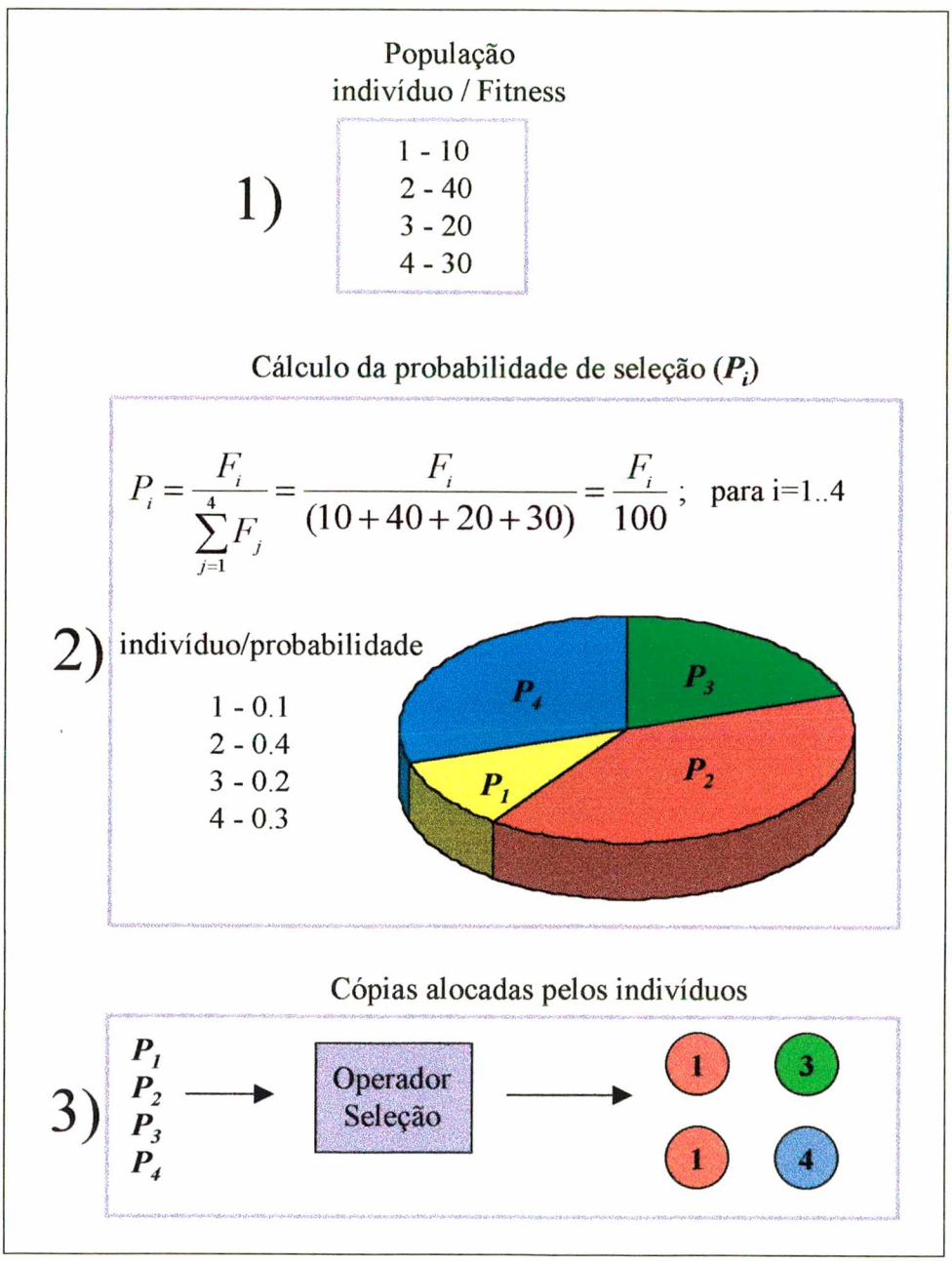

Figura 4. 4 Ilustração do funcionamento do *rolleta* em um população de 4 indivíduos

(1) A cada indivíduo é atribuído um *fitness* baseado no valor da função objetivo.

(2) Calcula-se a probabilidade de seleção de cada indivíduo.

(3) Cópias dos indivíduos são alocadas baseadas na probabilidade de seleção.

Outro operador de seleção bastante utilizado é o *tournament*. Este escolhe um grupo de N indivíduos aleatoriamente, sendo que, cada indivíduo da população possui igual probabilidade de ser sorteado. No entanto, é selecionado o indivíduo de *fitness* mais alto entre os indivíduos do grupo escolhido. Dessa forma indivíduos com *fitness* mais elevado terão maior possibilidade de serem selecionados. O *tournament* deve ser aplicado duas vezes para determinar dois indivíduos para a posterior utilização do *crossover*. O parâmetro N pode ser variado entre 1 e N. Se N é igual a 1, todos os indivíduos da população possuem a mesma probabilidade de seleção, não importando o valor do seu *fitness*, sendo que, neste caso tem-se uma seleção completamente aleatória. Se N é o tamanho da população, o melhor indivíduo dessa sempre será selecionado. Estes são os casos extremos, porém, normalmente se utiliza um valor de $N = 2$.

4.7.2 *Crossover*

O operador de *crossover* é responsável pela geração de novos indivíduos a partir da combinação dos indivíduos selecionados. Para decidir se o *crossover* deve ou não ser aplicado, efetua-se um sorteio cujo resultado dependerá do valor de um parâmetro conhecido por **taxa de *crossover***. Quando se utilizam valores próximos a um para a taxa de *crossover* tem-se, na maior parte dos casos, a geração de filhos cujos genes são obtidos a partir da combinações dos pais. No outro extremo, para uma taxa de *crossover* próxima a zero, os filhos serão, em sua grande maioria, idênticos aos pais. Em geral, utilizam-se valores altos para este parâmetro, como por exemplo 0.95, pois valores pequenos tendem a diminuir a eficiência do algoritmo genético sem melhorar os resultados finais da otimização.

O *crossover* utilizado no *SGA* é conhecido por *crossover 1-ponto*. A partir da seleção de dois pais, o *crossover 1- ponto* sorteia uma posição ao longo do comprimento das *strings* e troca os *bits* entre estas após o ponto de *crossover*. Ver figura 4.5.

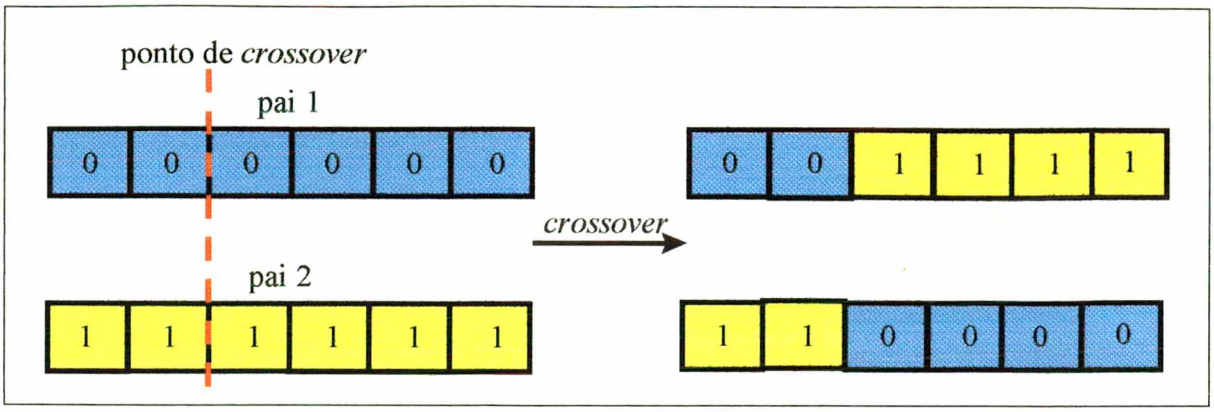

Figura 4. 5- Ilustração do funcionamento do *crossover 1-ponto*

Outro operador de *crossover* bastante conhecido é chamado de *crossover uniforme*. Este operador gera apenas um filho a partir de dois pais, sendo que, utiliza uma probabilidade de 50% de que qualquer *bit* no filho provenha do primeiro pai, caso contrário o *bit* provém do segundo pai. Uma variação deste operador, conhecida por *crossover parametrizado uniforme,* possui um parâmetro que determina a probabilidade de que qualquer *bit* no filho seja herdado do primeiro pai. A definição deste parâmetro em 1 resulta em um filho igual ao primeiro pai, e definindo-o em 0 resulta um filho idêntico ao segundo pai. Este parâmetro pode aceitar qualquer valor no intervalo [0,1] para obter o *grau* de mistura desejado entre os pais.

4.7.3 Mutação

Mutação é um operador que tem por objetivo evitar a perda de diversidade genética da população. A mutação é necessária ao algoritmo genético pois a utilização apenas do *crossover* e seleção pode resultar na igualdade de um ou mais detetores em todos os indivíduos da população. Nesta situação, caso o valor contido no detetor que perdeu diversidade não corresponda ao ótimo, não será possível mudá-lo por *crossover* pois todas as *strings* possuem o mesmo valor no detetor (figura 4.3), e qualquer cruzamento efetuado resultará na transferência da mesma carga genética dos pais para os filhos.

O operador de mutação utilizado no *SGA* é a mutação *uniforme*, sendo que, cada gene recebe uma probabilidade (também chamada de **taxa de mutação**) para que esse operador seja aplicado. Dessa forma, deve-se percorrer a cada geração todos os *bits* de todas as *strings* da população, e para cada um deles decidir, por sorteio, se a mutação deve ou não ser aplicada. Em caso afirmativo, troca-se o valor do *bit* (figura 4.6). Normalmente, valores de taxa de mutação de 1% são adequados para a maioria dos problemas. Deve-se tomar cuidado para não utilizar a taxa de mutação elevada demais, transformando o algoritmo genético em um método enumerativo.

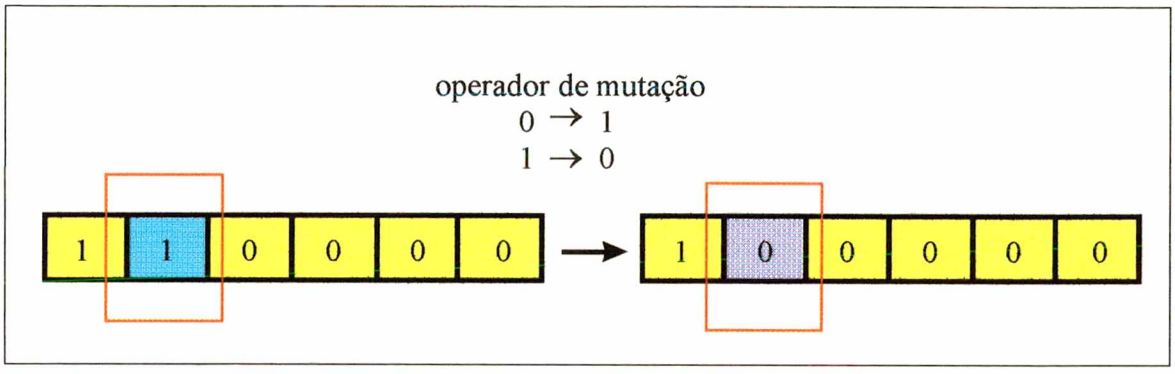

Figura 4. 6- Mutação ocorrida em um gene da *string*

A mutação é um operador secundário (tem pequena probabilidade de ocorrer), mas permite o melhoramento da população, sem perda de diversidade, quando utilizada de forma adequada conjuntamente com os operadores de *crossover* e seleção.

4.8 Efeito dos Operadores Genéticos Básicos

• A utilização de seleção isoladamente tenderá a preencher a população com cópias do melhor indivíduo, que não é necessariamente a solução do problema.

• O utilização apenas dos operadores de seleção e *crossover* poderá levar a convergência prematura do algoritmo genético.

• O emprego de mutação exclusivamente induz um procedimento enumerativo na busca do ótimo.

4.9 Outros Operadores

Nesta seção serão descritos os operadores de escalonamento e elitismo. Estes são alguns dos operadores opcionais que podem ser utilizados para aumentar a eficiência e robustez de um algoritmo genético.

4.9.1 Escalonamento

O escalonamento é um operador que tem por função manter a competição entre os indivíduos de uma população a níveis aceitáveis. O escalonamento pode ser utilizado conjuntamente com operadores que atribuem a probabilidade de seleção dos indivíduos de forma proporcional ao *fitness*, tal como a *rolleta*.

No início de uma otimização aparecem, freqüentemente, indivíduos muito superiores ao restante da população (super-indivíduos). Quando se utiliza a seleção por *rolleta*, os super-indivíduos assumem uma significativa proporção da população em poucas gerações. Isto é indesejável e pode levar a uma convergência prematura. No outro extremo, durante a otimização ou próximo ao seu final, os *fitness* do pior, médio e do melhor indivíduos podem estar muito próximos. Caso não seja aplicada nenhuma medida de controle dos níveis de competição, os elementos ruins, médios e melhores alocarão o mesmo número de cópias a partir deste momento. Dessa forma a busca pelo ótimo se transformará em processo enumerativo. Em ambos os casos, no começo e no final da otimização, o escalonamento do *fitness* pode melhorar o desempenho do

algoritmo genético. Uma representação do operador de escalonamento pode ser visto nas figuras 4.7 e 4.8.

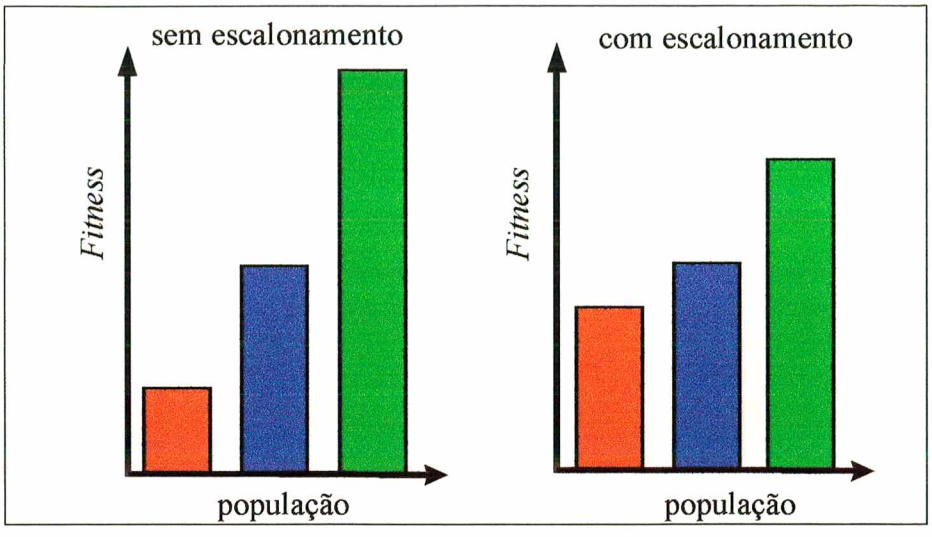

Figura 4. 7 – Modificação do *fitness* efetuada pelo escalonamento no início da otimização
(Evita a convergência prematura)

A figura 4.7 mostra a ação do operador de escalonamento em uma população que contém um super-indivíduo. O *fitness* de todos os indivíduos abaixo da média é aumentado e o *fitness* de todos os superiores a média, incluindo o super-indivíduo, é diminuído.

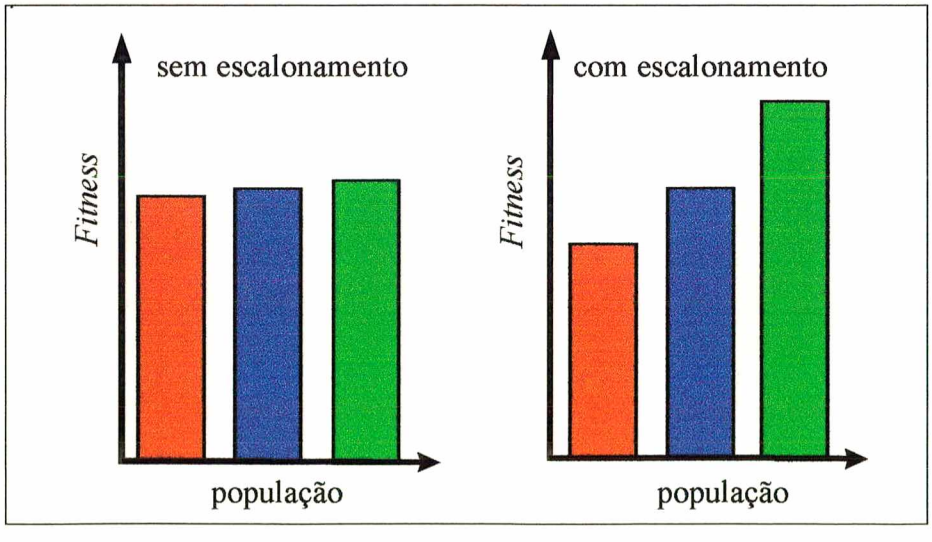

Figura 4. 8 - Modificação do *fitness* efetuada pelo escalonamento no final da otimização
(Evita que a otimização se torne uma busca enumerativa)

Na figura 4.8, percebe-se que o *fitness* do menor, médio e máximo estão muito próximos. O escalonamento age aumentando as diferenças, de modo que o *fitness* dos menores que a média é diminuído e o dos maiores que a média é aumentado.

Serão descritos quatro tipos de escalonamento:

- *Linear*;
- *Truncamento Sigma*;
- *Lei de potência*;
- Procedimentos de *ranking*.

Escalonamento linear: Definindo o *fitness* natural f e o *fitness* escalonado F. O escalonamento *linear* relaciona F a f de acordo com a equação 4.2:

$$F = a.f + b \quad (4.2)$$

Os coeficientes a e b podem ser escolhidos de diversas formas, entretanto, o *fitness* escalonado médio F_{medio} é normalmente igualado ao *fitness* natural médio f_{medio} (equação 4.3).

$$F_{medio} = f_{medio} \quad (4.3)$$

Para calcular os dois coeficientes da equação 4.2 deve-se definir outra relação. A mais utilizada é a seguinte:

$$F_{max} = C_{mult}.f_{medio} \quad (4.4)$$

Onde: C_{mult} é um valor constante maior que *um*;

F_{max} é o *fitness* escalonado do melhor indivíduo da população.

Para populações pequenas (50 a 100 indivíduos) C_{mult} = 1.2 a 2 tem sido utilizado com sucesso. Uma ilustração do escalonamento *linear* pode ser vista na figura 4.9.

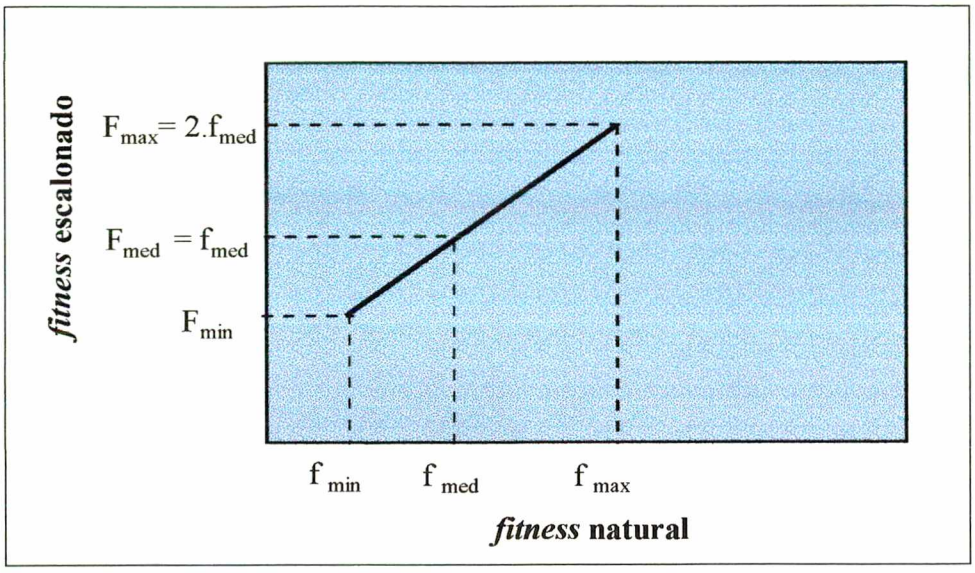

Figura 4. 9 –Ilustração do funcionamento do escalonamento *linear*

Próximo ao final da otimização, os indivíduos da população podem ter *fitness* muito semelhantes. Nesta situação, o escalonamento aumenta a diferença entre o *fitness* do pior, médio e do melhor de modo a evitar que a otimização se torne pesquisa enumerativa. Isto, por sua vez, dificulta a aplicação do escalonamento *linear* quando algumas *strings* estão bem abaixo da média da população, porém, o *fitness* médio está próximo do máximo. Se o escalonamento é aplicado nesta situação o alargamento requerido para separar os médios dos bons fará que as *strings* ruins tenham valores negativos após o escalonamento. Quando ocorre esta situação, mantém-se o *fitness* natural médio igual ao *fitness* escalonado médio, porém mapeia-se o mínimo *fitness* natural para o *fitness* escalonado = 0. Dessa forma os parâmetros contidos no escalonamento *linear* deverão ser calculados a partir das relações (4.5) e (4.6).

$$F_{medio} = f_{medio} \qquad (4.5)$$

$$F_{min} = 0 \qquad (4.6)$$

Onde: F_{min} = é o *fitness* do pior indivíduo da população.

O escalonamento *linear* pode não funcionar adequadamente quando há diferenças significativas entre o *fitness* do pior, médio e melhor indivíduos da população, ou seja, nas seguintes situações: quando o *fitness* do menor está bem abaixo da médio mas o *fitness* médio está próximo do máximo ou, de forma contrária, quando o *fitness* do menor está próximo da médio mas o *fitness* do médio está distante do melhor. O primeiro caso, já descrito acima, é resolvido modificando as relações utilizadas para o cálculo dos coeficientes da equação do escalonamento. Esta medida evita a atribuição de *fitness* negativos aos piores indivíduos. No entanto, o escalonamento *linear* atribuirá um valor de *fitness* para os indivíduos próximos a média praticamente igual ao do melhor. No segundo caso, o *fitness* escalonado do pior ficará ainda mais próximo do médio. Nos dois casos percebe-se que o escalonamento *linear* tem dificuldades de explicitar, de modo adequado, a diferença entre os indivíduos. Este comportamento tende a diminuir a eficiência do algoritmo genético pois um *fitness* praticamente igual é atribuído a indivíduos que possuem desempenhos diferentes.

Truncamento sigma: Neste operador utiliza-se a variância da população para transformar o *fitness*, equação 4.7.

$$F = \max\left[f - (f_{med} - c.\sigma), 0\right] \quad (4.7)$$

Onde: F = *fitness* escalonado;

f = *fitness* natural;

f_{med} = *fitness* médio da população;

c = constante; normalmente entre 1 e 3;

σ = variância da população;

Max = O maior entre $f - (f_{med} - c.\sigma)$ e 0.

Escalonamento com lei de potência: Neste escalonamento, o *fitness* escalonado é tomado como o *fitness* natural elevado a alguma especificada potência, equação 4.8.

$$F = f^k \quad (4.8)$$

Escalonamento bilinear: O escalonamento *bilinear*, proposto neste trabalho, é semelhante ao *linear*. A diferença está na utilização de uma equação linear para escalonar o *fitness* dos indivíduos superiores a média e outra para os indivíduos inferiores (figura 4.10). Esta modificação tem por objetivo minimizar os problemas que o escalonamento *linear* pode apresentar quando há diferenças significativas entre o desempenho do pior, médio e melhor indivíduos da população.

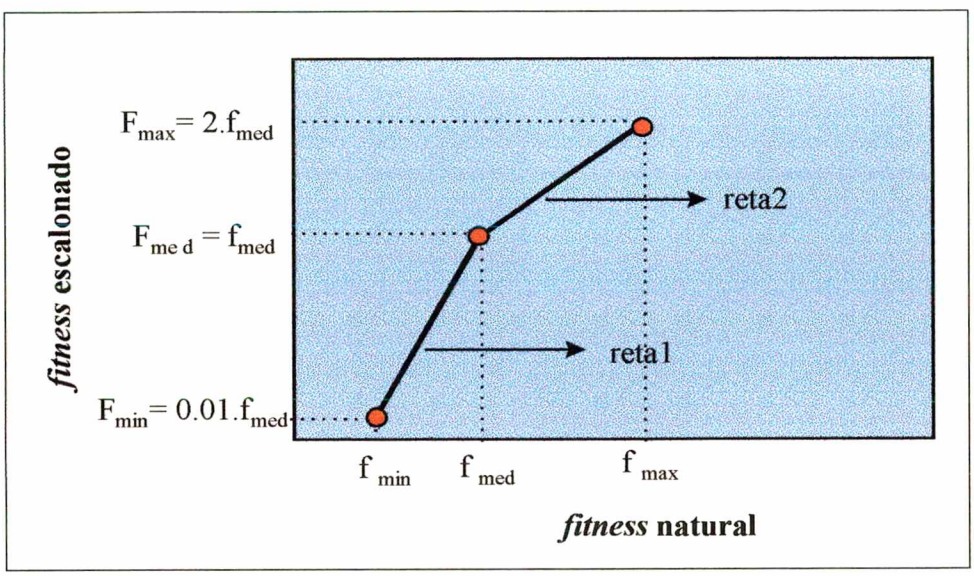

Figura 4. 10- Ilustração do funcionamento do escalonamento *bilinear*

De modo análogo ao *Linear* calculam-se os coeficientes da equação de escalonamento (4.2) a partir das relações (4.3) e (4.4). A equação 4.4 deve ser utilizada para o escalonamento do *fitness* dos indivíduos com desempenho superior ao médio. Para escalonar o *fitness* dos indivíduos com desempenho inferior ao médio utilizam-se as equações (4.10), (4.11) e (4.12):

$$F = a''.f + b'' \quad (4.10)$$

$$F_{med} = f_{med} \quad (4.11)$$

$$F_{min} = C_{min}.f_{medio} \quad (4.12)$$

Onde: C_{min} é o número de cópias esperadas para o pior indivíduo da população. Um valor de C_{min} em torno de 0.01 forneceu bons resultados em testes preliminares.

Um valor normalmente utilizado para a constante k é de 1.005. Entretanto, o valor de k depende do problema a ser otimizado, e deveria ser modificado durante a corrida para alongar ou encurtar a diferença entre os *fitness* dos indivíduos da população.

Procedimentos de Ranking: Neste tipo de escalonamento a população é ordenada com base no valor da função objetivo no sentido do melhor para o pior (*ranking*). O valor do *fitness* é atribuído em função da posição no *ranking* e, portanto, indiretamente a partir da função objetivo. O melhor indivíduo recebe o valor mais alto e a medida em que se desce o *ranking* o *fitness* atribuído diminui. As duas distribuições mais utilizadas na atribuição do *fitness* em função da posição do *ranking* são:

Distribuição linear: Neste caso fixa-se o valor do *fitness* para o melhor e pior elementos da população. Os elementos intermediários recebem valores entre o máximo e o mínimo respeitando a sua posição no *ranking*;

Distribuição Exponencial: Para esta distribuição fixa-se o valor do *fitness* do melhor elemento da população e atribui-se um fator de decaimento dado pela equação 4.9.

$$\frac{P_{i+1}}{P_i} = k \quad (4.9)$$

Onde: P_i é o i-ésimo melhor elemento do ranking e P_{i+1} é o i-ésimo + 1;

k é uma constante entre 0 e 1, normalmente 0.9, que dá o formato exponencial a esta distribuição.

O valor absoluto de *fitness* atribuído ao melhor e pior elementos da população na lei de decaimento linear e para o melhor elemento na lei exponencial não é importante, já que o *fitness* de cada elemento da população será normalizado posteriormente para ser convertido em probabilidade de seleção.

4.9.2 Elitismo

O elitismo é o operador que repassa os melhores indivíduos da população de uma geração para a próxima sem que estes sofram qualquer tipo de alteração.

No algoritmo genético, um excelente indivíduo pode aparecer na população em determinada geração e ser perdido devido ao caráter estocástico do método. Neste caso, muitas gerações podem ser necessárias para que seja encontrado outro semelhante. Para não perder os melhores indivíduos da população em determinada geração, costuma-se repassá-los diretamente para a próxima geração sem que sejam alterados por *crossover* ou mutação. O número de indivíduos repassados deve ser uma pequena fração da população pois um elitismo elevado pode levar a convergência prematura do algoritmo genético. O valor recomendado está em torno de 5 a 10 % da população.

4.10 Modificação da Faixa de Interesse de Otimização

A decodificação das *strings* gera apenas valores inteiros não negativos. Porém, na maioria dos casos, esse não é o domínio de interesse da otimização e a precisão obtida é muito baixa. Para mapear o valor decodificado de cada *string* para o domínio de interesse efetua-se uma transformação linear conforme a equação 4.13.

$$X = \frac{U_{\max} - U_{\min}}{2^N - 1} . I + U_{\min} \quad (4.13)$$

Onde:

$[U_{minimo}, U_{máximo}]$ são os limites inferior e superior da variável de interesse;

I - é o inteiro sem sinal obtido da decodificação das *strings*;

X - variável real limitada a faixa de interesse;

N - número de *bits* na *string*.

Assim, admitindo que o tamanho das *strings* é N=4 a transformação linear resultará.

Decodificação		Transformação
0 0 0 0 →	0	→ U_{min}
1 1 1 1 →	2^4	→ U_{max}
outros valores são mapeados linearmente		
entre estes limites U_{min} e U_{max}		

Deve-se notar que o termo $\pi = \frac{U_{max} - U_{min}}{2^N - 1}$ na equação 4.13 é a precisão e portanto, pode-se controlar a faixa de operação e a precisão da variável pela manipulação de U_{max}, U_{min} e N.

4.11 Otimização de Problemas Multivariáveis

Para resolver problemas com múltiplas variáveis pode-se simplesmente justapor todas as variáveis codificadas (*substrings*) em uma formando apenas uma *string*. Claro, cada variável terá seu próprio comprimento e valores de U_{min} e U_{max}.

Por exemplo, em um problema de otimização com 3 variáveis, onde cada uma delas é codificada como *substrings* de 4 *bits*, concatenam-se as *substrings* da seguinte forma:

grupo de *substrings*				*string*
0001	0101	1100	→	000101011100
U_1	U_2	U_3		

Onde: U_1, U_2, U_3 = variáveis 1,2 e 3 respectivamente.

4.12 Mapeamento da Função Objetivo para avaliação do *Fitness*

Quanto maior o valor da função objetivo de um indivíduo maior também deve ser o *fitness* atribuído. No entanto, conforme comentado, o *fitness* não pode, ou mesmo não deve em muitas situações, ser igualado a função objetivo. Como por exemplo o *fitness* não poderá ser igualado a função objetivo quando essa última assumir valores negativos pois o *fitness* representa uma medida de adaptação, e posteriormente será convertido em probabilidade. Para resolver este problema faz-se o *fitness* de cada indivíduo igual ao valor da função objetivo acrescido de um valor positivo fixo para toda população em uma geração, porém, esse pode ser alterado de uma geração para outra. Os indivíduos que mesmo assim ficarem com *fitness* negativo são mapeados para zero. Matematicamente esta operação é dada pela equação 4.14:

$$\begin{aligned} Fitness(X) &= u(X) + C_{min} \quad se \quad u(X) + C_{min} > 0 \\ &= 0 \qquad\qquad\quad se \quad u(X) + C_{min} \leq 0 \end{aligned} \tag{4.14}$$

Onde:

u(x) = valor da função objetivo;

C_{min} = Coeficiente positivo somado ao valor da função objetivo.

O coeficiente C_{min} é normalmente escolhido como o valor absoluto do elemento com menor *fitness* da população na corrente geração ou como uma função da variância da população.

4.13 Resolução de Problemas de Minimização

Conforme a descrição do operador de seleção, os indivíduos com maior valor de função objetivo são aqueles que possuem probabilidade de seleção mais elevada. Dessa forma a pesquisa do ótimo é direcionada, com maior intensidade, em torno destes indivíduos. Este procedimento

resulta numa maximização. Para resolver problemas de minimização, utilizando algoritmos genéticos, deve-se multiplicar a função custo por menos um. Esta operação transformará a minimização em uma maximização e não modificará a solução do problema. A transformação não garante que o *fitness* seja não negativo para todos os casos. Caso isso ocorra deve-se proceder de forma análoga ao exposto na seção 4.12.

4.14 Restrições

Muitos problemas práticos contém restrições que devem ser satisfeitas. Um caso típico é a otimização de um processo a partir de um modelo do mesmo. As restrições aparecem em função de limitações em equipamentos e condições de operações. Restrições são normalmente classificadas como relações de igualdade e desigualdade.

Restrições de igualdade aparecem em problemas que envolvam balanços globais de massa, energia etc. Por exemplo, na otimização de um problema relacionado a equilíbrio de fases só faz sentido pesquisar soluções em que o somatório da fração molar em cada fase seja unitário. Neste caso, se o balanço de massa é satisfeito, qualquer indivíduo é um candidato a satisfazer também as equações de equilíbrio. Para tratar de restrições de igualdade deve-se colocar um procedimento de validação diretamente no *crossover*. Assim evita-se que o algoritmo genético opere com populações de indivíduos inválidos, que no caso do problema de equilíbrio de fases citado equivale ao não fechamento dos balanços globais.

Problemas com restrições de desigualdade podem ser divididos em pouco e muito restritos: Para resolver problemas do primeiro tipo, basta fazer que qualquer indivíduo que viole uma restrição não receba nenhum *fitness*. No segundo tipo, o problema pode ter tantas restrições que encontrar uma solução possível pode ser tão difícil quanto encontrar o ótimo. A melhor maneira de lidar com este tipo de problema é penalizar as *strings* que violam as restrições, diminuindo seu *fitness*. Então um problema com restrições é transformado em um problema irrestrito pela associação de um custo ou penalidade por cada restrição violada. O método em

questão é conhecido como o método das penalizações. O objetivo deste método é fazer que as *strings* que violam as restrições sejam penalizadas, porém, seu *fitness* não deve ser zerado já que pode haver boas combinações de *bits* nessas.

Matematicamente um problema com restrições:

$$\text{minimizar } g(x)$$
$$\text{sujeito a } b_i(x) \geq 0 \text{ onde } i = 1,2...n$$

Pode ser transformado em outro onde um termo de penalidade é incluído na função objetivo, GOLDBERG (1989) [52]:

$$\text{minimizar } g(x) + r.\sum_{i=1}^{n}(\varphi \mid b_i(x) \mid)$$

onde: φ - função penalidade;

r - coeficiente de penalidade.

A função penalidade é normalmente assumida como uma função quadrática. O coeficiente *r* pode ser diferente para cada tipo de restrição, de forma que a violação de qualquer restrição resulte em um aumento do valor nominal da função custo.

5 ALGORITMOS GENÉTICOS EM CODIFICAÇÃO REAL

5.1 Introdução

No capítulo 4 foi descrito um algoritmo genético que utiliza codificação binária. Entretanto, este tipo de codificação pode gerar *strings* muito grandes na representação de problemas multivariáveis, sendo que, a aplicação dos operadores genéticos *bit* a *bit* demandará um elevado tempo computacional. Para minimizar este problema pode-se trabalhar diretamente com operadores para números reais. Além do menor comprimento de *strings*, a codificação real apresenta outras vantagens em relação à codificação binária:

- Na codificação real não há necessidade de conversões de *strings* para avaliação da função objetivo pois cada gene corresponde a uma variável. Em codificação binária, vários genes (*bits*) são utilizados para representar uma única variável;

- O limite de precisão da solução obtida em codificação real é o da precisão da máquina. Em codificação binária este limite é baseado no número de *bits* utilizados na representação das variáveis;

- A utilização de codificação real permite um maior controle em relação à ação dos operadores genéticos nas *strings*, pois cada gene representa uma variável. No caso da codificação binária, a aplicação dos operadores genéticos nas *strings* produz modificações nos fenótipos que são difíceis de serem previstas.

Pelas vantagens descritas da codificação real em relação à binária, e motivado pela aplicação de algoritmos genéticos em codificação real na resolução de problemas com grande dimensionalidade, como por exemplo o treinamento de redes neurais *feedforward* (MONTANA

(1995) [10]), este trabalho será focalizado no estudo e aplicação de operadores genéticos para problemas em codificação real.

Os operadores que devem ser alterados, em relação à codificação binária, são aqueles que trabalham a nível das *strings*, ou seja, a mutação e o *crossover*. O escalonamento e a seleção operam diretamente no *fitness* dos indivíduos e, portanto, serão os mesmos não importando o tipo de codificação.

Neste capítulo serão descritos os operadores de *crossover* e mutação para codificação real mais comumente encontrados na literatura (HERRERA *et alli* (1994) [53], POHLHEIM (1997) [54]). Algumas combinações destes operadores terão seu desempenhos comparados na otimização dos problemas propostos no próximo capítulo.

5.2 Crossover

O *crossover* em codificação binária e real apresentam as seguintes semelhanças:

- Quando ocorre um *crossover* só há combinação entre os mesmos genes nos diferentes pais;
- A probabilidade de um par de indivíduos selecionados sofrer *crossover* depende de um parâmetro chamado de taxa de *crossover*.

Como diferenças principais podem-se citar:

- Os genes em codificação binária são os *bits* e o genes em codificação real são as variáveis do problema;
- O *crossover* em codificação binária se resume à cópia e troca de *bits*. No entanto, em codificação real há maior flexibilidade na definição de operações nos genes.

Entre os operadores de *crossover* em codificação real mais comuns encontram-se: *crossover discreto, plano, aritmético* e *intermediário*.

Crossover discreto: Neste tipo de *crossover*, dois filhos são formados a partir de dois pais selecionados. Para cada gene presente no cromossomo dos pais faz-se um sorteio entre duas opções com igual probabilidade. Se do sorteio for obtida a primeira opção, copia-se o valor do gene contido no segundo pai para o primeiro filho e o valor contido no primeiro pai para o segundo filho. Na segundo caso, efetua-se o mesmo procedimento porém, troca-se a ordem dos pais. Uma ilustração do funcionamento do *crossover discreto* pode ser vista na figura 5.1.

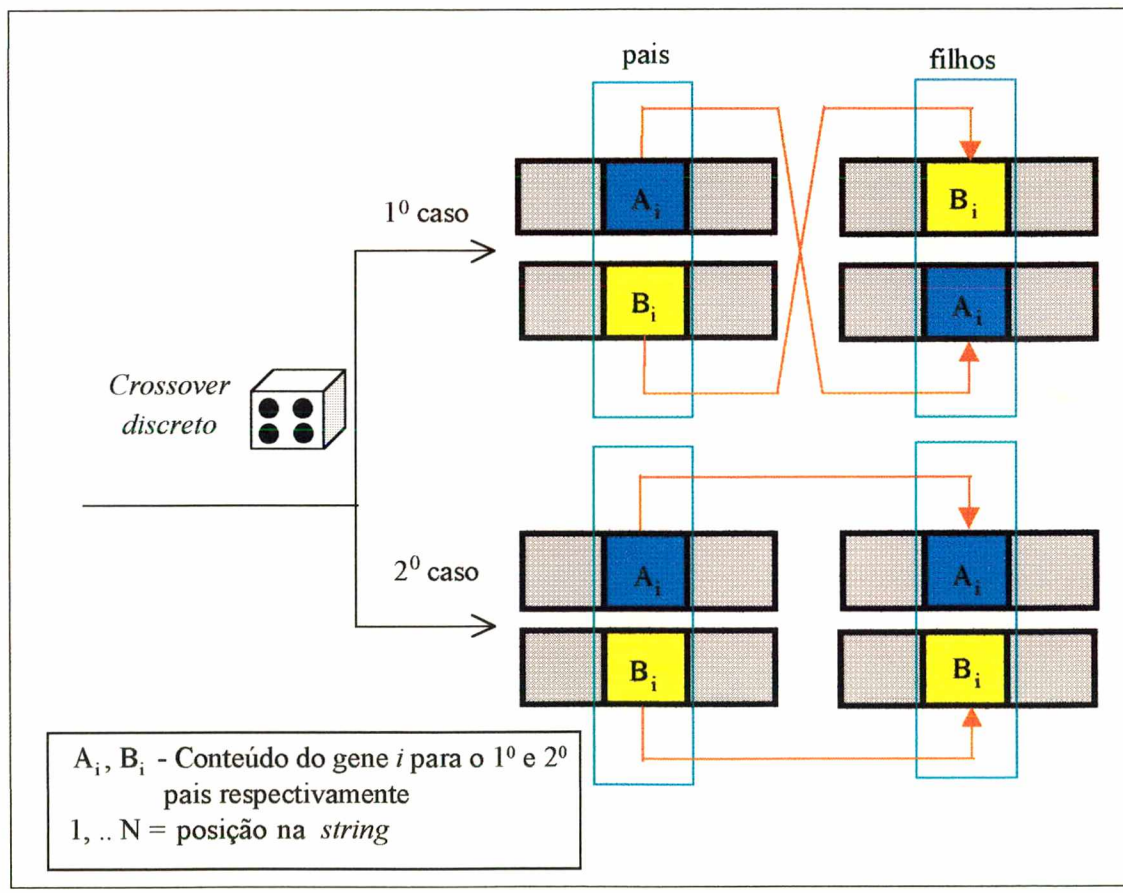

Figura 5. 1 - Ilustração do funcionamento do *crossover discreto* para um gene de duas *strings* selecionadas

Crossover plano: Neste tipo de *crossover* gera-se apenas um filho a cada dois pais selecionados. O valor atribuído a cada gene do filho será um valor aleatório contido no intervalo delimitado pelo respectivo gene no cromossomos dos pais. Ver figura 5.2

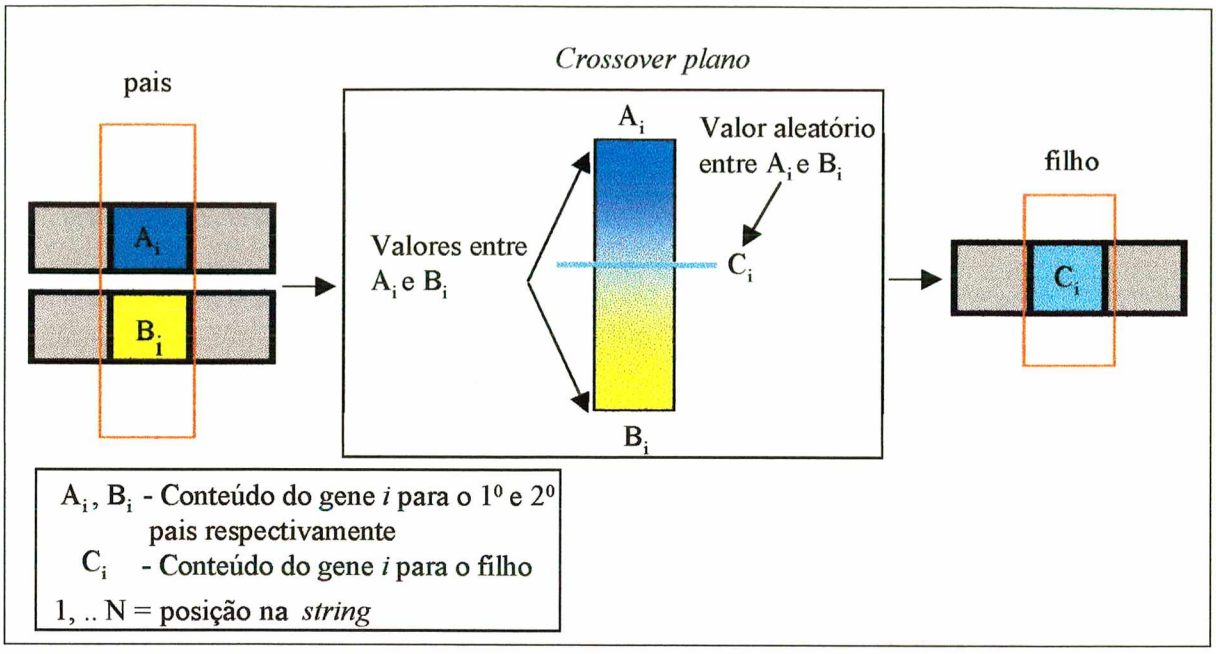

Figura 5. 2 – Ilustração do funcionamento do *crossover plano*

A atribuição do valor de um gene a um filho, dentro do intervalo delimitado pelo respectivo gene nos pais selecionados, é um procedimento comum a vários operadores de *crossover*, porém, implementado de forma diferenciada em cada caso. Este procedimento será de agora em diante citado no texto com o nome de *interpolação*.

Crossover aritmético: No *crossover aritmético* gera-se dois filhos a cada dois pais selecionados. Para cada gene no cromossomo dos pais, sorteia-se um valor entre 0 e 1 para o parâmetro α e, aplica-se esse valor nas equações (5.1) e (5.2).

$$\text{Filho\_1[i]} = \text{pai\_1[i]} + \alpha\,(\,\text{pai\_2[i]} - \text{pai\_1[i]}) \quad (5.1)$$

$$\text{Filho\_2[i]} = \text{pai\_2[i]} + \alpha\,(\,\text{pai\_1[i]} - \text{pai\_2[i]}) \quad (5.2)$$

Onde:

i = posição na *string* variando entre [1, comprimento da *string*];

Filho_1[i], filho_2[i] = Valor do i-ésimo gene no primeiro e segundo filhos;

Pai_1[i], Pai_2[i] = Valor do i-ésimo gene no primeiro e segundo pais.

Analisando as equações (5.1) e (5.2), percebe-se que o *crossover aritmético* gera genes aleatórios porém, simétricos entre os genes dos pais selecionados. Portanto, o *crossover aritmético* pratica uma *interpolação*, ou seja:

Para α=0 $\Rightarrow$ filho_1[i] = pai_1[i] e filho_2[i] = pai_2[i].

Para α=1 $\Rightarrow$ filho_1[i] = pai_2[i] e filho_2[i] = pai_1[i].

Para α em [0,1] $\Rightarrow$ filho_1[i] e filho_2[i] recebem valores aleatórios simétricos entre pai_1[i] e pai_2[i].

Uma ilustração do funcionamento do *crossover aritmético* pode ser vista na figura 5.3.

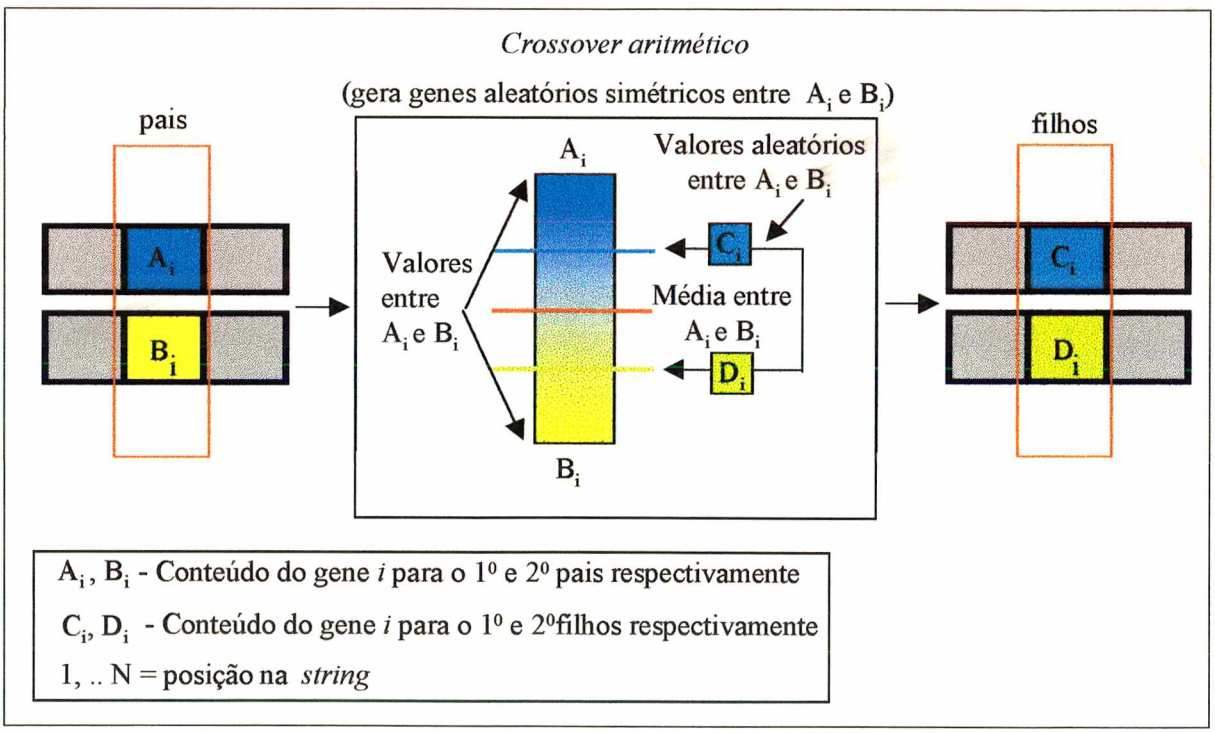

Figura 5. 3 - Ilustração do funcionamento do *crossover aritmético*

Como comentado, o *crossover aritmético* também implementa a *interpolação* porém, utiliza um mecanismo diferente do *crossover plano*.

Crossover intermediário: Neste *crossover* cada gene de um filho é gerado de acordo com a equação 5.3:

$$Filho[i] = pai\_1[i] + \alpha\,(\,pai\_2[i] - pai\_1[i]) \quad (5.3)$$

A nomenclatura utilizada na equação 5.3 é a mesma das equações (5.1) e (5.2).

A principal diferença entre o *crossover aritmético* e o *intermediário* é que no último, o valor α pode assumir valores menores que zero ou maiores que um. O valor α pode ser escolhido aleatoriamente no intervalo [-d,1+d], sendo que, normalmente utiliza-se d = 0.25. Neste caso α poderia variar entre [-0.25, 1.25].Quando α assume um valor entre 0 e 1 o *crossover intermediário* gera o filho através de uma *interpolação*. Para α menor que zero ou maior que um atribui-se a cada gene do filho um valor que está fora do intervalo delimitado pelo gene dos pais que o geraram (*extrapolação*).

Uma ilustração do *crossover intermediário* pode ser vista na figura 5.4.

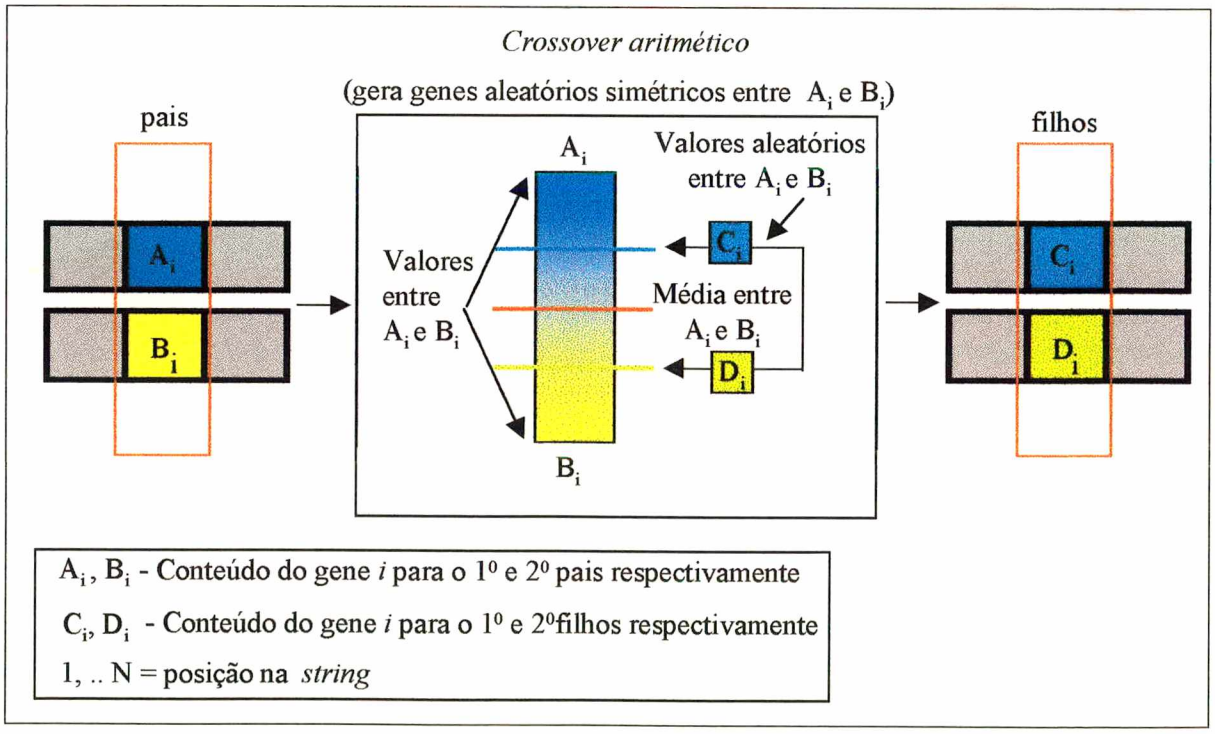

Figura 5. 4 - Ilustração do funcionamento do *crossover intermediário*

O *crossover intermediário* pode produzir soluções da otimização fora da faixa de interesse, principalmente nas primeiras gerações, devido à grande diversidade da população. Para

evitar a divergência do algoritmo genético deve ser utilizado, conjuntamente com o *crossover intermediário*, um procedimento para controlar a *extrapolação*. A figura 5.5 mostra uma ilustração sobre o funcionamento do procedimento para controlar a *extrapolação*.

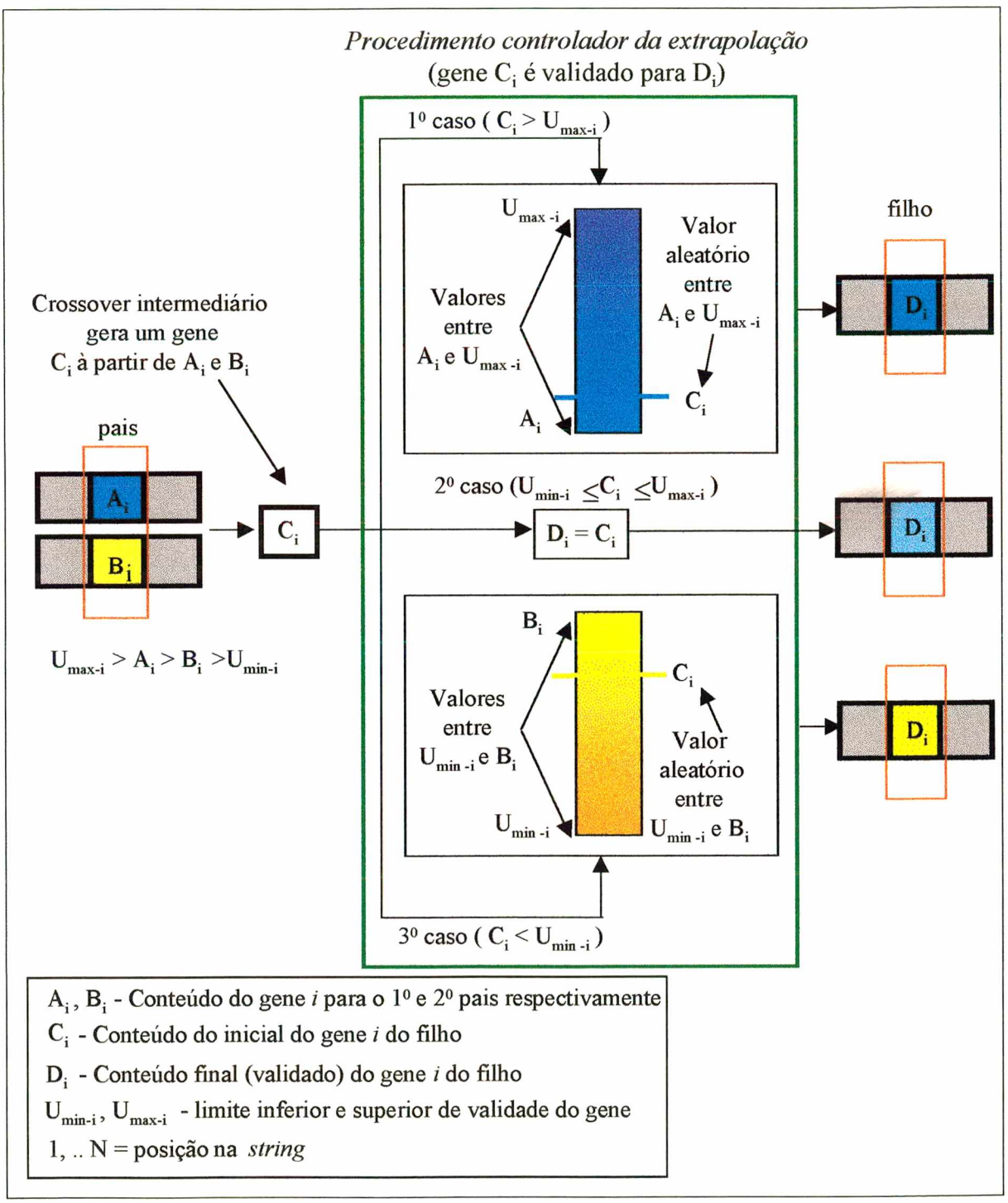

Figura 5. 5 – Procedimento para controlar a *extrapolação*

O procedimento para controlar a *extrapolação* verifica se os genes da *string* do filho gerado estão dentro da faixa de interesse da otimização, se estiverem, nenhuma correção é aplicada à *string*, caso contrário, cada gene que ultrapassou a faixa de otimização recebe um novo valor entre o limite ultrapassado (inferior ou superior) e o respectivo valor do gene contido no pai mais próximo.

5.3 Mutação

A mutação em codificação binária e real apresentam as seguintes características em comum:

- O operador de mutação será aplicado ou não através de um sorterio a ser feito, separadamente, para cada gene de todos os indivídos da população;

- A probabilidade de um gene sofrer mutação depende de um parâmetro conhecido por taxa de mutação.

As diferenças entre a mutação em codificação binária e real são as mesmas citadas para a mudança de codificação no caso do operador de *crossover*.

Entre os operadores de mutação em codificação real mais comuns encontram-se: a mutação *uniforme*, *por deslocamento* e a *exponencial*.

Mutação uniforme: Este operador substitui o conteúdo de cada gene em que atua por um valor aleatório dentro da faixa de otimização. Para evitar que a otimização se torne um procedimento enumerativo atribui-se uma probabilidade pequena para a taxa de mutação (normalmente 1%). Uma ilustração do funcionamento da mutação *uniforme* pode ser vista na figura 5.6.

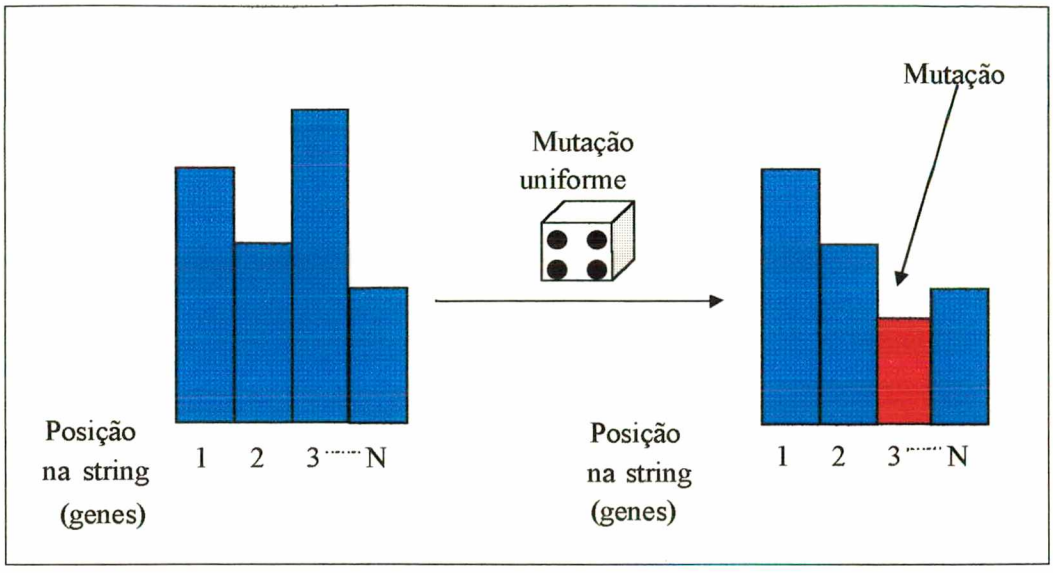

Figura 5. 6 – Ilustração do funcionamento da mutação *uniforme*

Mutação por deslocamento: Neste tipo de mutação, soma-se a cada posição da *string* um pequeno valor. O valor adicionado deve ser diferente para cada posição da *string*, podendo ser calculado da seguinte forma: defini-se uma valor limite para cada gene, normalmente um porcentagem da faixa de interesse da otimização, e multiplica-se por um número aleatório com distribuição uniforme entre 0 e 1. O sinal deste valor também deve ser sorteado com a mesma probabilidade de ser positivo ou negativo (Ver figura 5.7). Neste operador assume-se um valor alto para a taxa de mutação, normalmente 1, o que significa que cada gene sofrerá mutação. No entanto, os genes são apenas ligeiramente modificados pela aplicação da mutação por *deslocamento*.

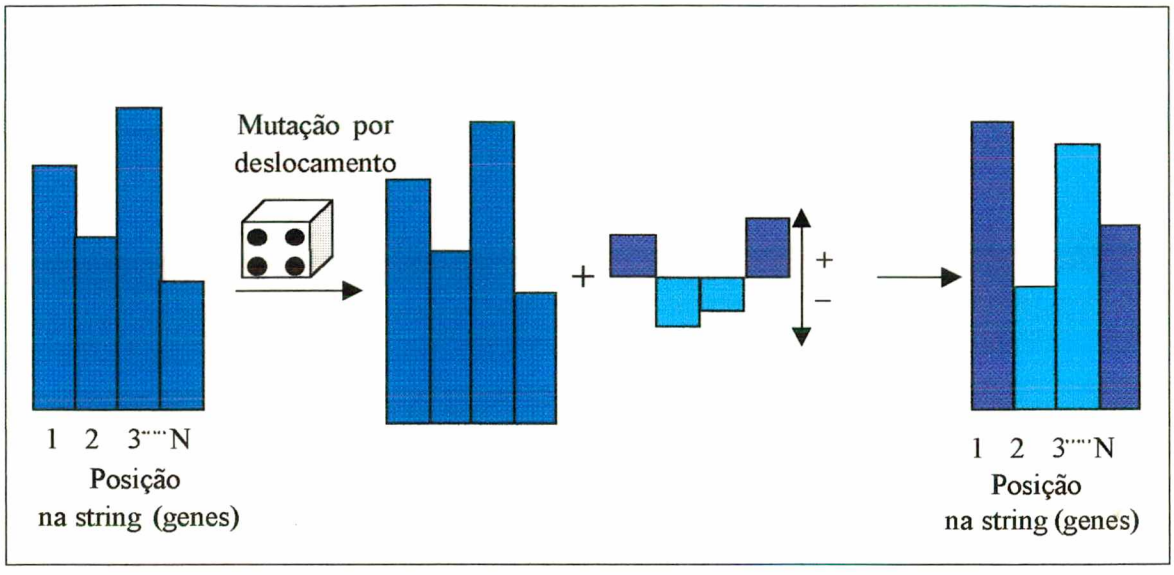

Figura 5. 7 - Ilustração do funcionamento da mutação *por deslocamento*

Mutação exponencial: Uma variante da mutação por *deslocamento* é a mutação *exponencial*. A diferença básica é a forma como se calcula o valor a ser adicionado a cada gene. No caso da mutação *exponencial*, sorteia-se para cada gene um valor a partir de uma distribuição de probabilidade exponencial com média zero. Dessa forma tem-se uma maior possibilidade de somar valores pequenos e uma menor possibilidade de somar valores elevados a cada gene que sofre a mutação. Deve-se utilizar uma taxa de mutação pequena para evitar pesquisa enumerativa.

5.4 Fluxograma do Algoritmo Genético em Codificação Real

Um esquema do algoritmo genético em codificação real pode ser visto na figura 5.8. Nesta nota-se, em relação ao *SGA* descrito na seção 4.5, a ausência do procedimento de decodificação de *strings*, a inclusão do escalonamento e do procedimento para controle da *extrapolação* (se for utilizado o *crossover intermediário*).

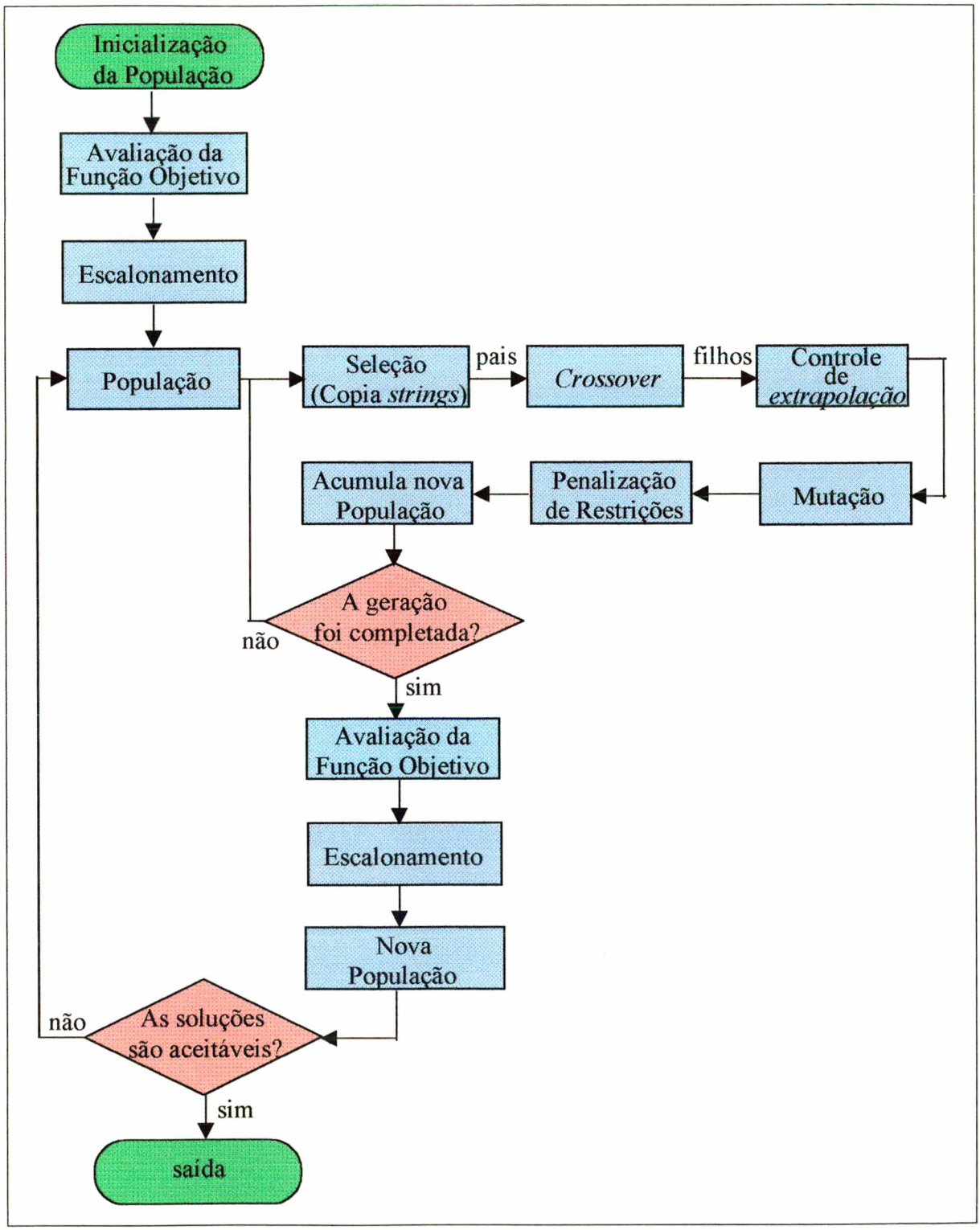

Figura 5. 8 – Fluxograma do algoritmo genético em codificação real

6 ANÁLISE DO DESEMPENHO DE ALGORITMOS GENÉTICOS

6.1 Introdução

O objetivo deste capítulo é determinar, entre os operadores descritos no capítulo 5 e alguns outros propostos, que conjunto resulta no melhor algoritmo genético. Inicialmente foi feito um estudo de várias combinações de operadores genéticos na otimização de alguns problemas. Devido ao grande número de combinações possíveis entre os operadores foram escolhidas apenas as mais relevantes. Baseado no estudo efetuado foram propostas algumas modificações nos operadores genéticos, sendo que, as modificações feitas foram testadas no treinamento de uma rede neural *feedforward*. Finalmente, o melhor algoritmo genético determinado foi comparado a um *software* de uso geral na resolução de alguns *benchmarks* de otimização.

6.2 Estudo de Operadores Genéticos

Para estudar a influência dos operadores genéticos foram escolhidos três problemas nos quais deseja-se determinar o mínimo global. Em cada caso o ótimo é conhecido e, portanto, pode-se analisar o desempenho de cada conjunto de operadores testados. Os problemas escolhidos, descritos abaixo, são: uma *função multimodal*, uma *função descontínua* e um problema de *ajuste de parâmetros*.

Função multimodal: Este problema tem por objetivo verificar a capacidade dos algoritmos genéticos em determinar o ótimo em uma região onde existem muitos mínimos locais. Uma função que apresenta tal característica é a descrita pela equação 6.1.

$$f(X) = \sum_{i=1}^{4} (0.1 \cdot |x_i| + |\text{sen}(x_i)|) \quad (6.1)$$

$x_i \in [-10,10]$ para i =1, .. 4.

Os valores das variáveis que otimizam o problema são X = [0, 0, 0, 0] e o valor do ótimo global é $f(X) = 0$. Na figura 6.1, pode ser vista uma versão unidimensional da função multimodal. A função apresenta comportamento semelhante nas outras dimensões e portanto, o número de mínimos locais cresce acentuadamente em relação ao aumento do número de dimensões. Além do elevado número de mínimos locais, a derivada é descontínua em cada um desses pontos.

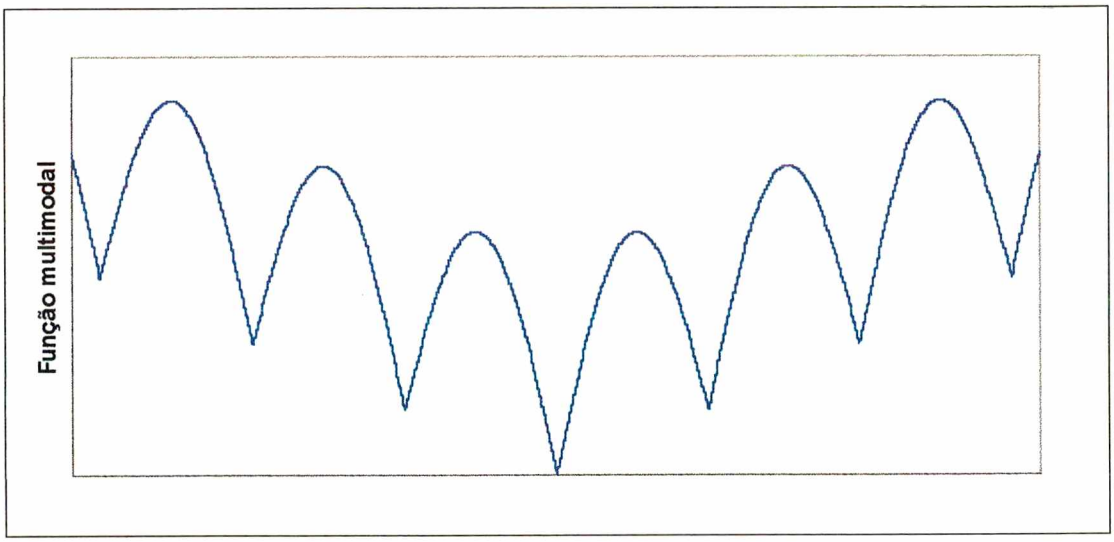

Figura 6. 1 -Versão unidimensional da função multimodal

Função descontínua: Funções descontínuas não podem ser otimizadas com métodos baseados em derivadas. Para testar a capacidade dos algoritmos genéticos em lidar com tal dificuldade foi utilizada a função dada pela equação 6.2.

$$f(X) = \sum_{i=1}^{4} (\text{inteiro}(x_i) + 0.1 \cdot x_i) \quad (6.2)$$

$x_i \in [-8,8]$ para i=1 até 4

O ótimo global localiza-se no ponto X = [-8, -8, -8, -8] com $f(X)$ = -35.6. Na figura 6.2 pode ser vista uma versão unidimensional da função descontínua.

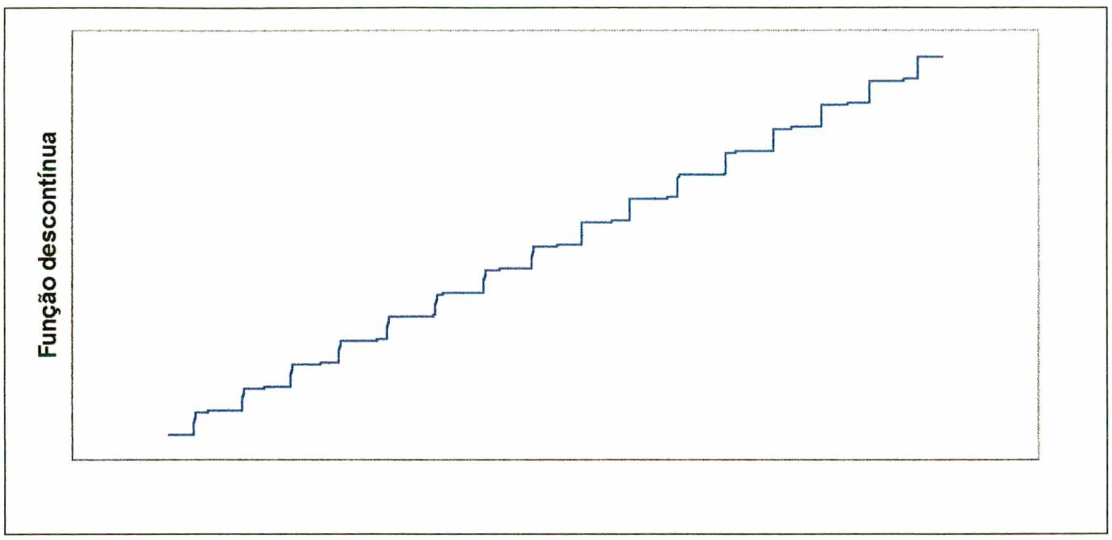

Figura 6. 2 -Versão unidimensional da função descontínua

Ajuste de parâmetros: O objetivo deste problema é verificar se os algoritmos genéticos conseguem evoluir em superfícies praticamente planas. O problema escolhido consiste na determinação dos parâmetros de uma cúbica que melhor se ajusta a um grupo de dados gerado também por uma cúbica.

Neste problema, gerou-se inicialmente um grupo de dados com 101 pontos utilizando-se a cúbica descrita na equação 6.3. A variável independente (x) foi definida como variável discreta no intervalo [0,4] com um passo de 0.04.

$$F = 0.5x^3 - 1x^2 - 3x + 5\text{, onde: x = 0, 0.04, 0.08 .... 4.} \quad (6.3)$$

A função objetivo (equação 6.4) foi definida como erro quadrático entre o grupo de dados e uma cúbica genérica (expressão 6.5).

$$H(P) = \sum_{i=1}^{NP} (G_i - F_i)^2 \qquad (6.4)$$

Onde:

N_p = número de pontos do grupo de dados;

P = parâmetros a serem determinados (a,b,c,d), sendo que, admitiu-se [-10,10] como intervalo de pesquisa para todos os parâmetros.

$$G = a.x^3 + b.x^2 + c.x + d \quad (6.5)$$

O erro quadrático de ajuste, em princípio, pode ser zerado pois o objetivo é ajustar uma cúbica a um grupo de dados obtido através da mesma forma funcional. Dessa forma os algoritmos genéticos deveriam recuperar os parâmetros originais dos quais foi gerado o grupo de dados. O ótimo do problema é portanto: P = [0.5,-1, -3, 5] e $H(P)$ =0.

Uma idéia do comportamento da função erro quadrático pode ser visto na figura 6.3. Nesta mostram-se três curvas em que o parâmetro c assumiu os valores de –3, -2, -1. Os parâmetros a e d foram fixados em 0.5 e 5 respectivamente e b adotou valores no intervalo[–2,0].

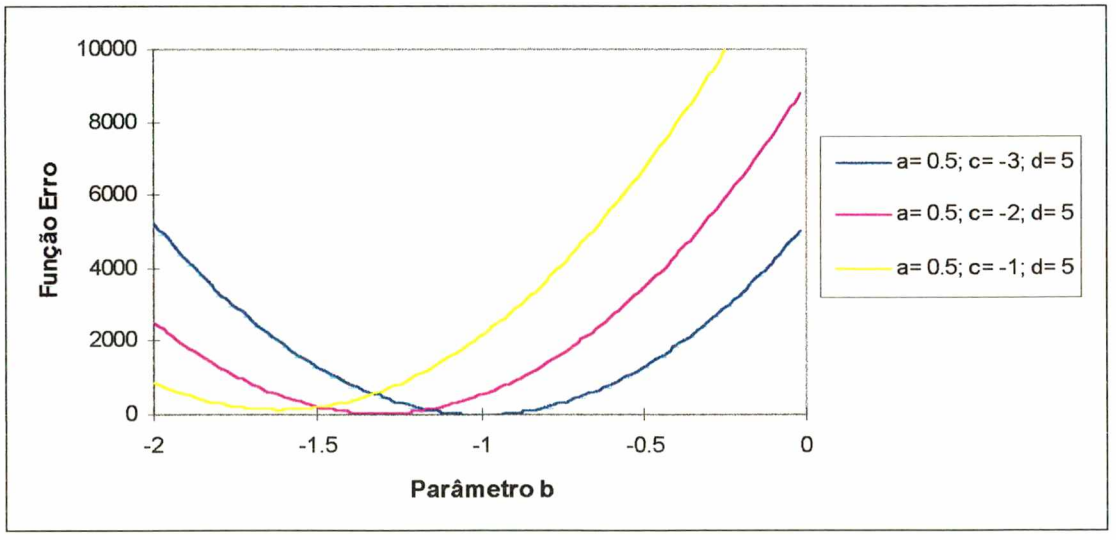

Figura 6. 3- Gráfico paramétrico da erro quadrático entre um grupo de dados e uma equação de ajuste cúbica

Observando a figura 6.3, nota-se que os locais onde o erro quadrático é pequeno correspondem aos pontos que o gradiente da curva é praticamente nulo, ou seja, há uma região praticamente plana na superfície do erro quadrático que passa pelo ótimo.

6.2.1 Combinações de Operadores Testadas

As combinações de operadores testadas nas otimizações são listadas a seguir:

Crossover plano* e *intermediário. Alguns tipos de *crossover*, tais como o *aritmético* e o *plano*, são bastante semelhantes pois efetuam apenas a *interpolação* entre os respectivos genes dos pais selecionados, diferindo ligeiramente na forma de efetuar tal operação. Entretanto, o *crossover intermediário* gera novos indivíduos utilizando tanto a *interpolação* como a *extrapolação* dos genes dos pais selecionados. Por esta razão foi escolhido somente um dos operadores que efetuam a *interpolação*, neste trabalho o *crossover plano*, para comparação com o *crossover intermediário*;

Mutação *exponencial* e *uniforme*. Os operadores de mutação *exponencial* e *por deslocamento* alteram o valor dos genes somando a cada uma desses um pequeno valor. No entanto, a mutação *uniforme* modifica o valor de cada gene em que atua aleatoriamente dentro da faixa de interesse da otimização. Todos estes operadores modificam aleatoriamente as *strings* em que atuam, no entanto, a mutação *exponencial* e a *por deslocamento* são mais semelhantes se comparadas à mutação *uniforme*. Por esta razão escolheu-se para comparação, neste trabalho, a mutação *exponencial* que altera pouco os genes e a mutação *uniforme* que modifica completamente os genes em que atua;

Escalonamento *linear* e *bilinear*. Embora o escalonamento seja independente do tipo de codificação empregada, este operador também foi testado. O escalonamento *bilinear* (proposto nesse trabalho) foi comparado com o *linear* (comumente mais utilizado).

6.2.2 Parâmetros do Algoritmo Genético

Nas otimizações efetuadas, os seguintes parâmetros do algoritmo genético foram empregados:

- População = 100 indivíduos;

- Taxas de *crossover* = 1. Valores pequenos utilizados em alguns testes preliminares não resultaram em aumento do desempenho do algoritmo genético e tornaram a convergência mais lenta. Por esta razão utilizou-se um valor de taxa de *crossover* igual 1 em todas as otimizações;

- Taxas de mutação diferentes em cada problema foram utilizadas para maximizar o desempenho dos algoritmos genéticos. Em cada caso será informada oportunamente a taxa de mutação utilizada;

- O elitismo = 4%;

- Parâmetro $\alpha \in [-0.25, 1.25]$. Nas otimizações com o *crossover intermediário*, permitiu-se ao parâmetro α variar no intervalo dado, conforme mencionado e indicado na seção 5.2;

6.2.3 Metodologia de Testes Empregada

Adotou-se a seguinte metodologia na determinação do melhor conjunto de operadores: Para cada problema descrito, testaram-se todas as combinações de *crossover-mutação-escalonamento* descritas na seção 6.2.1 para um número suficiente de gerações. Em cada caso,

foram efetuadas 10 otimizações, sendo que, utilizou-se a média dos desempenhos obtidos para análise e comparação dos vários conjuntos de operadores genéticos.

Os algoritmos genéticos obtidos das diferentes combinações de operadores foram analisados por dois indicadores calculados a partir do valor da função objetivo: O primeiro é a convergência, obtida da evolução do erro de otimização (valor absoluto da diferença entre o ótimo global e a função objetivo correspondente ao melhor indivíduo da população) em relação ao número de gerações. O segundo indicador é a diversidade da população, sendo que, propõem-se neste trabalho uma expressão para o cálculo da diversidade em codificação real.

A diversidade da população para problemas em codificação binária pode ser calculada facilmente, conforme descrito na seção 4.6. Em codificação real não há uma forma equivalente, no entanto, como os problemas testados possuem ótimo conhecido, pode-se calcular a diversidade utilizando esta informação.

A diversidade é um indicador dinâmico, varia de uma geração para outra, que informa a possibilidade da população atingir o ótimo. Dessa forma, pode-se obter uma estimativa da diversidade calculando a relação entre a dispersão da população, em termos de função objetivo, e a distância média dessa até o ótimo, também relativa a função objetivo. Matematicamente pode-se calcular a diversidade, para problemas de minimização, conforme a equação 6.6

$$D(n) = \frac{disp(n)}{Med(n) - Ótimo} \quad (6.6)$$

Onde:

n = geração;

D(n) = Diversidade; índice dinâmico;

Ótimo = Valor da função objetivo que otimiza o problema;

Med(n) = média dos valores da função objetivo da população na geração *n*;

disp(n) = alguma medida de dispersão de população na geração *n*.

Como medida de dispersão poderia ser definido o desvio padrão da população, no entanto, para obter valores normalizados de diversidade, será utilizada a diferença entre a média da população e o indivíduo com menor valor de função objetivo, ou seja:

$$D(n) = \frac{Med(n) - Min(n)}{Med(n) - Ótimo} \quad (6.7)$$

Analisando a expressão 6.7 observa-se que se a dispersão da população for pequena em relação a distância que separa a população do ótimo, dificilmente ou somente após muitas gerações, o ótimo será atingido. Neste caso tem-se pequena diversidade ou D→ 0. No outro extremo, se a dispersão da população for praticamente igual a distância que separa a média dessa até o ótimo tem-se uma grande possibilidade que o ótimo seja atingido em poucas gerações (alta diversidade ou D→ 1). Na figura 6.4 ilustra-se uma situação em que a diversidade é baixa e, na figura 6.5 um caso de diversidade alta.

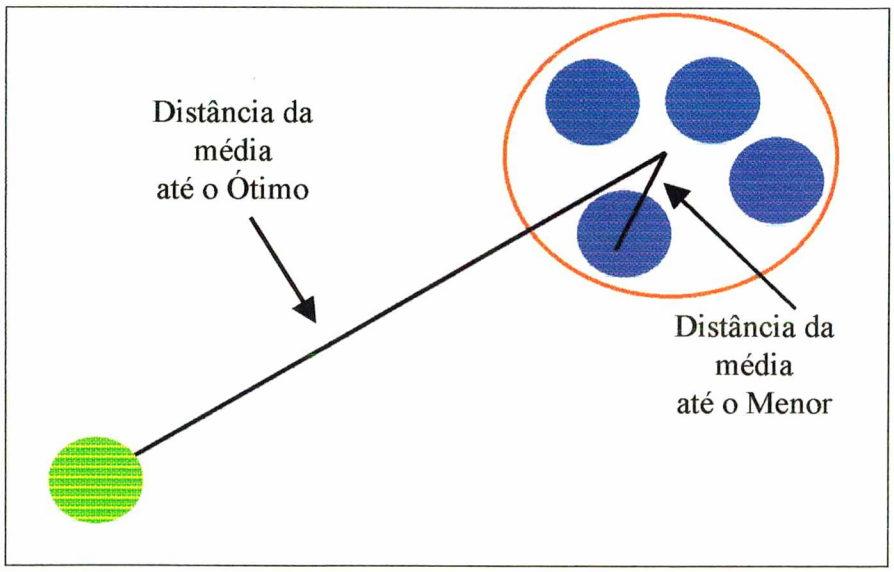

Figura 6. 4 – Ilustração de um caso em que a diversidade da população é baixa

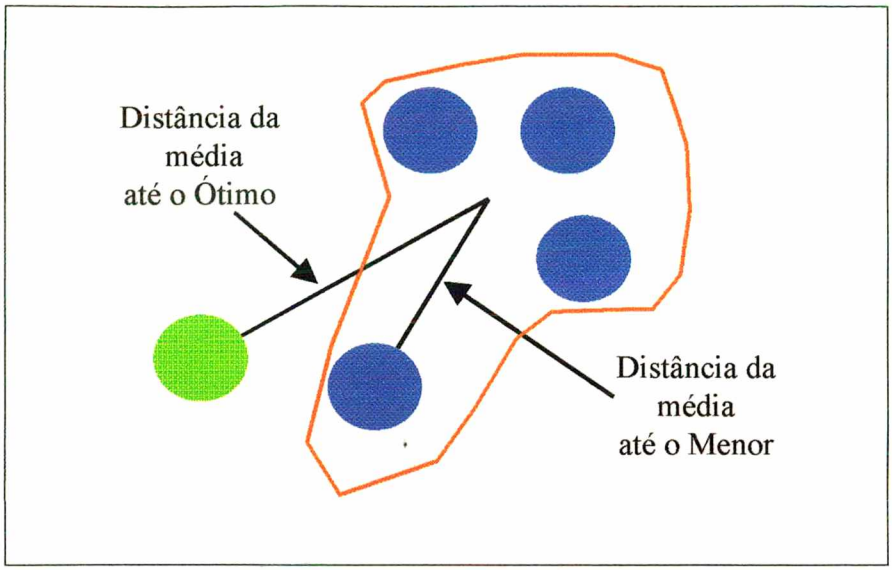

Figura 6. 5 - Ilustração de um caso em que a diversidade da população é alta

Aplicando um raciocínio análogo ao utilizado na obtenção da equação 6.7 chega-se a uma expressão para o cálculo da diversidade em um problema de maximização:

$$D(n) = \frac{Max(n) - Med(n)}{Otimo - Med(n)} \quad (6.8)$$

Onde: Max(n) = menor valor da função objetivo da população na geração *n*.

O indicador proposto para o cálculo da diversidade é empírico e, pode apresentar problemas em algumas situações:

• Mesmo que diferentes *strings* da população possuam um valor de função objetivo semelhante não significa que o conteúdo de seus genes também seja. Neste caso ainda pode haver grande diversidade na população, porém, o indicador que é calculado apenas a partir da função objetivo, afirmaria que a diversidade é pequena;

•Se a distância $|Otimo - Med| \rightarrow 0$ então para um problema de minimização tem-se que $|Otimo - Min| \rightarrow 0$. Dessa forma, pode ocorrer a divisão de dois números muito pequenos (indeterminação).

No primeiro caso pode-se pensar que se os algoritmos genéticos conseguem oferecer boas soluções para problemas complexos utilizando somente valores da função objetivo, então, espera-se que um índice calculado somente a partir da função objetivo também possa oferecer informações a respeito do comportamento do algoritmo genético.

O segundo caso pode ser minimizado computacionalmente através da utilização de blocos de exceção. Este recurso funciona da seguinte maneira: Um código que possa apresentar algum tipo de problema, tal como uma divisão por zero, é colocado dentro de um bloco de teste. Caso ocorra alguma irregularidade, a sequência de códigos que normalmente o programa efetua é interrompida e o programa passa a executar o código presente dentro do bloco de exceção.

Neste caso específico, o cálculo da diversidade deveria ser codificado computacionalmente da seguinte forma:

```
Experimente
        { Cálculo da Diversidade}
Exceção
        D = 0
```

Se o ótimo for atingido por toda população tem-se que $|Otimo - Med| \rightarrow 0$ e $|Otimo - Min| \rightarrow 0$, o que equivale a D→ 0. Nesta situação, ocorrerá uma divisão de dois números muito pequenos no cálculo do índice. Porém, como o cálculo da diversidade é efetuado dentro de um bloco de teste, o problema ocorrido fará com que o programa passe a executar os códigos dentro do manipulador da exceção, ou seja, D→0. Este é precisamente o valor que a diversidade deveria receber e portanto, a exceção foi manipulada corretamente.

Pode-se ainda argumentar que antes que a exceção seja disparada, pode haver erros numéricos envolvidos na divisão de dois números muito pequenos quando a população tende para o ótimo. Porém, o índice proposto será utilizado apenas para determinar qualitativamente o comportamento dos algoritmos genéticos.

Apesar dos problemas citados, cabe ressaltar os seguintes pontos sobre o indicador proposto:

- Fácil implementação computacional;
- Os resultados são intuitivos. O indicador pode variar entre 0 e 1, o que corresponde a baixa e alta diversidade da população respectivamente;
- Independência do problema a ser resolvido.

6.2.4 Resultados

Devido ao grande número de combinações possíveis de operadores e, pela influência do escalonamento e mutação ser menos relevante que o *crossover*, fez-se uma análise conjunta dos dois primeiros e outra para o operador de *crossover*. Cabe ressaltar que o escalonamento é um operador opcional e a mutação é um operador secundário conforme comentado anteriormente. Dessa forma, para analisar as combinações efetuadas fixou-se inicialmente o tipo de mutação e de escalonamento e, modificou-se o *crossover* para avaliar a influência desse operador. A seguir, fixou-se o *crossover* e alterou-se o escalonamento e a mutação.

1) Funcão multimodal: O resultado das simulações relativas à otimização da função multimodal pode ser visto na tabela 6.1. Esta mostra todas as combinações de operadores descritos na seção 6.2.1. Para cada caso, mostra-se o mínimo, a média e o máximo erros finais de otimização, além da taxa de mutação que maximizou o desempenho de cada algoritmo genético testado.

Tabela 6.1- Desempenho de vários algoritmos genéticos na otimização da função multimodal

				Erro de Otimização		
Crossover	Mutação	Taxa de Mutação	Escal.	Mínimo	Média	Máximo
plano	*uniforme*	0.005	*linear*	3.433E-06	5.559E-06	1.438E-05
plano	*exponencial*	0.005	*linear*	5.653E-07	1.682E-06	2.596E-06
plano	*uniforme*	0.015	*bilinear*	1.212E-09	7.977E-08	1.471E-06
plano	*exponencial*	0.002	*bilinear*	4.004E-14	1.489E-10	1.939E-06
intermediário	s. mutação	0.000	*linear*	2.549E-146	3.139E-145	3.80E-141
intermediário	s. mutação	0.000	*bilinear*	5.940E-206	5.069E-204	6.45E-201

A tabela 6.1 mostra claramente uma grande superioridade no desempenho dos algoritmos genéticos que utilizam o *crossover intermediário*. A tabela 6.1 fornece apenas os resultados finais das otimizações efetuadas para 2500 gerações. Uma análise mais detalhada, em função da média do erro de otimização e da diversidade, é mostrada a seguir.

* *Influência do operador de Crossover na convergência do Algoritmo Genético.*

Nas figuras 6.6 a 6.9 mostra-se a evolução do erro de otimização, em escala logarítmica, com os *crossover plano* e *intermediário* para várias combinações dos operadores de mutação e escalonamento. Em todos casos, percebe-se a grande superioridade do *crossover intermediário*, sendo que, a diferença de desempenhos entre os *crossover* cresce a medida que o número de gerações aumenta.

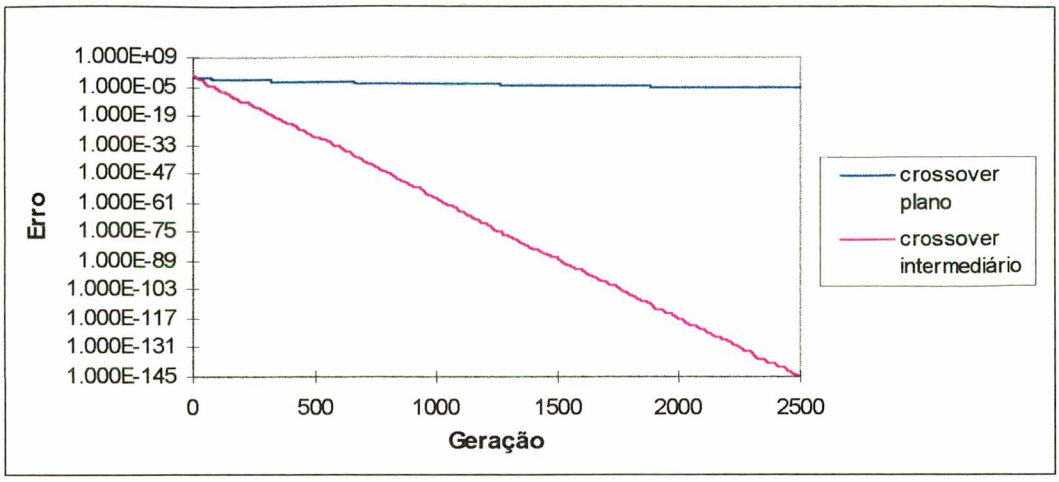

Figura 6. 6 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *linear* e mutação *uniforme*)

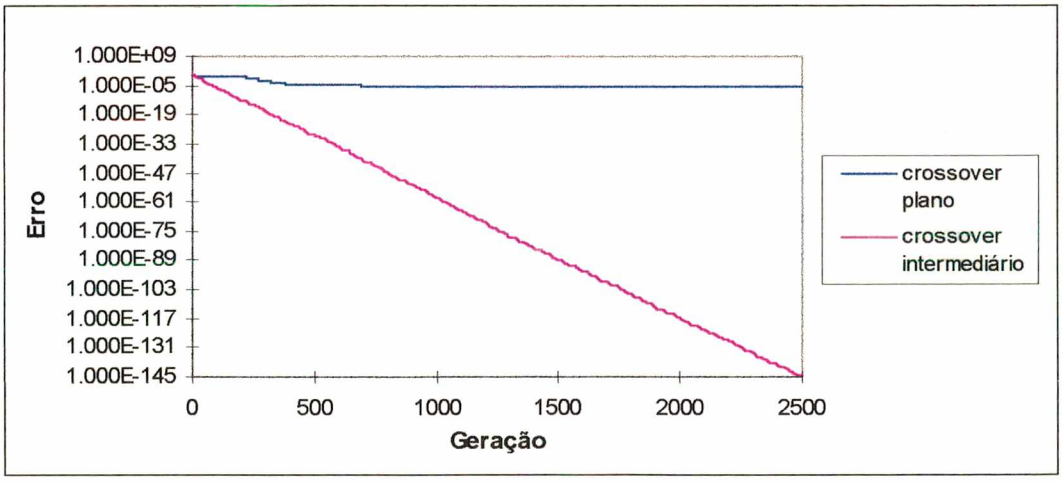

Figura 6. 7 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *linear* e mutação *exponencial*)

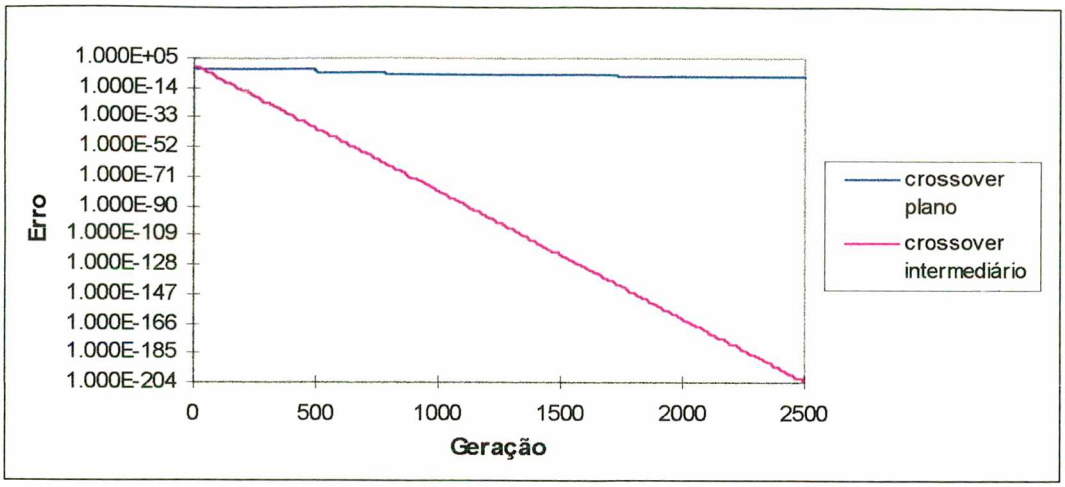

Figura 6. 8 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *bilinear* e mutação *uniforme*)

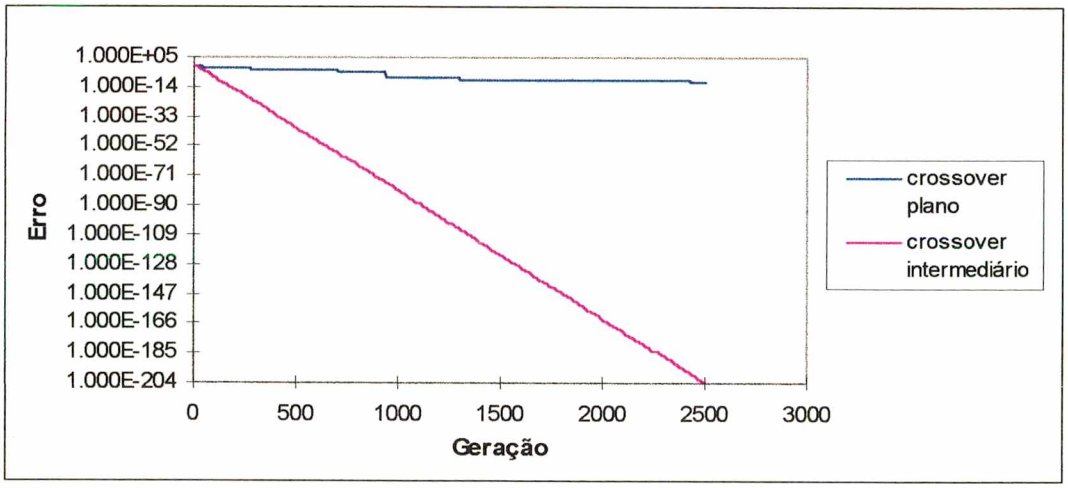

Figura 6. 9 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *bilinear* e mutação *exponencial*)

A convergência obtida com o *crossover intermediário* foi exponencial, efeito observado pela linha reta na escala logarítmica. O melhor resultado foi obtido sem mutação, ou seja, o *crossover intermediário* conseguiu manter a diversidade da população. No caso do *crossover plano*, o erro diminui lentamente e a evolução dos algoritmo genéticos, que utilizam este *crossover*, é bastante influenciada pela mutação.

* *Influência do operador de Mutação e Escalonamento na convergência dos Algoritmos Genéticos.*

Na figura 6.10 mostra-se a evolução do erro de otimização com o *crossover plano,* para as combinações dos operadores de mutação e escalonamento. Na figura 6.11, mostra-se a mesma curva para o *crossover intermediário.* Neste último caso observa-se apenas a influência do operador de escalonamento pois os melhores resultados foram obtidos sem mutação

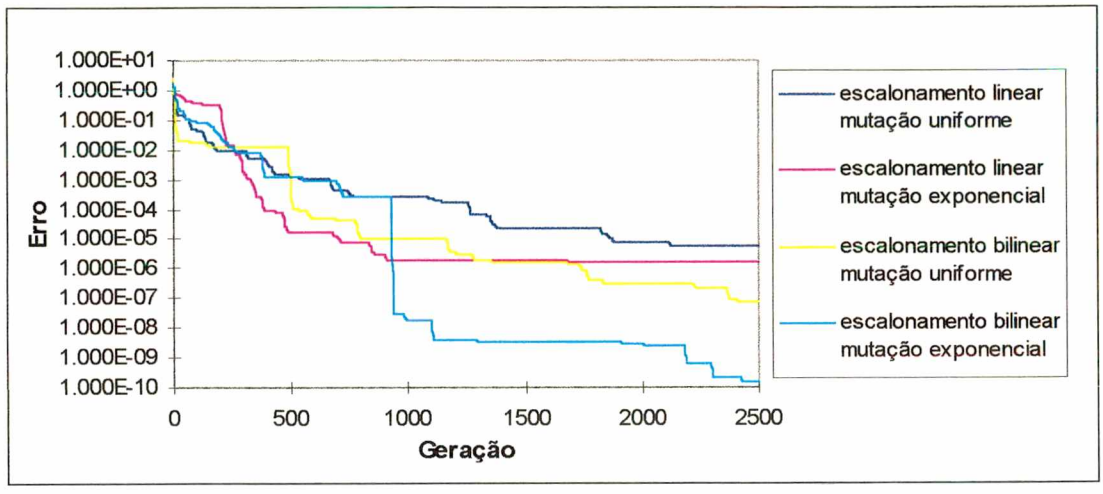

Figura 6. 10 - Influência dos operadores de mutação e escalonamento na convergência do algoritmo genético (*crossover plano*)

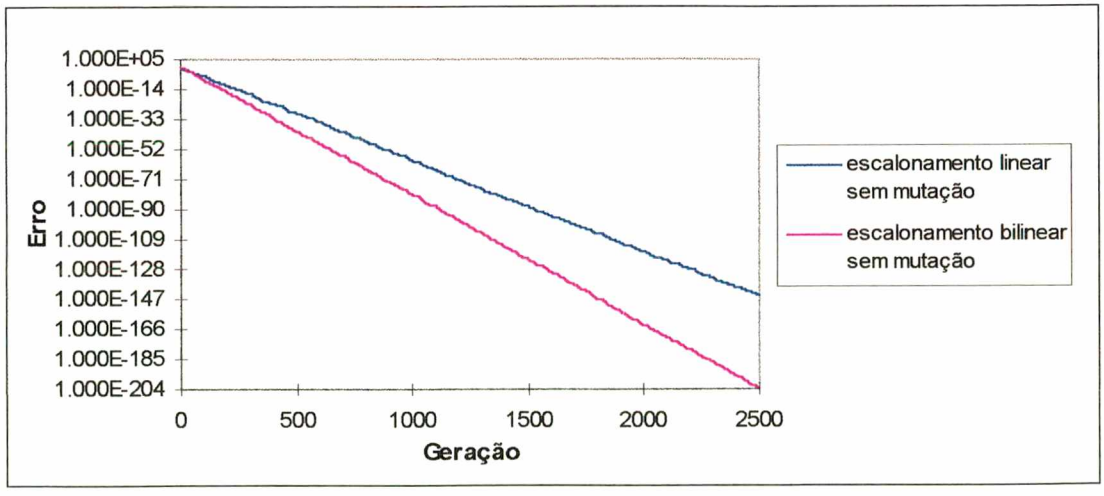

Figura 6. 11 - Influência do operador de escalonamento na convergência do algoritmo genético (*crossover intermediário*)

Analisando a figura 6.10, nota-se que para o *crossover plano* há um aumento da eficiência do algoritmo genético tanto pela utilização da mutação *exponencial*, quanto do escalonamento *bilinear*. Para o *crossover intermediário* (figura 6.11), a mutação apenas diminui a eficiência do algoritmo genético, que converge mais rápido para o ótimo sem a influência dessa. Os resultados obtidos da figura 6.10 mostram que, no caso da mutação, é melhor efetuar pequenas modificações nos genes (mutação *exponencial*) do que alterá-los bruscamente (mutação *uniforme*). Em relação ao escalonamento, como observado nas figuras 6.10 e 6.11, o *bilinear* foi superior mostrando sua melhor capacidade manter os níveis adequados de competição dentro da população

* *Influência dos operadores de crossover e mutação na diversidade da população*

A análise de diversidade foi baseada nos operadores de *crossover* e mutação pois esses alteram o valor dos genes e, portanto, modificam diretamente os valores da função objetivo dos indivíduos da população. Deve-se lembrar que o índice de diversidade é baseado na função objetivo. Outros operadores, como por exemplo o escalonamento e a seleção, poderiam ser considerados, no entanto, sua influência é indireta, sendo que, esses operadores estão associados a alocação de indivíduos para posterior aplicação do *crossover* e mutação.

O procedimento para a análise da diversidade foi realizado da seguinte forma: Para cada tipo de *crossover*, escolheu-se o conjunto de operadores (mutação, escalonamento) que gerou o algoritmo genético mais eficiente. Em seguida, calculou-se a diversidade da população para esse algoritmo, mantendo os parâmetros inalterados (taxa de mutação). O mesmo procedimento foi efetuado na avaliação da diversidade sem a utilização de mutação. Dessa forma pode-se avaliar separadamente a influência da *mutação* e do *crossover* na diversidade da população.

Na figura 6.12 mostra-se a evolução do índice de diversidade da população, com e sem mutação, para o algoritmo genético que utiliza o *crossover plano*, escalonamento *bilinear* e mutação *exponencial*. Este algoritmo foi o que obteve o menor erro médio de otimização utilizando o *crossover plano* (ver tabela 6.1).

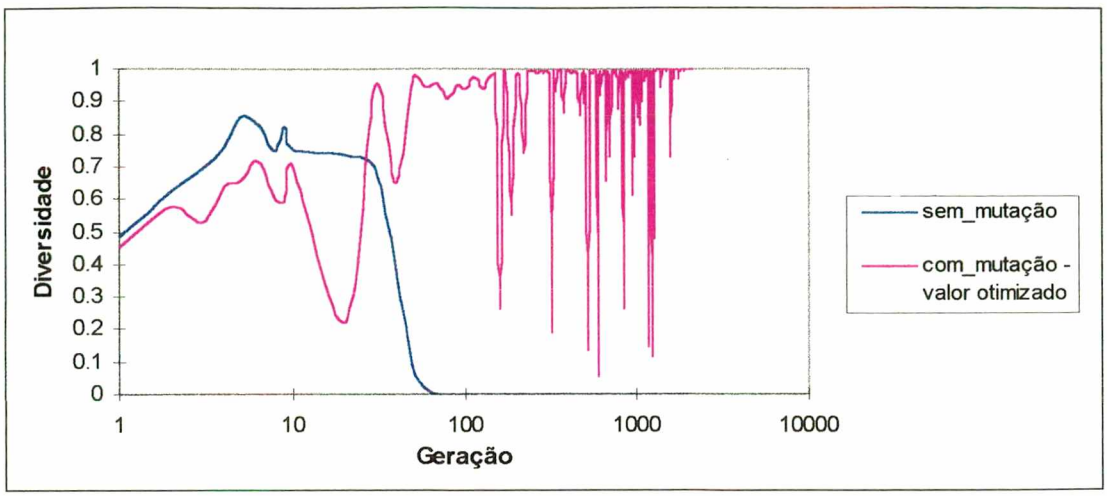

Figura 6. 12 – Influência do *crossover plano* na diversidade da população

Observando a figura 6.12 nota-se que as curvas de diversidade com e sem mutação são semelhantes até aproximadamente 20 gerações. A partir deste ponto, a diversidade da população da curva sem mutação diminui rapidamente e, de aproximadamente 80 gerações em diante, torna-se praticamente nula. Esta semelhança nas primeiras gerações é devido à diversidade inicial presente na população e a ação do operador de *crossover* (a mutação não é importante inicialmente). A partir de 80 gerações, as curvas possuem um comportamento completamente diferenciado devido à ação do operador de mutação. Um fato interessante a ser observado na curva com mutação acima de 100 gerações é a faixa de variação dos valores da diversidade, que alterna-se entre valores próximos a 0 e 1. Este efeito pode ser atribuído à combinação das ações da mutação e do *crossover*. A mutação aumenta a diversidade porém, o *crossover plano* apenas interpola e age em sentido contrário diminuindo a diversidade. Tem-se um jogo de forças que não resulta em eficiência na convergência pois, como pode ser visto nas figuras 6.6 a 6.9, o erro de otimização diminui lentamente.

A figura 6.13 mostra a diversidade para o algoritmo genético que utiliza o *crossover intermediário* e escalonamento *bilinear*. Neste caso mostra-se apenas a curva sem mutação que também corresponde ao melhor resultado obtido.

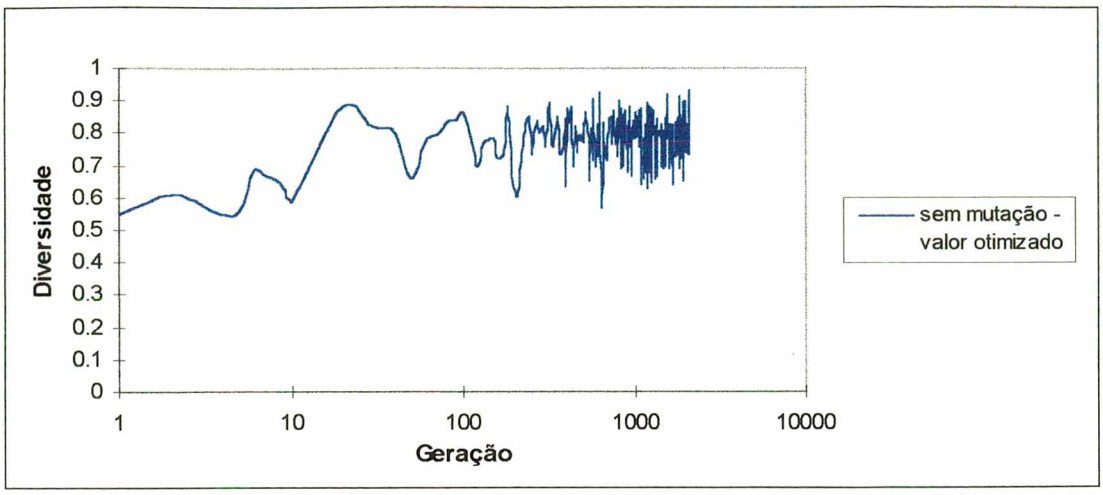

Figura 6. 13 - Influência do *crossover intermediário* na diversidade da população

Na figura 6.13 pode-se notar pela curva de diversidade que o *crossover intermediário* é capaz de manter a diversidade da população sem o auxílio da mutação. Obteve-se com este operador uma alta diversidade sem muita dispersão, aproximadamente entre 0.6 e 0.9. Isto significa uma diversidade relativamente constante da população ao longo de toda otimização.

2) Funcão descontínua: Na tabela 6.2 mostra-se o resultado das simulações relativas a otimização da função descontínua.

Tabela 6.2- Desempenho de vários operadores na otimização da função descontínua

				Erro de Otimização		
Crossover	Mutação	Taxa de Mutação	Escal.	Mínimo	Média	Máximo
plano	*uniforme*	0.002	*linear*	7.434E-03	1.619E-02	3.043E-02
plano	*exponencial*	0.003	*linear*	3.724E-04	4.580E-04	6.078E-04
plano	*uniforme*	0.005	*bilinear*	4.376E-03	6.514E-03	1.318E-02
plano	*exponencial*	0.002	*bilinear*	1.740E-04	4.101E-04	5.590E-04
Intermediário	*uniforme*	s. mutaçao	*binear*	7.11E-15	7.11E-15	7.11E-15
Intermediário	*exponencial*	s. mutaçao	*bilinear*	7.11E-15	7.11E-15	7.11E-15

A tabela 6.2 mostra, de forma semelhante ao obtido na otimização da função multimodal, uma grande superioridade no desempenho dos algoritmos genéticos que utilizam o *crossover intermediário*. Nota-se na tabela 6.2 que os erros finais de otimização relacionados ao *crossover*

intermediário são muito pequenos e correspondem a um mesmo valor (7.11E-15). Este pequeno erro observado é da ordem de grandeza do número de casas decimais utilizada representação numérica (16 casas decimais equivalentes a 1.0 e-16) e, portanto, a obtenção do mesmo valor em várias otimizações pode ser atribuída a problemas numéricos.

A tabela 6.2 fornece apenas os resultados finais das simulações até 1500 gerações. A seguir mostra-se um estudo detalhado do desempenho de cada combinação de operadores.

* *Influência do operador de Crossover na convergência do Algoritmo Genético.*

Nas figuras 6.14 a 6.17 mostra-se a evolução do log(erro) de otimização com os *crossover plano* e *intermediário* para várias combinações dos operadores de mutação e escalonamento. Em todos casos, verifica-se a superioridade do *crossover intermediário*.

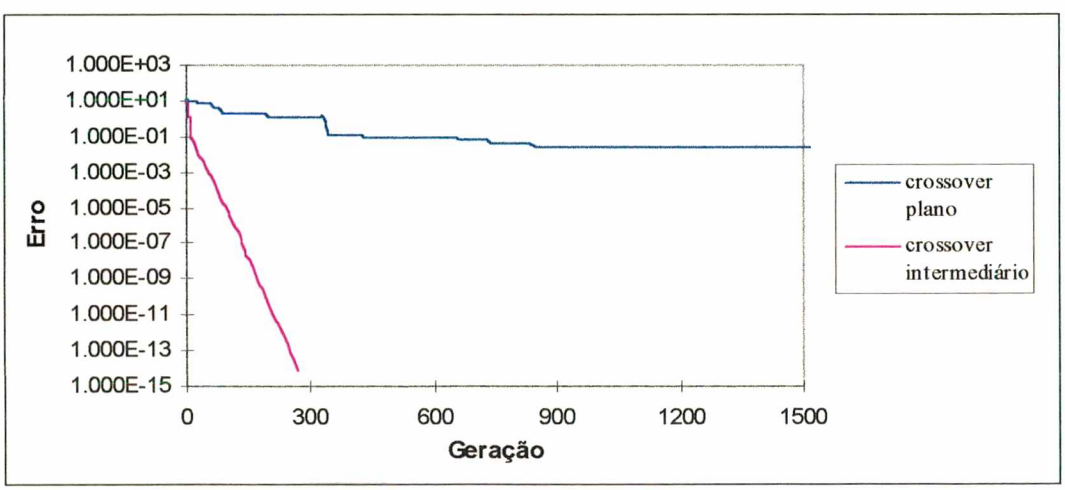

Figura 6. 14 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *linear* e mutação *uniforme*)

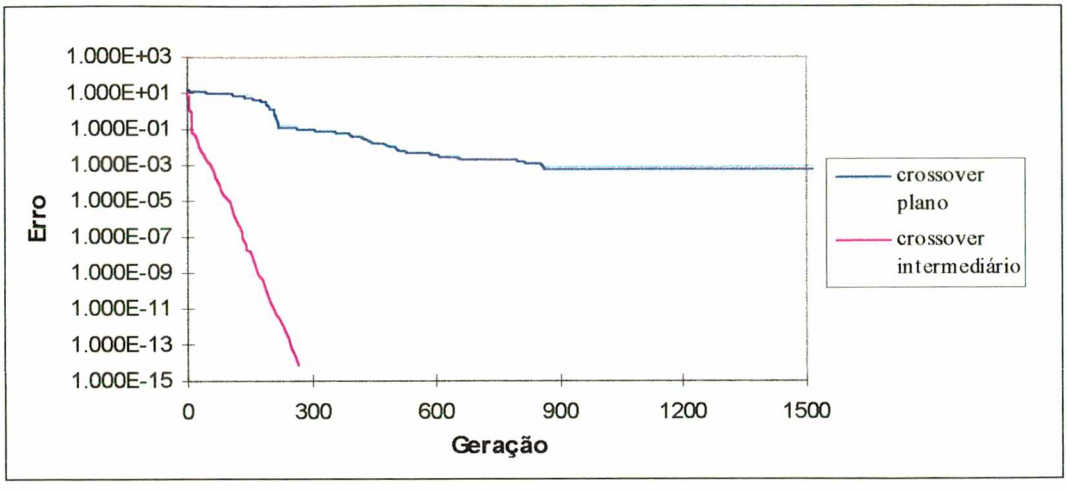

Figura 6. 15 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *linear* e mutação *exponencial*)

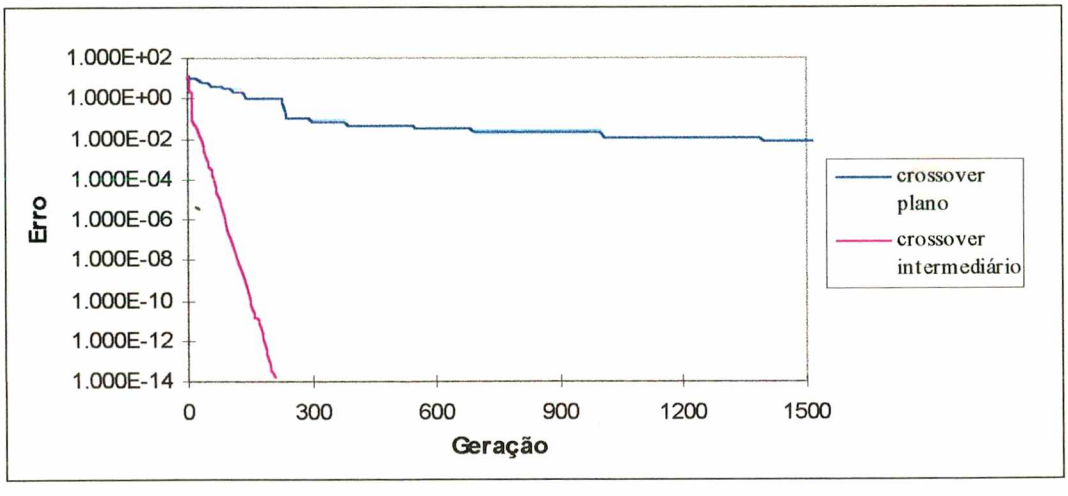

Figura 6. 16- Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *bilinear* e mutação *uniforme*)

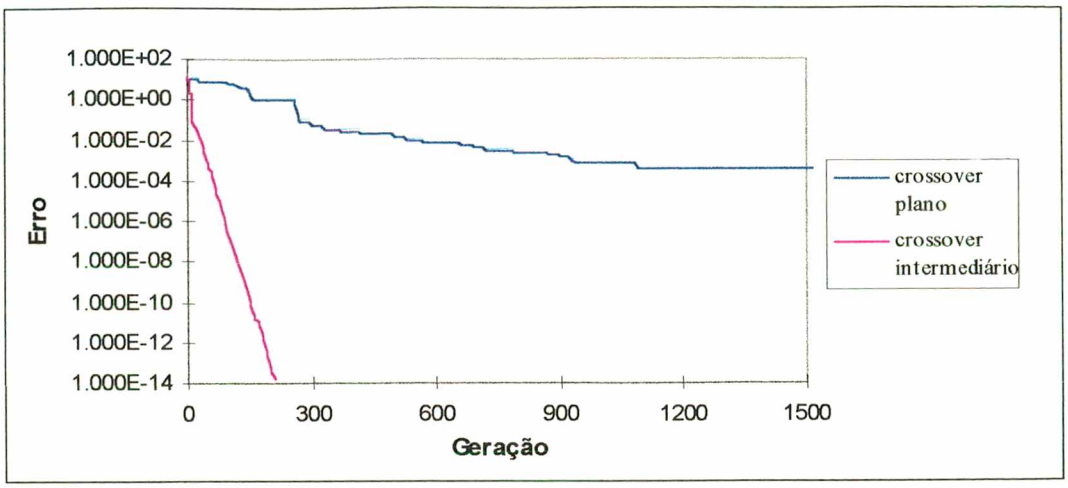

Figura 6. 17 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *bilinear* e mutação *exponencial*)

Nas figuras 6.14 a 6.17 nota-se um comportamento semelhante, em relação a evolução do erro, ao das figuras 6.6 e 6.9 para a função multimodal. Novamente, há uma grande diferença de desempenho entre os algoritmos genéticos que utilizam o *crossover plano* (convergência assintótica do erro) e os que empregam o *crossover intermediário* (convergência exponencial).Também, o melhor resultado foi obtido sem mutação quando se utiliza o *crossover intermediário.*

* *Influência do operador de Mutação e Escalonamento na convergência dos Algoritmo Genéticos.*

Na figura 6.18 mostra-se a evolução do log(erro) de otimização com o *crossover plano* para as combinações dos operadores de mutação e escalonamento.

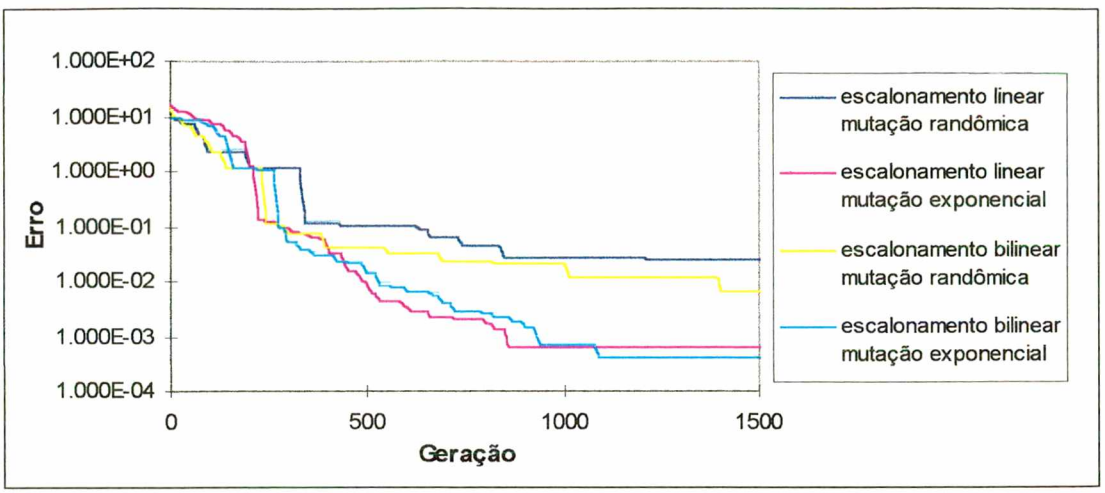

Figura 6. 18 - Influência dos operadores de mutação e escalonamento na convergência do algoritmo genético (*crossover plano*)

A figura 6.19 mostra a evolução do log(erro) com o *crossover intermediário*. Neste caso, pode-se observar apenas a influência do operador de escalonamento uma vez que os melhores resultados foram obtidos sem mutação.

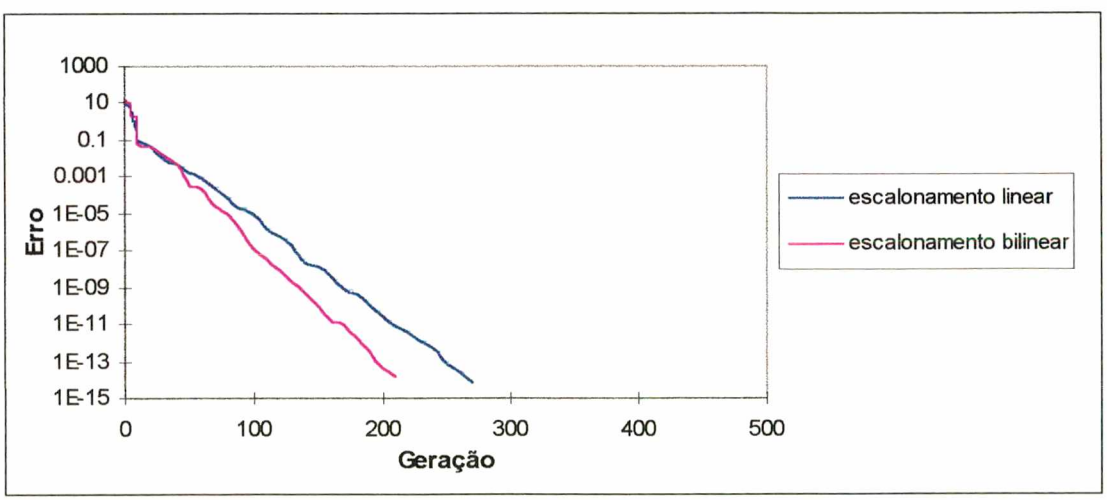

Figura 6. 19 - Influência do operador de escalonamento na convergência do algoritmo genético (*crossover intermediário*)

Na figura 6.18, nota-se que para o *crossover plano* há um aumento da eficiência do algoritmo genético tanto pela utilização da mutação *exponencial*, quanto do escalonamento

bilinear. Para o *crossover intermediário*, figura 6.19, não houve necessidade de *mutação* e o escalonamento *bilinear* apenas aumentou a velocidade de convergência. Dessa forma obtiveram-se resultados semelhantes, em termos do comportamentos das curvas de erro, aos obtidos na otimização da função multimodal (figuras 6.10 e 6.11).

* *Análise da influência os operadores de crossover e mutação na diversidade da população*

Para verificar a influência dos operadores de *crossover* e mutação na diversidade da população, calculou-se a evolução da diversidade, com e sem mutação, para os algoritmos genéticos que resultaram no menor erro de otimização utilizando os *crossover plano* e *intermediário* (ver tabela 6.2). Os resultados da evolução da diversidade podem ser vistos nas figuras 6.20 e 6.21.

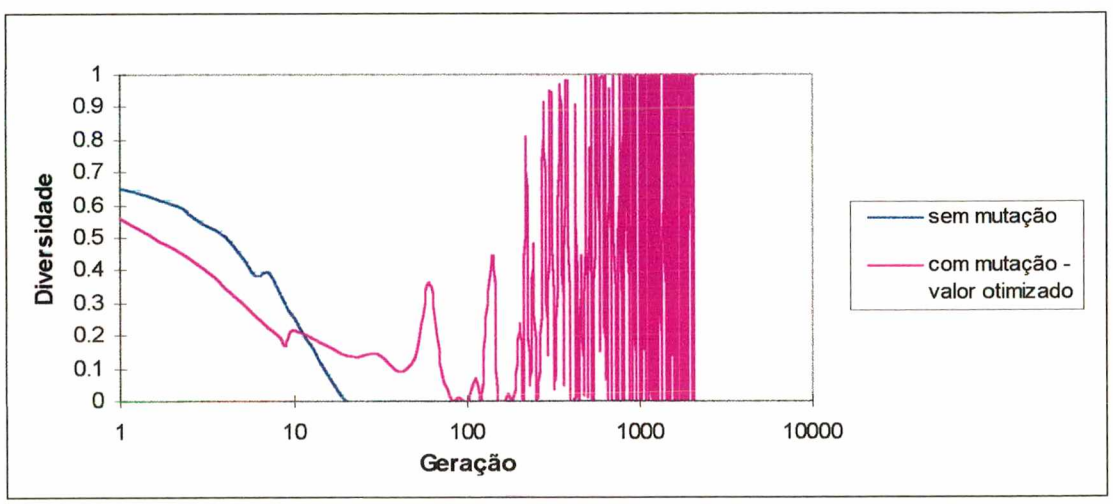

Figura 6. 20 - Influência do *crossover plano* na diversidade da população

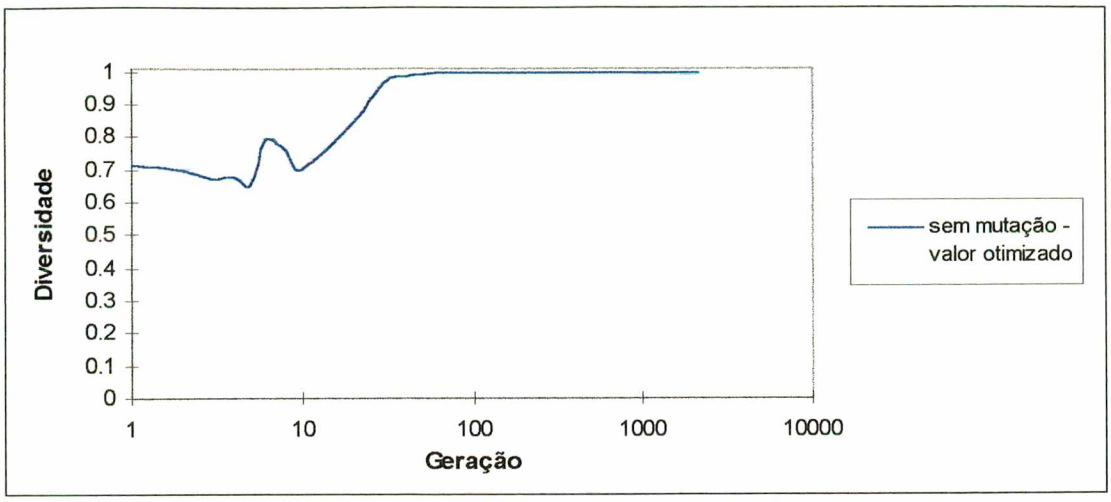

Figura 6. 21 - Influência do *crossover intermediário* na diversidade da população

As figuras 6.20 e 6.21, correspondentes as curvas de evolução da diversidade da função descontínua, possuem o mesmo comportamento das curvas de diversidade da função multimodal, figuras 6.12 e 6.13. Portanto, as mesmas análises feitas para a função multimodal valem para a função descontínua.

Cabe ressaltar que utilizando-se o *crossover intermediário* na otimização de dois problemas completamente diferentes, funções multimodal e descontínua, observou-se características em comum nas curvas de erro e diversidade:

• A diminuição do erro em relação ao número de gerações foi exponencial. (curvas do log(erro) $_x$ número de gerações são retas);

• Os melhores resultados foram atingidos sem qualquer mutação;

• O valor de diversidade é alto e a dispersão dessa é pequena, ou seja, a diversidade não se alterou muito ao longo das gerações.

Desta forma as características observadas nas curvas de erro e diversidade poderiam servir como um indicador para teste de operadores genéticos. A partir de um grupo de problemas com

ótimo conhecido seria possível avaliar o desempenho de operadores genéticos em relação às características descritas.

3)Otimização dos Parâmetros da função cúbica: A tabela 6.3 apresenta os resultados finais das simulações correspondentes à otimização dos parâmetros da função cúbica.

Tabela 6.3 - Desempenho de vários operadores na otimização dos parâmetros da função cúbica

				Erro de Otimização		
Crossover	Mutação	Taxa de Mutação	Escal.	Mínimo	Média	Máximo
plano	*uniforme*	0.01	*linear*	3.503E-01	1.741E+00	1.880E+00
plano	*exponencial*	0.008	*linear*	1.521E-05	2.205E-03	6.460E-01
plano	*uniforme*	0.01	*bilinear*	1.749E-01	5.896E-01	4.320E+00
plano	*exponencial*	0.008	*bilinear*	1.620E-01	1.683E+00	4.644E+00
intermediário	*uniforme*	0.003	*linear*	1.133E-03	1.931E-03	8.662E-03
intermediário	*exponencial*	0.001	*linear*	8.344E-05	1.006E-04	1.102E-03
intermediário	*uniforme*	0.003	*bilinear*	1.996E-07	1.526E-05	1.844E-05
intermediário	*exponencial*	0.002	*bilinear*	8.064E-10	2.167E-08	9.038E-05

Os resultados da tabela 6.3 correspondem a 5000 gerações. Novamente, nota-se superioridade dos algoritmos genéticos que utilizam *crossover intermediário.*

* *Influência do operador de Crossover na convergência do Algoritmo Genético.*

Nas figuras 6.22 a 6.25 mostra-se a evolução do log(erro) de otimização com os *crossover plano* e *intermediário* para várias combinações dos operadores de mutação e escalonamento. Em todos casos observou-se um desempenho superior dos algoritmo genéticos que utilizam o *crossover intermediário* em relação aos que empregam o *crossover plano*.

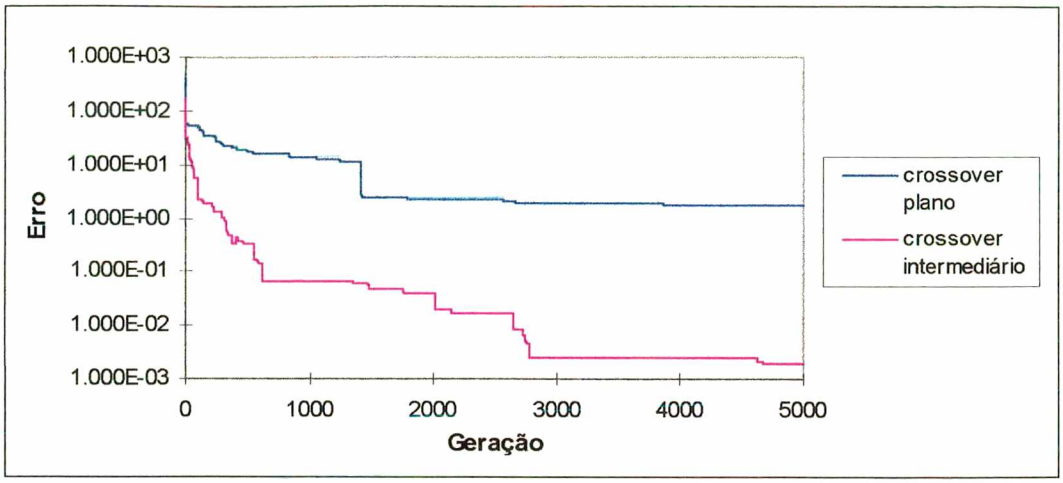

Figura 6. 22 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *linear* e mutação *uniforme*)

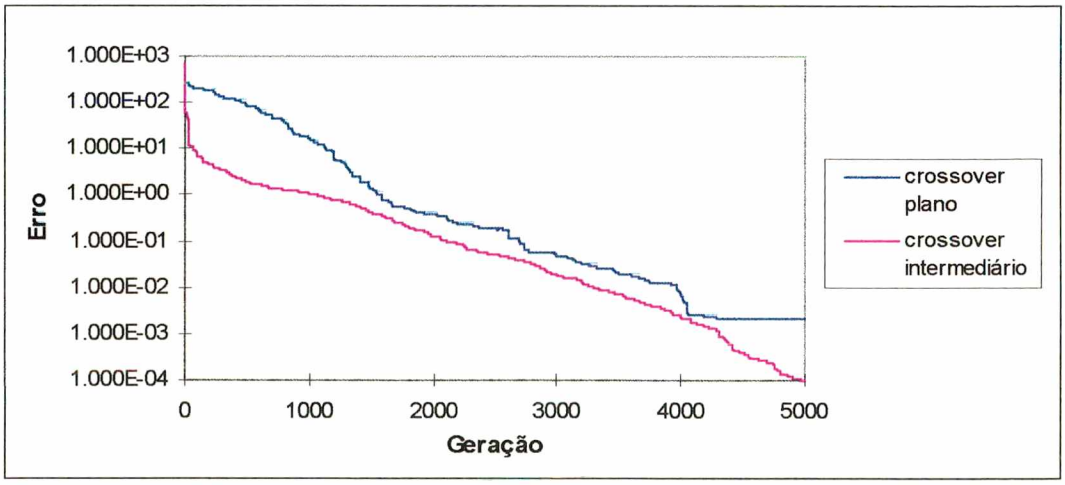

Figura 6. 23 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *linear* e mutação *exponencial*)

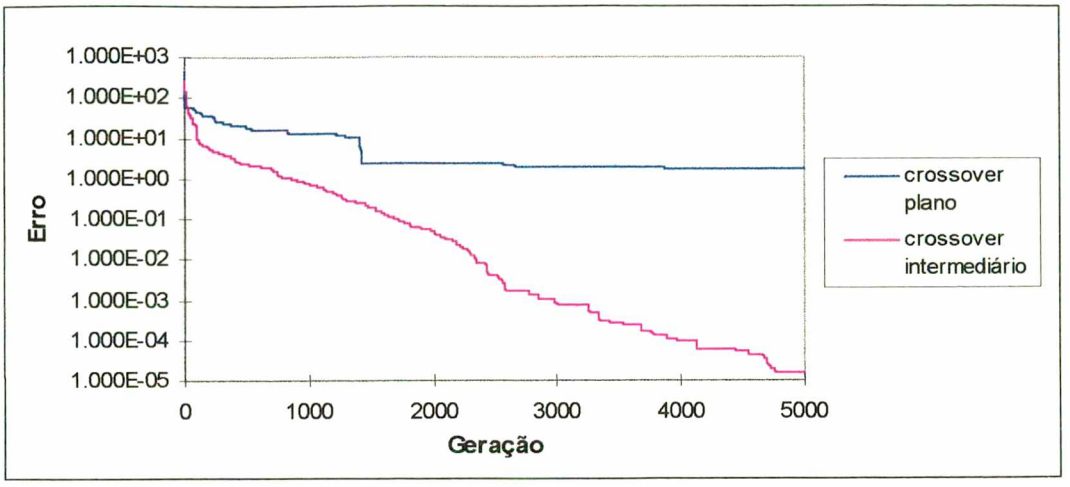

Figura 6. 24 - Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *bilinear* e mutação *uniforme*)

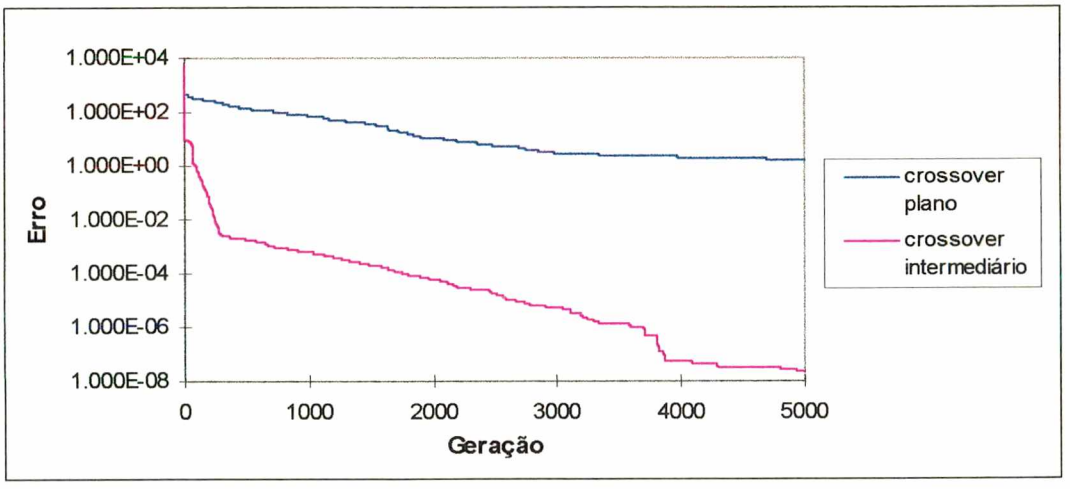

Figura 6. 25- Influência dos operadores de *crossover* na convergência do algoritmo genético (escalonamento *bilinear* e mutação *exponencial*)

Como pode ser observado nas figuras 6.22 a 6.25 os resultados obtidos com o *crossover intermediário* foram superiores aos conseguidos com o *crossover plano*. Utilizando-se este último operador o algoritmo genético convergiu lentamente para o ótimo, da mesma forma nos outros problemas propostos. No entanto, a maior diferença, em relação aos outros problemas propostos, está na convergência do algoritmo genético com *crossover intermediário*, neste caso as curvas de erro apresentam inflexões. Além do mais, o valor ótimo de taxa de mutação não é mais igual a zero (ver tabela 6.3).

** Influência dos operadores de Mutação e Escalonamento da convergência do Algoritmo Genético.*

Nas figura 6.26 mostra-se a evolução do log(erro) utilizando-se o *crossover plano* para as combinações dos operadores de mutação e escalonamento. Na figura 6.27, mostra-se a mesma curva para o *crossover intermediário*.

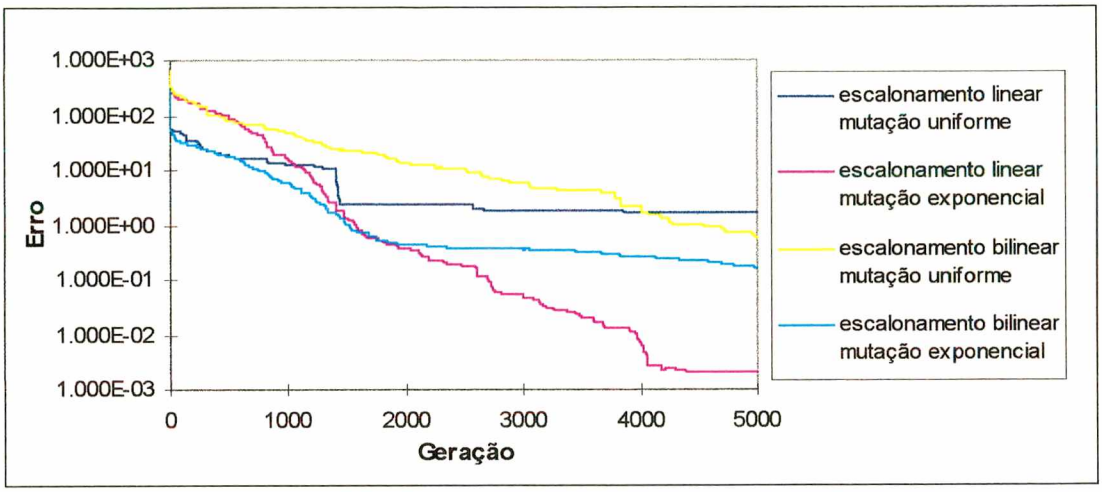

Figura 6. 26 - Influência dos operadores de mutação e escalonamento na convergência do algoritmo genético (*crossover plano*)

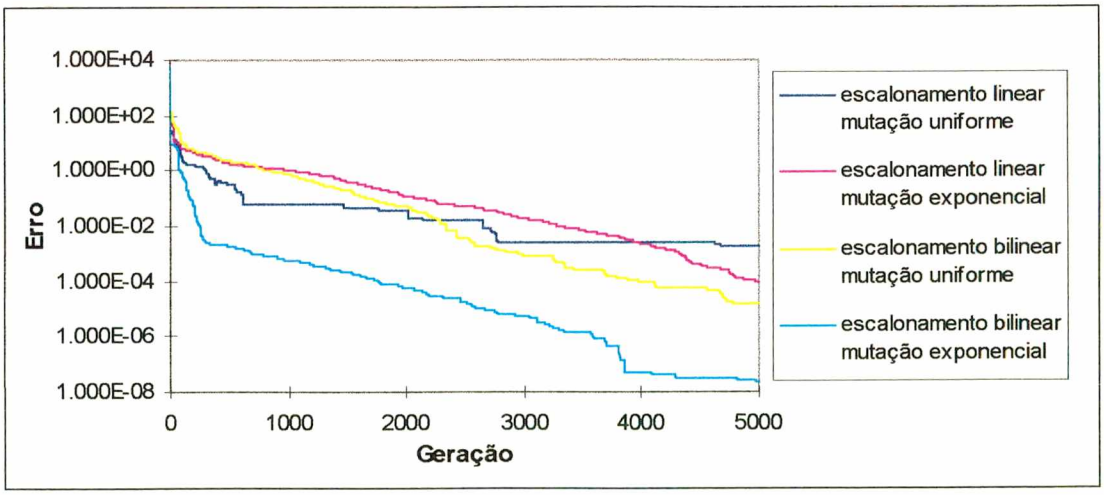

Figura 6. 27 - Influência dos operadores de mutação e escalonamento na convergência do algoritmo genético (*crossover intermediário*)

Das figuras 6.26 e 6.27, observa-se que os melhores resultados foram obtidos com a mutação *exponencial*. O escalonamento *linear* formou uma combinação melhor com o *crossover plano* (figura 6.26) e o escalonamento *bilinear* com o *crossover intermediário* (figura 6.27). Cabe ressaltar que o algoritmo genético que obteve o melhor desempenho na otimização dos parâmetros da cúbica foi o mesmo que no caso das funções *multimodal* e *descontínua*, sendo que, os operadores que constituem o melhor algoritmo genético determinado até o momento são: *crossover intermediário*, mutação *exponencial* e escalonamento *bilinear*.

• *Análise da influência dos operadores de crossover e mutação na diversidade*

Nas figuras 6.28 a 6.31 mostra-se a influência dos operadores de *crossover* e mutação na evolução da diversidade da população. As figuras adicionais em escala log-log, figuras 6.29 e 6.31, tem por objetivo mostrar alguns efeitos ocorridos com a diversidade da população para taxa de mutação zero.

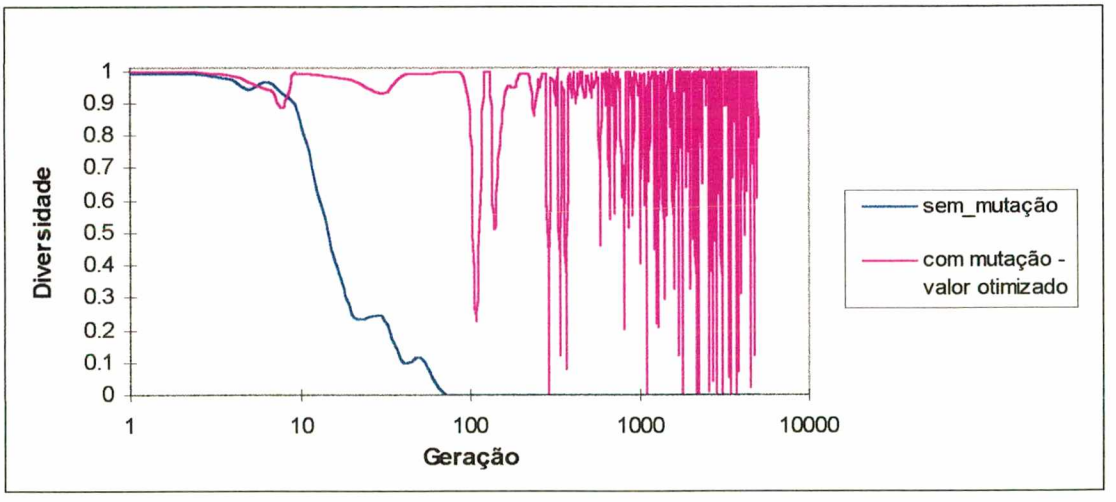

Figura 6. 28 - Influência do *crossover plano* (escala semi-log) na diversidade da população

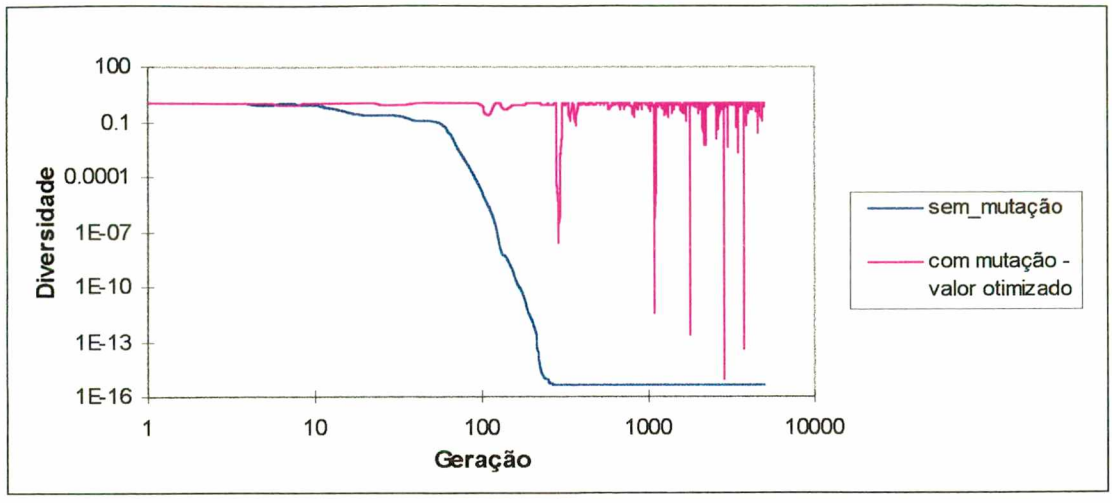

Figura 6. 29 - Influência do *crossover plano* (escala log-log) na diversidade da população

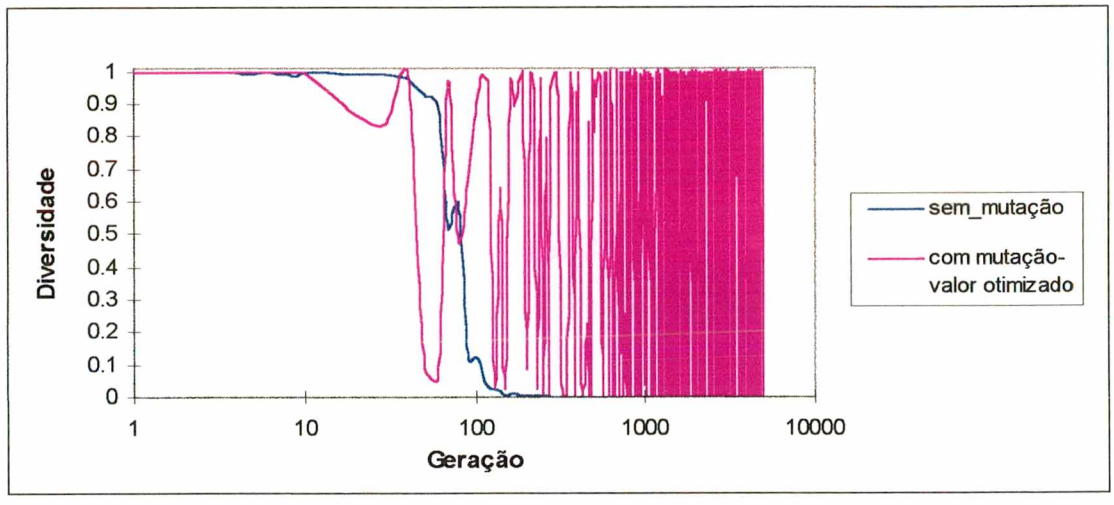

Figura 6. 30 - Influência do *crossover intermediário* (escala semi-log) na diversidade da população

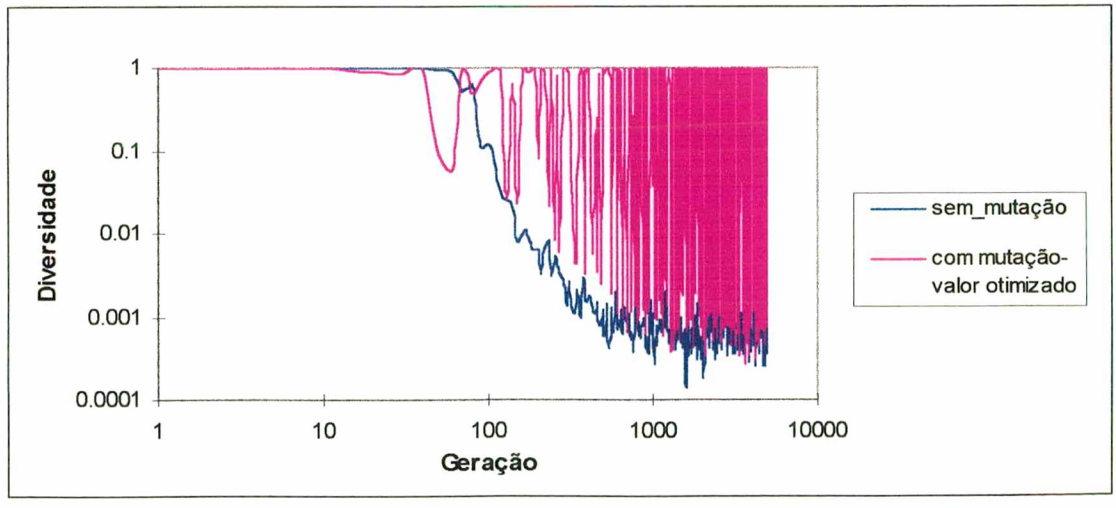

Figura 6. 31 - Influência do *crossover intermediário* (escala log-log) na diversidade da população

As curvas da evolução da diversidade da população relativas ao *crossover plano* com e sem mutação, figura 6.28, apresentam o mesmo comportamento que as obtidas na simulação das funções *multimodal* e a da *descontínua*. A grande diferença em relação a otimização dos outros problemas propostos encontra-se no comportamento da curva da diversidade para o algoritmo genético que utiliza o *crossover intermediário* (figura 6.30). Esta curva não mais se assemelha às curvas de diversidade do *crossover intermediário* obtidas das funções *descontínua* e *multimodal*, parecendo-se mais com as curvas do *crossover plano*. Ainda assim, pode-se notar diferenças entre a diversidade da população para os *crossover* se as curvas de diversidade das figuras 6.28 e 6.30 forem graficadas em escalas log-log (figuras 6.29 e 6.31). Comparando as curvas sem mutação das figuras 6.29 e 6.31 percebe-se que o *crossover intermediário* consegue manter a diversidade em torno de um valor pequeno mas constante, enquanto o *crossover interpolador* produz um decréscimo exponencial da diversidade da população.

Mesmo que para este problema o algoritmo genético com *crossover intermediário* necessite de mutação, o erro de otimização encontrado é bem menor do que o obtido com o *crossover plano*. Isto indica que apesar do *crossover intermediário* não conseguir, por si próprio, manter a diversidade da população pelo menos aproveita melhor a diversidade produzida pela mutação quando comparado ao *crossover plano*.

6.3 Modificações no Algoritmo Genético

Há situações em que a eficiência do algoritmo genético diminui consideravelmente. Entre as principais razões estão a não linearidade da função objetivo, a interação entre variáveis e o aumento de dimensionalidade do problema. Nestes casos, pode ser extremamente difícil para o algoritmo genético determinar "bons" indivíduos.

Duas estratégias foram testadas para tentar aumentar eficiência do algoritmo genético obtido: A primeira foi a substituição do elitismo e a segunda foi a utilização de um novo *crossover*, proposto neste trabalho, que é uma modificação do *crossover intermediário*.

O elitismo foi utilizado no trabalho de MONTANA (1995) [10]. Este operador compara o valor de função objetivo de cada novo indivíduo gerado com o pior da população. Caso o novo indivíduo seja melhor que esse último, o novo indivíduo é colocado na população e pior retirado dela. Nas simulações que seguem, o elitismo descrito será denominado de *substitui os mais fracos* e o que foi utilizado nas otimizações até o momento de *recoloca os melhores*.

O novo *crossover*, que será chamado de agora em diante de *direcional*, utiliza o valor da função objetivo para guiar a pesquisa e aumentar a probabilidade de geração de melhores filhos. Este operador gera dois novos indivíduos a partir de dois selecionados através dos seguintes procedimentos:

1) Verifica-se qual o melhor pai;

2) Gera-se o primeiro filho: Formam-se pares entre os mesmos genes nas *strings* dos pais selecionados e a seguir, para cada par, determina-se um valor aleatório dentro do intervalo delimitado por estes. O valor encontrado será atribuído ao gene do filho (*interpolação*). O procedimento é repetido até que todos os genes do primeiro filho sejam determinados;

3) Gera-se o segundo filho: Calcula-se, para cada gene dos indivíduos selecionados, a distância entre um gene no primeiro pai e o respectivo gene no segundo pai. Em seguida, extrapola-se esta distância na direção do melhor pai gerando um novo ponto no espaço de pesquisa. O valor atribuído a cada gene do segundo filho será determinado aleatoriamente entre o gene contido no melhor pai selecionado e o do ponto obtido por (*extrapolação*). Repete-se este procedimento até que a *string* do segundo filho seja completada. Uma ilustração do novo operador pode ser vista na figura 6.32.

O *crossover direcional* pode produzir soluções da otimização fora da faixa de interesse, principalmente nas primeiras gerações devido à grande diversidade da população. Para evitar que isso ocorra pode-se utilizar o mesmo procedimento utilizado pelo *crossover intermediário* no controle da *extrapolação* (seção 5.2). O *crossover direcional* pode ser visto como uma versão local do operador de seleção. Neste último, atribui-se um maior *fitness* aos melhores indivíduos,

esperando-se que o ótimo esteja próximo a estes. No operador proposto, os filhos são gerados próximos ao melhor pai. Dessa forma a pesquisa que o *crossover direcional* realiza nos indivíduos selecionados é semelhante a efetuada pelo operador de seleção na população.

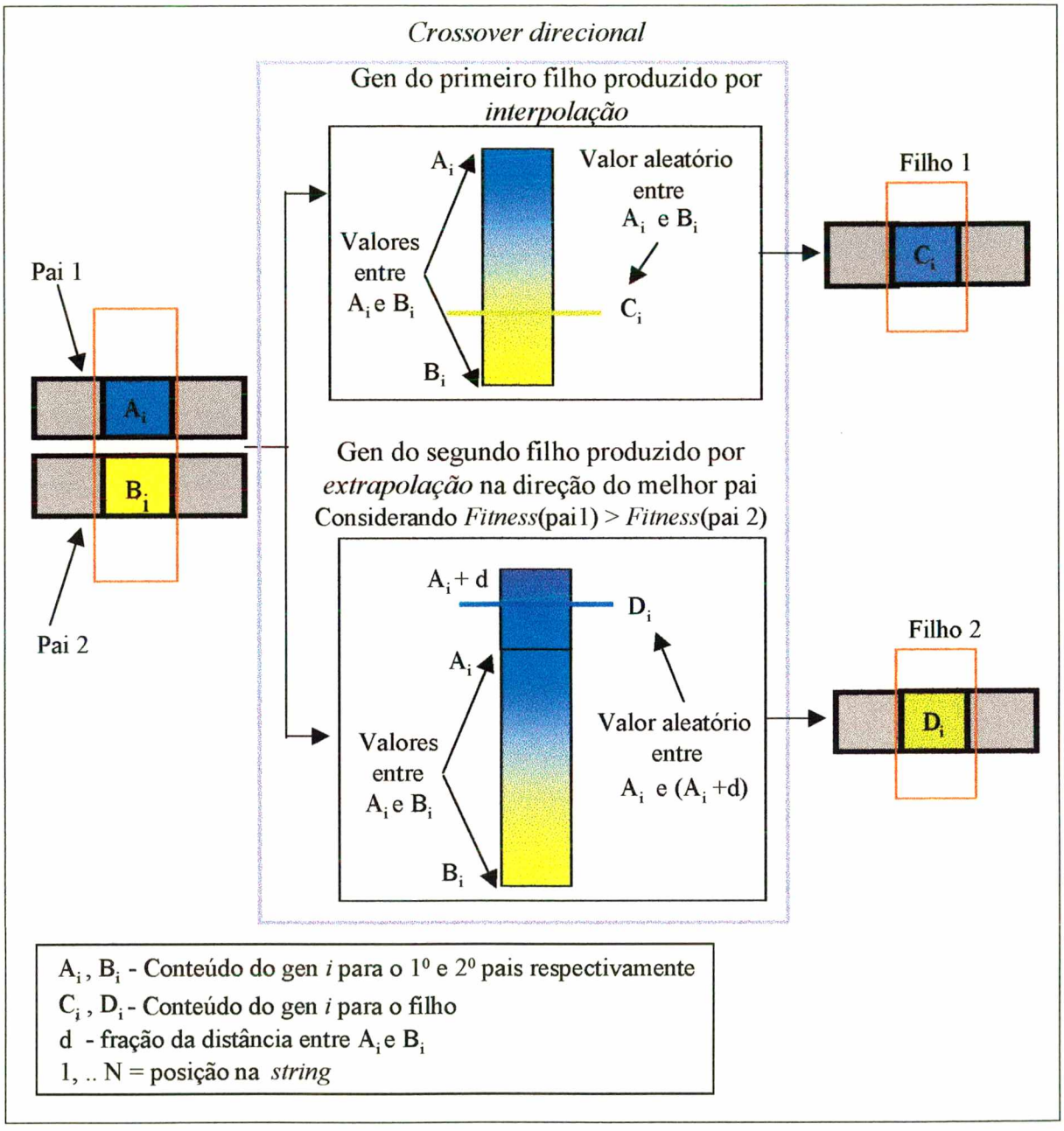

Figura 6. 32 – Ilustração do funcionamento do *crossover direcional*

6.4 Determinação dos Parâmetros de uma Rede Estática

O problema escolhido para comparar os algoritmos genéticos, com e sem modificações, consistiu do treinamento de uma rede neural *feedforward*, O treinamento de redes neurais é um problema adequado para teste de algoritmos genéticos, já que se trata de um problema com elevado número de variáveis e interação entre essas, além da não linearidade presente devido à topologia da rede.

A função objetivo utilizada foi o erro médio quadrático de predição entre a saída da rede e a saída de um processo. Os dados do processo foram obtidos da seguinte equação dinâmica:

$$Y(s) = \left(\frac{2}{3s+1} - \frac{1}{2s+1} \right) u(s)$$

Onde: Y(s) é saída do processo;

U(s) é a variável de perturbação do processo no domínio de Laplace.

A saída do processo, no domínio do tempo, foi obtida utilizando o método *Runge-Kutta* de 4-5 ordem com passo ajustável, assumindo-se um formato de dente de serra para a variável de perturbação. O intervalo de amostragem utilizado foi de 0.1 segundos. A perturbação e a saída do processo podem ser vistas nas figuras 6.33 e 6.34 respectivamente.

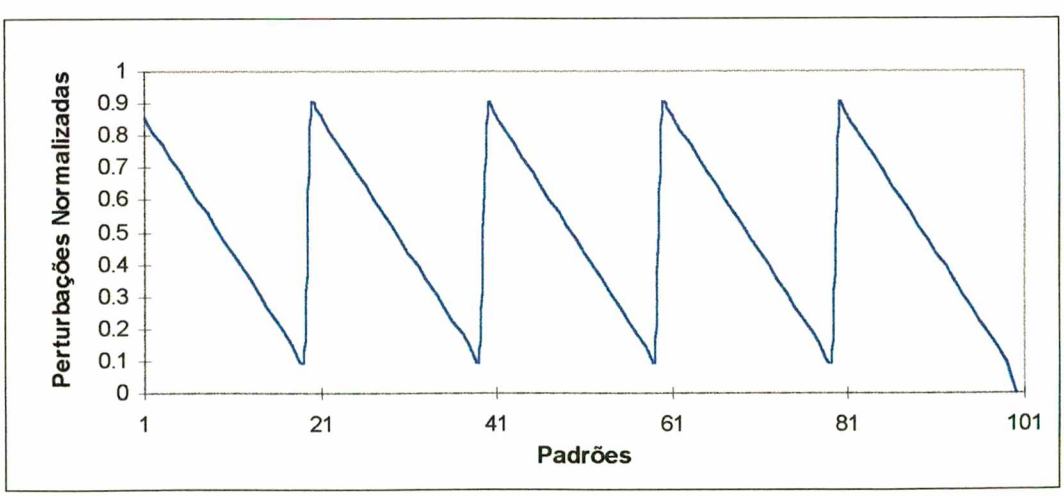

Figura 6. 33 - Variável de perturbação do Processo

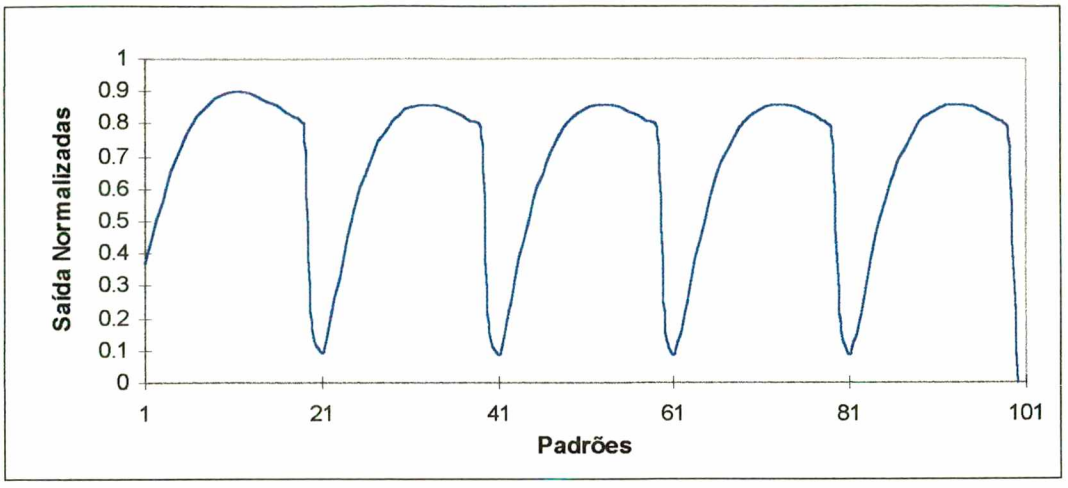

Figura 6. 34 - Saída do Processo

Na identificação do sistema foi utilizada uma rede neural *feedforward* com 3 camadas. A arquitetura da rede foi formada por 3 entradas, 7 neurônios na camada escondida, 1 na camada de saída e 1 bias por neurônio, o que resulta em um total de 36 pesos a serem determinados. As entradas da rede neural consistiram dos valores atrasados da saída do processo e da variável de perturbação, ou seja, de (y(t), y(t-1), u(t)) respectivamente. A rede neural foi utilizada como um preditor e portanto, a saída deveria ser a mais próxima possível de y(t+1).

O problema em questão é semelhante ao problema da cúbica descrito na seção 6.2, já que se trata de estimação de parâmetros, porém, neste caso não há solução analítica devido à estrutura fortemente não linear da rede neural.

Cabe ressaltar que nesta etapa deseja-se apenas minimizar o erro quadrático de treinamento sem importar a capacidade de aprendizagem da dinâmica do sistema pela rede neural. Esta capacidade será estudada posteriormente no capítulo de identificação de sistemas.

6.4.1 Resultados

Para cada combinação de *crossover* (*intermediário-direcional*) e elitismo (*recoloca melhores - substitui mais fraco*) efetuou-se um total de 10 corridas, sendo que, anotou-se em cada caso o mínimo, a média e máximo erros quadráticos de treinamento em 600 gerações. Nos algoritmos genéticos testados foram mantidos o escalonamento *bilinear* e a mutação *exponencial* que faziam parte do melhor algoritmo genético determinado na seção 6.2. A taxa de *crossover* foi fixada em 1 e a taxa de mutação determinada em cada caso. A faixa de variação dos pesos foi admitida em [-6,6] (este intervalo foi determinado através de um estudo preliminar). Os resultados do treinamento, para todas as combinações dos operadores genéticos descritas após 600 gerações, podem ser vistos na tabela 6.4.

Tabela 6.4 - Comparação do erro quadrático de treinamento para os operadores genéticos testados

			Erro médio quadrático		
Crossover	Elitismo	Taxa de Mutação	Mínimo	Média	Máximo
intermediário	*recoloca os melhores*	0.003	1.506E-03	1.997E-03	2.347E-03
intermediário	*substitui mais fracos*	0.003	2.287E-03	2.511E-03	2.799E-03
direcional	*recoloca os melhores*	0.002	1.479E-03	1.859E-03	2.279E-03
direcional	*substitui mais fracos*	0.002	4.792E-04	6.370E-04	1.214E-03

Observando a tabela 6.4, percebe-se que houve uma diminuição do erro de treinamento obtido pela modificação tanto do elitismo quanto do *crossover*. A média do erro médio quadrático de treinamento da rede neural foi diminuído cerca de três vez, de 1.997e-3 para 6.37e-4, pela utilização do *crossover direcional* e o elitismo *substitui mais fracos*.

Na figura 6.35 ilustra-se a saída do processo e da rede neural para o caso correspondente à média do erro médio quadrático do melhor conjunto de operadores determinado anteriormente (*crossover intermediário* e elitismo *recoloca melhores*), e na figura 6.36 as mesmas curvas para o algoritmo genético que utiliza o *crossover direcional* e elitismo *substitui mais fracos*.

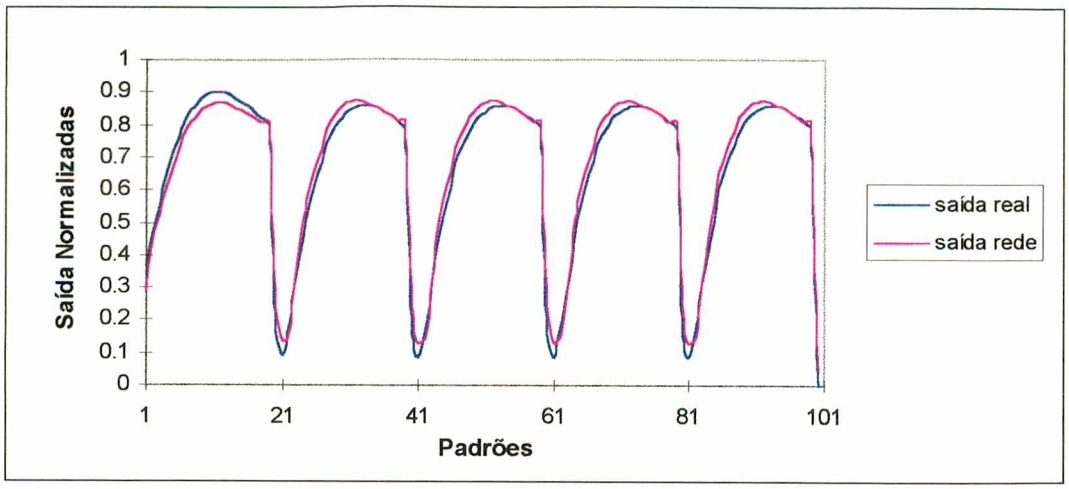

Figura 6. 35 -Saída real e a predita pela rede

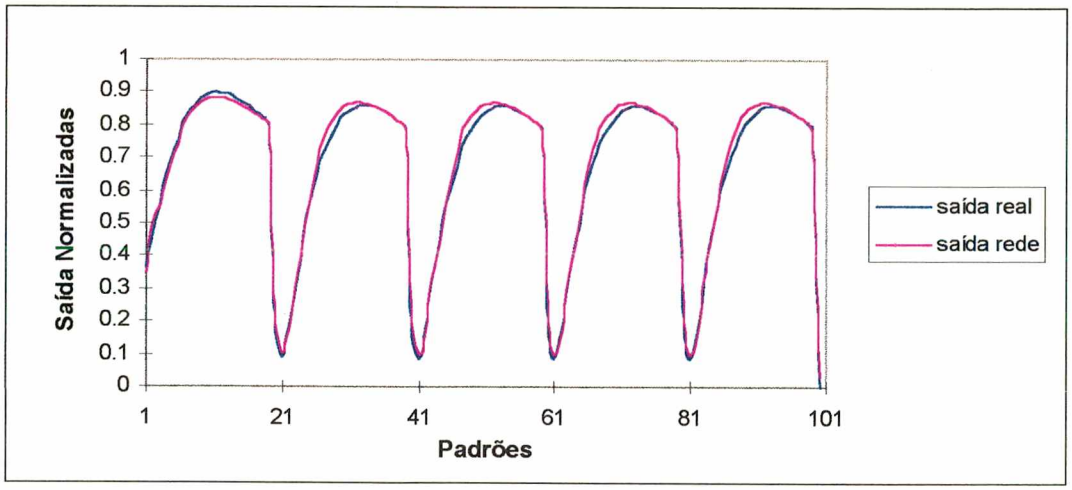

Figura 6. 36- Saída da rede e do sistema (melhor resultado)

Comparando as figuras 6.35 e 6.36 nota-se que a inclusão das novas modificações diminuiu sensivelmente o erro de predição nas regiões extremas do grupo de dados.

O desempenho dos algoritmos genéticos foi também comparado ao do método de treinamento *backpropagation.* O método em questão foi implementado por MAZZUCO (1996) [44] em sua dissertação de mestrado. Na tabela 6.5 mostra-se os resultados obtidos de 10 otimizações efetuadas e o fator de aprendizagem empregado. Em todos os casos utilizaram-se

60.000 épocas para o treinamento da rede *feedforward*, o que equivale exatamente as 600 gerações utilizadas no algoritmo genético para uma população de 100 indivíduos.

Tabela 6.5 – Erro de predição utilizando o método *backpropagation*

	Erro médio quadrático		
Fator de aprendizagem	mínimo	médio	maior
0.01	5.678E-04	7.573E-04	1.09E-03

Comparando as tabelas 6.4 e 6.5, nota-se que o mínimo e a média do erro quadrático de treinamento obtidos com o algoritmo genético modificado foram inferiores aos do método *backpropagation*. O resultado obtido é bastante positivo, lembrando-se que o algoritmo genético é um **método geral** que pode ser empregado, com algumas modificações, no treinamento de qualquer arquitetura de rede neural. Por outro lado, o método *backpropagation* é **específico** para o treinamento de redes *feedforward* já que utiliza o conhecimento de derivadas.

6.5 Comparação do Algoritmo Genético Modificado com outro encontrado na Literatura

Os *benchmarks* de otimização em algoritmos genéticos são problemas com as mais variáveis características, tais como: unimodalidade, multimodalidade, descontinuidades, alta interação entre variáveis etc. Dessa forma pode-se avaliar o desempenho de um algoritmo genético em relação a um conjunto de *benchmarks* adequadamente selecionados. O desempenho do algoritmo genético com o *crossover direcional* foi comparado ao *software Gaot* (*toolbox* do *Matlab*®), HOUCK *et alli* (1995) [55] na minimização dos seguintes problemas, reconhecidos como *benchmarks* na literatura:

Função 1: Função Michalewicz's, POHLHEIM (1997) [54]:

$$F_1 = -\sum_{i=1}^{N} sin(x_i) \cdot sin^{2m}(\frac{i.x^2}{\pi})$$

onde: m =10;

$x_i \in [0, \pi]$, i=1, ... N; N é a dimensão.

Ótimo ≈ - 4.687 com N = 5 e -9.66 para N =10.

Função 2: Função Griewangk's, POHLHEIM (1997) [54]:

$$F_2 = \frac{1}{d}\sum_{i=1}^{N}(x_i - 100)^2 - \prod_{i=1}^{N}\cos(\frac{x_i - 100}{\sqrt{i}}) + 1$$

onde: d = 4000;

$x_i \in [-600,600]$ i=1, ... N;

Ótimo ≈ 1.0E-4.

Função 3: Função Schwefel's, POHLHEIM (1997) [54]:

$$F_3 = -\sum_{i=1}^{N} x_i \cdot sin(\sqrt{|x_i|})$$

onde: $x_i \in [-500,500]$ i=1, ...N;

Ótimo ≈ -N*418.9829;

Função 4: Função De' Jong (1), GOLDBERG (1989) [52]:

$$F_4 = \sum_{i=1}^{N} x_i^2$$

onde: $x_i \in [-5.12, 5.12]$ i=1, ...N;

Ótimo ≈ 1.0E-6;

Função 5: Função De'Jong (5), GOLDBERG (1994) [52].

$$F_5 = -\left(0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right)$$

Onde:

$x_i \in [-65.536, 65.536]$; i=1,2;

a_{ij} estão definidos no apêndice 2;

Ótimo ≈ -1.002.

Função 6: Estimativa de parâmetros

Gerou-se um grupo de dados com 20 pontos, variando-se as variáveis independentes *x* e *y* de acordo com a seguinte equação:

$$H_i = 1 + 2.5.x + 3.5.x^2 + 0.y + 4.5.y^2 + 6.x.y$$

O grupo de dados obtido pode ser visto no apêndice 3.

A função objetivo foi definida como o erro quadrático de aproximação entre o grupo de dados e a expressão:

$$G_i = c_1 + c_2.x + c_3.x^2 + c_4.y + c_5.y^2 + c_6.x.y$$

Dessa forma, a função objetivo pode ser escrita como:

$$F_6 = \sum_{i=1}^{20}(H_i - G_i)^2$$

Como se está tentando ajustar uma curva a um grupo de dados gerado através da mesma forma funcional tem-se que:

Ótimo ≈ 0.0.

Na resolução de cada problema foram efetuadas um total de 10 corridas, sendo registrados o melhor, médio e pior desempenhos em cada caso. As funções F_1, F_2, F_3, F_4 foram otimizadas para dimensões 2, 5 e 10.

6.5.1 Parâmetros e Operadores

Os operadores e parâmetros utilizados no *software Gaot* e no algoritmo genético com o *crossover direcional* podem ser vistos abaixo.

***Software Gaot*:**

Os parâmetros do *software Gaot* e os operadores utilizados na otimização dos problemas propostos são descritos abaixo:

- Seleção *tournament*;
- Mutação *uniforme* com os parâmetros *default* propostos no *Gaot*;
- População = 100 indivíduos;
- Número de gerações = 5000.

Três tipos de *crossover* foram utilizados: *arithX* (*crossover aritmético*), *simpleX* (*crossover simples*) e *heuristicX* (*crossover heuristico*). Os parâmetros para os operadores de *crossover* foram os *default* do *toolbox Gaot.*

Algoritmo Genético com o *Crossover Direcional*

Operadores:

- Seleção: *rolleta* com escalonamento *bilinear*;
- Mutação: *exponencial e uniforme*.

Parâmetros:

- Taxa de *Crossover* = 1;
- Taxa de Mutação = Uma taxa de mutação diferente foi utilizada em cada caso para otimizar o desempenho do algoritmo genético;
- População = 100 indivíduos;
- Número de gerações = 5000.

6.5.2 Resultados

Para cada *benchmark* descrito comparou-se o desempenho do algoritmo genético modificado com o dos algoritmos genéticos formados dos vários operadores de *crossover* implementados no *software Gaot*. Em cada caso, foram efetuadas 10 otimizações, sendo que, nas tabelas 6.6 a 6.17 podem ser vistos os resultados finais das otimizações efetuadas. Para cada problema, apresentaram-se os resultados em duas tabelas: na primeira, compara-se o desempenho do algoritmo genético com o *crossover direcional* em relação aos implementados no *software Gaot*. Na segunda tabela são mostradas as variáveis obtidas da otimização utilizando o *crossover direcional*.

Função F1

Tabela 6.6 – Resultados das otimizações efetuadas para a função *F1*

			Desempenho	
Dimensão	*Crossover*	Tipo de mutação /Taxa	Melhor	Média
2	*arithX*	(*)	-1.80130305	-1.80130166
2	*simpleX*	(*)	-1.80130116	-1.80128552
2	*heuristX*	(*)	-1.80130147	-1.80130132
2	*direcional*	-	-1.80130341	-1.80130341
2	Ótimo	-1.801		
5	*arithX*	(*)	-4.20520437	-4.20518233
5	*simpleX*	(*)	-4.20518536	-4.20510225
5	*heuristX*	(*)	-4.20519281	-4.20514676
5	*direcional*	*uniforme* - 0.03	-4.68765818	-4.68765817
5	Ótimo	-4.6876		
10	*arithX*	(*)	-9.17646029	-9.17480864
10	*simpleX*	(*)	-9.17524758	-9.16396247
10	*heuristX*	(*)	-9.17596147	-9.16756746
10	*direcional*	*uniforme*- 0.09	-9.66015171	-9.66015171
10	Ótimo	-9.66		

(*) *Default Gaot*

Tabela 6.7 – Variáveis obtidas no melhor desempenho atingido com o *crossover direcional* (função *F1*)

	Valor para cada dimensão		
Variável	2	5	10
1	2.20290552	2.20290552	2.20290551
2	1.57079632	1.57079632	1.57079632
3	-	1.28499157	1.28499158
4	-	1.92305847	1.92305847
5	-	1.72046977	1.72046977
6	-	-	1.57079633
7	-	-	1.45441397
8	-	-	1.75608652
9	-	-	1.65571742
10	-	-	1.57079633

Função F2

Tabela 6.8 – Resultados das Simulações para função *F2*

			Desempenho	
Dimensão	*Crossover*	Tipo de mutação /Taxa	Melhor	Média
2	*arithX*	(*)	2.53143235E-0.7	7.3403909E-0.7
2	*simpleX*	(*)	1.0945375E-0.8	1.70459664E-0.6
2	*heuristX*	(*)	7.83587196E-0.7	1.5408318E-0.6
2	*direcional*	*uniforme* – 0.03	2.55481399E-15	4.11567890E-15
2	Ótimo		1.E-04	
5	*arithX*	(*)	5.36998479E-02	7.32787721E-02
5	*simpleX*	(*)	7.86887972E-02	6,37186367E-02
5	*heuristX*	(*)	6.37113723E-02	2.97913824E-01
5	*direcional*	*exponencial* -0.09	1.47724077E-02	2.21561436E-02
5	Ótimo		1.E-04	
10	*arithX*	(*)	1.80691633E-01	5.78078055E-01
10	*simpleX*	(*)	1.69190541E-01	3.73321618E-01
10	*heuristX*	(*)	2.40438689E-01	2.94484008E-01
10	*direcional*	*exponencial*-0.09	2.46370638E-02	6.88488726E-02
10	Ótimo		1.E-04	

(*) *Default Gaot*

Tabela 6.9 – Variáveis obtidas no melhor desempenho atingido com o *crossover direcional* (função *F2*)

	Valor para cada dimensão		
Variáveis	2 (E+02)	5 (E+02)	10 (E+02)
1	1.00000000	0.96859977	0.96859959
2	1.00000000	1.00000000	0.96561572
3		1.00000000	0.10543322
4		0.99999999	0.10627064
5		1.07007296	0.10000004
6			0.99999884
7			1.00000046
8			1.00000057
9			1.00000099
10			1.00000056

Função F3

Tabela 6.10 - Resultados das otimizações efetuadas para a função *F3*

			Desempenho	
Dimensão	*Crossover*	Taxa de mutação/Taxa	Melhor	Média
2	*arithX*	(*)	-8.37965760E+002	-8.37965130E+002
2	*simpleX*	(*)	-8.37965384E+002	-8.37961776E+002
2	*heuristX*	(*)	-8.37960240E+002	-8.37949695E+002
2	*direcional*	-	-8.37965774E+002	-8.37965774E+002
2	Ótimo	-8.37965800E+002		
5	*arithX*	(*)	-2.09491256E+003	-2.09484950E+003
5	*simpleX*	(*)	-2.09483477E+003	-2.09480300E+003
5	*heuristX*	(*)	-2.09490556E+003	-2.09484290E+003
5	*direcional*	-	-2.09491443E+003	-2.09491443E+003
5	Ótimo	-2.09491450E+003		
10	*arithX*	(*)	-4.18972844E+003	-4.18933613E+003
10	*simpleX*	(*)	-4.18930242E+003	-4.18886007E+003
10	*heuristX*	(*)	-4.18931633E+003	-4.18877309E+003
10	*direcional*	*uniforme*-0.03	-4.18982887E+003	-4.18982887E+003
10	Ótimo	-4.18982900E+003		

(*) Default *Gaot*

Tabela 6.11 – Variáveis obtidas no melhor desempenho atingido com o *crossover direcional* (função *F3*)

	Valor em cada dimensão		
Variável	2 (1.0E+002*)	5(1.0E+002*)	10(1.0E+002*)
1	4.20968746	4.20968745	4.20968746
2	4.20968746	4.20968745	4.20968746
3	-	4.20968746	4.20968744
4	-	4.20968746	4.20968745
5	-	4.20968746	4.20968746
6	-	-	4.20968746
7	-	-	4.20968747
8	-	-	4.20968746
9		-	4.20968747
10	-	-	4.20968746

Função F4

Tabela 6.12- Resultados das otimizações efetuadas para a função *F4*

			Desempenho	
Dimensão	*Crossover*	Taxa de mutação/Taxa	Melhor	Média
2	*arithX*	(*)	8.04200000E-08	2.05078200E-06
2	*simpleX*	(*)	1.31101000E-06	6.49634000E-06
2	*heuristX*	(*)	5.58770000E-07	2.51206667E-06
2	*direcional*	-	6.75729688E-123	2.28692114E-120
2	Ótimo	0		
5	*arithX*	(*)	6.80707000E-06	5.30554533E-05
5	*simpleX*	(*)	2.04040100E-05	4.00868267E-05
5	*heuristX*	(*)	4.72830700E-05	1.33201250E-04
5	*direcional*	-	4.88959196E-67	8.79335983E-65
5	Ótimo	0		
10	*arithX*	(*)	9.06938600E-05	1.63846617E-04
10	*simpleX*	(*)	6.77483110E-04	1.40682888E-03
10	*heuristX*	(*)	6.21517610E-04	8.40086430E-04
10	*direcional*	-	4.67719619E-38	4.92231569E-36
10	Ótimo	0		

(*) Default *Gaot*

Tabela 6.13 – Variáveis obtidas no melhor desempenho atingido com o *crossover direcional* (função *F4*)

	Valor em cada dimensão		
Variável	2	5	10
1	6.05208291E-62	2.88228477E-34	7.29647408E-20
2	-5.56284651E-62	-2.55366720E-34	-3.08475092-E-20
3	-	-2.13625510E-35	-5.23974361E-20
4	-	4.34956437E-35	-7.72916790E-20
5	-	-3.25343539E-35	-8.25595635E-20
6	-	-	-1.03116617E-19
7	-	-	-3.63609754E-20
8	-	-	-8.36600383-E-20
9	-	-	-4.05271457E-21
10	-	-	-7.73975417E-20

Função F5

Tabela 6.14 - Resultados das otimizações efetuadas para a função *F5*

		Desempenho	
Crossover	Tipo de mutação/ Taxa	Melhor	Média
arithX	(*)	-1.00200015	-1.00200015
simpleX	(*)	-1.00200015	-1.00200015
heuristX	(*)	-1.00200015	-1.00200015
direcional	*uniforme*- 0.03	-1.00200015	-1.00200015
Ótimo	-1.002		

(*) *Default Gaot*

Tabela 6.15 – Variáveis obtidas no melhor desempenho atingido com o *crossover direcional* (função *F5*)

Variável	Valor
1	-3.19783104E+001
2	-3.19783805E+001

Função F6

Tabela 6.16- Resultados das otimizações efetuadas para a função *F6*

		Desempenho	
Crossover	Tipo de mutação/ taxa	Melhor	Média
arithX	(*)	9.121938907E-04	6.8595874501E-03
simpleX	(*)	4.214213621E-05	3.1401141832E-03
heuristX	(*)	6.481405247E-04	1.2593700217E-02
direcional	-	4.77063346E-17	4.77063346E-17
Ótimo	0		

(*) *Default Gaot*

Tabela 6.17 – Variáveis obtidas no melhor desempenho atingido com o *crossover direcional* (função *F6*)

Parâmetro	Valor
C_1	3.50000000E+00
C_2	2.50000000E+00
C_3	4.00000001E+00
C_4	1.57632716E-9
C_5	4.99999999E+00
C_6	-3.90000000E+00

6.5.3 Análise de Resultados

Como pode ser observado nas tabelas 6.6, 6.8, 6.10, 6.12, 6.14 e 6.16, o desempenho dos operadores implementados no *software Gaot* foram bastante similares e não se pode afirmar qual o melhor entre eles. Entretanto, em todos os casos, o algoritmo genético com o *crossover direcional* obteve os melhores resultados.

Na tabela 6.6, são mostradas as simulações feitas para a função *F1*. Neste caso, os melhores resultados para o *crossover direcional* foram atingidos sem mutação para dimensões 2, 5 e 10. Os resultados obtidos foram superiores aos indicados na referência, mostrando a grande precisão que pode ser alcançada com o operador proposto.

Das funções testadas, a função F_2 (tabela 6.8) foi a única em que o ótimo não foi determinado em todas as dimensões. Ainda assim, o *crossover direcional* foi superior aos outros *crossover* implementados no *software Gaot*. A dificuldade de otimização do problema pode ser atribuída a grande interação presente entre as variáveis.

Na tabela 6.10, relativa a otimização da função *F3*, observa-se que foram obtidos excelentes resultados para todos os *crossover*. Ainda assim, o algoritmo genético com o *crossover direcional* foi ligeiramente superior para todas as dimensões. Em particular, o valor ótimo da função F_3 é *-N*418.9829*, [POHLHEIM] (1997) [54] ou seja, *-N*K*, onde K é uma constante. Desta forma o desempenho para a função *F3* pode ser avaliado pela relação: K = Valor do ótimo encontrado pelo algoritmo genético / *-N*. Utilizando os dados mostrados na tabela 6.10 chega-se aos seguintes resultados:

$n = 2$: $\Rightarrow$ K = -8.37965800E+002 /-2 = 418.982887;

$n = 5$: $\Rightarrow$ K = -2.09491450E+003 /-5 = 418.982886;

$n = 10$: $\Rightarrow$ K = -4.18982900E+003 /-10 = 418.982887;

Praticamente o mesmo valor para a constante foi obtido em todas as dimensões, indicando um excelente desempenho do algoritmo genético com o *crossover direcional*.

Para a função *F4* (função De' Jong1), cujo mínimo é zero, foi obtida uma grande diferença de desempenhos entre o *crossover* proposto e os implementados no *sotfware Gaot*. Os resultados obtidos com o *crossover direcional* foram muito próximos do mínimo real indicando, tal como nas outras funções, a grande precisão alcançada com o uso desse operador. Pode ser observado na tabela 6.12, que o melhor desempenho foi obtido em todas as dimensões sem mutação. Isto demonstra que, além da precisão atingida, o *crossover* proposto conseguiu também manter a diversidade da população.

Na função *F5* (tabela 6.14), o ótimo global foi encontrado em todos os casos e os resultados obtidos com todos os *crossover* foram bastante similares.

A função F_6 mostra um problema típico de estimação de parâmetros. Do mesmo modo que para a função *F4*, obteve-se uma grande diferença de desempenho entre o *crossover direcional* e os implementados no *toolbox Gaot*. Também, o melhor desempenho na otimização da função F_6 foi obtido sem mutação.

Na otimização das funções *F4* e *F6*, obteve-se um decréscimo exponencial entre o erro de otimização. Nas figuras 6.37 e 6.38, mostra-se a evolução do erro em escala logarítmica para as funções F_4 (10 dimensões) e F_6.

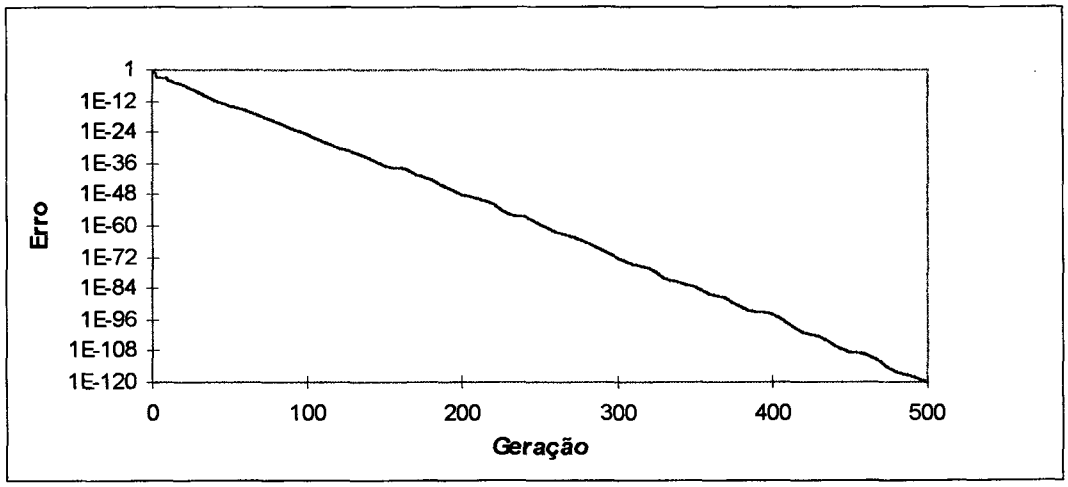

Figura 6. 37 – Evolução do erro de otimização da função F_4 (De Jong'1)

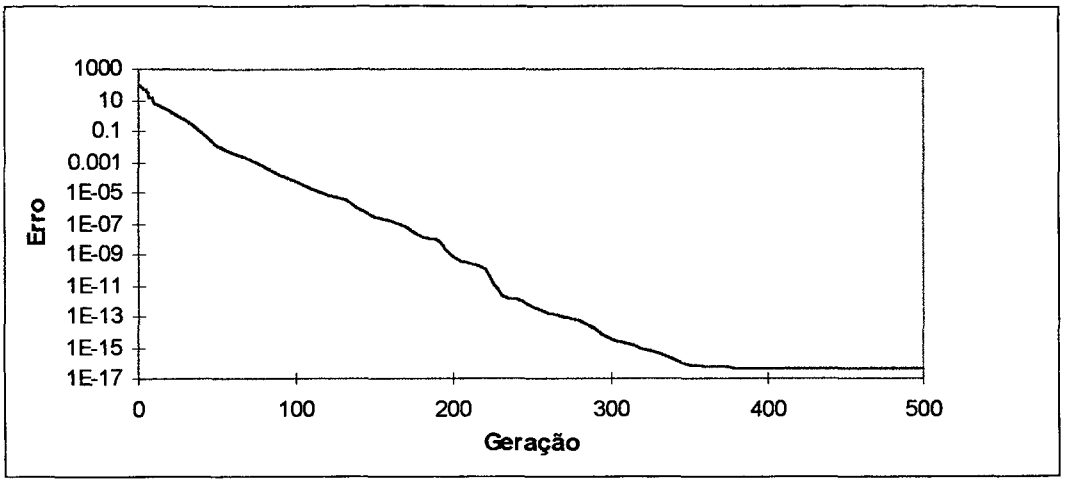

Figura 6. 38 – Evolução do erro de otimização da função F_6 (estimação de parâmetros)

6.6 Conclusões

Pela observação das figuras 6.6 a 6.9, 6.14 a 6.17 e 6.22 a 6.25 relativas ao desempenho dos *crossover plano* e *intermediário,* notou-se uma grande superioridade nos algoritmos genéticos que utilizavam o *crossover intermediário* em relação ao *crossover plano.*

O *crossover plano* gera novos genes pela *interpolação* do conteúdo dos genes contidos nas *strings*. Um algoritmo genético que utilize o *crossover plano* terá dificuldades em atingir o ótimo global se nenhuma *string* for gerada próxima a esse, no momento da criação da população. Ainda assim, mesmo que na população inicial se encontre algum indivíduo próximo ao ótimo, este indivíduo pode ser perdido pela natureza estocástica dos algoritmos genéticos. Pela característica do *crossover plano* a região de pesquisa, a medida que o algoritmo genético converge, vai diminuindo de tamanho até chegar próximo ao ótimo. Porém, se o espaço de procura é muito grande, raramente tem-se elementos da população espalhados adequadamente, causando a contração de toda a população em relação a um ponto, possivelmente longe do ótimo. Qualquer algoritmo genético que utilize este *crossover* necessita do auxílio da mutação, para que não se perca rapidamente a diversidade da população causando a convergência prematura.

Com relação ao *crossover intermediário,* observa-se que ele foi superior ao *interpolador* em todas as corridas. O segredo dele está no uso não só de *interpolação* mas, também de um "pouco" de *extrapolação*. No momento da criação da população, como comentado, os elementos pode não estar espalhados adequadamente e o uso da *extrapolação* permite a busca de soluções fora da região delimitada pelos atuais indivíduos da população. Portanto, este *crossover* necessita menos do auxílio mutação que os *crossover* que utilizam apenas *interpolação*. Esta característica pode ser observada nas tabelas 6.1 a 6.3 onde as taxas de mutação adequadas para o *GA* com *crossover intermediário* são sempre inferiores às utilizadas com o *crossover plano*.

As simulações realizadas para os problemas da função *multimodal* e *descontínua* mostraram que o algoritmo genético com o *crossover intermediário* atingiu o ótimo mesmo sem nenhuma mutação. Nestes casos, obteve-se um valor de erro extremamente baixo para quaisquer combinações de escalonamento e mutação, o que indica que a *extrapolação* não só acelerou a convergência como também aumentou muito a precisão. O *crossover intermediário* conseguiu gerar a diversidade suficiente sem o auxílio de mutação.

Conclui-se pela discussão e pelos resultados obtidos que o uso adequado da *extrapolação* no *crossover* pode aumentar em muito a eficiência do algoritmo genético.

As funções *multimodal* e *descontínua* apresentaram semelhanças no comportamento das curvas de erro e diversidade. Apesar de se tratar de dois problemas completamente diferentes, a presença destas semelhanças poderia servir como um critério, ainda que empírico, no julgamento do desempenho de operadores genéticos.

A mutação *uniforme* modifica um gene aleatoriamente dentro da faixa de interesse da otimização. Caso ocorra a mutação, se o valor de uma certo gene está ruim então ele será modificado e terá uma possibilidade de melhorar. Porém se está bom, ao se efetuar a mutação há um grande possibilidade de piorar. No caso da mutação *exponencial*, procura-se uma valor próximo ao valor atual da variável e, como pode ser visto na otimização dos problemas propostos, este operador funciona melhor que a mutação *uniforme*.

Os algoritmos genéticos que utilizaram escalonamento *bilinear* obtiveram um desempenho superior em relação ao *linear*, indicando sua melhor capacidade de manter os níveis de competição entre os indivíduos da população.

Pelo comportamento global da otimização das funções *multimodal*, *descontínua* e da *estimação dos parâmetros da cúbica* pode-se concluir que os melhores resultados foram obtidos com o *crossover intermediário*, mutação *exponencial* e com escalonamento *bilinear*.

No caso do problemas multivariáveis, tal como o treinamento de redes neurais, onde o *crossover* tem dificuldade de gerar bons indivíduos, torna-se necessário melhorar os operadores genéticos. Duas estratégias propostas obtiveram sucesso: A primeira relativa ao elitismo, que elimina apenas os elementos mais fracos da população e dessa forma, impede a perda de bons indivíduos. A segunda, é um novo *crossover* que busca soluções em torno do melhor pai selecionado. A estratégia utilizada no *crossover* emprega informação adicional (Função objetivo). As duas novas modificações mostraram-se eficientes na redução do erro de treinamento de um rede neural *feedforward*, provando que para problemas de maior dimensionalidade a grande dificuldade é a geração de bons indivíduos e garantia de que estes possam distribuir seus genes para muitos outros indivíduos.

O algoritmo genético modificado foi comparado ao método *backpropagation* obtendo resultados ligeiramente superiores que esse último. Cabe ressaltar que os algoritmos genéticos são métodos de otimização de aplicação geral enquanto o método *backpropagation* é específico ao treinamento de redes *feedforward*.

Como último teste, comparou-se o desempenho do algoritmo genético modificado com um *software* de uso geral (*Gaot*) na otimização de alguns *benchmarks* otimização, sendo que, o desempenho do algoritmo genético modificado foi superior em todas os casos ao *software Gaot*.

De acordo com GOLDBERG (1989) [52], no limite da eficiência do algoritmo genético, pode-se obter um aumento exponencial do número de cópias alocadas por bons indivíduos em relação ao número total de bons indivíduos, o que levaria a uma convergência exponencial. Alcançar tal limite na otimização de um problema depende de uma escolha adequada de um

conjunto de operadores. A grande precisão atingida com *crossover direcional* (funções F_4 e F_6), poderia indicar que esse estaria agindo como uma operador de pesquisa local. Dessa forma, tem-se a pesquisa global efetuada pela operador de seleção e, a pesquisa local feita pelo *crossover direcional*, na vizinhança do melhor indivíduo selecionado. Não há nenhuma garantia de que bons indivíduos sejam determinados com tal procedimento, no entanto, agindo dessa maneira é mais provável que o sejam.

Outra observação importante em relação ao *crossover direcional* é que na otimização das funções F_4 e F_6 os melhores resultados foram obtidos sem mutação. Isto demonstra a capacidade do *crossover direcional* em evitar a perda de diversidade da população.

Finalmente, os resultados atingidos em termos de precisão e convergência confirmam a eficiência dos operadores propostos, *crossover direcional* e escalonamento *bilinear*.

7 IDENTIFICAÇÃO DE SISTEMAS

7.1 Introdução

Por identificação entende-se a determinação de um modelo matemático para um sistema ou processo a partir da informação contida nas seqüências temporais de entradas e saídas, HSIA (1977) [56]. A identificação de sistemas também pode ser considerada como a abordagem experimental da modelagem de processos e, divide-se nas seguintes etapas:

- Planejamento experimental;
- Seleção da estrutura do modelo;
- Estimação de parâmetros;
- Validação.

7.1.1 Planejamento Experimental

Planejar experimentos é definir a seqüência de experimentos e a metodologia de coleta de dados, PINTO (1996). [57]. O planejamento experimental, na identificação de sistemas, está relacionado à escolha adequada de um sinal de perturbação para obter a maior quantidade de informação possível sobre a dinâmica do processo. Em geral, o planejamento de experimentos tem os seguintes objetivos, ASTROM (1984) [58]:

- Determinar o grau de dependência entre as variáveis;
- Quantificar a dependência fundamental entre as variáveis da forma mais precisa possível;

- Desenvolver e descriminar modelos;
- Reduzir o trabalho executado para obter uma informação;
- Reduzir o custo da informação;
- Agilizar o processo da obtenção da informação.

7.1.2 Seleção da estrutura do modelo

Para o engenheiro, pela própria natureza prática e exata da engenharia, a tarefa de modelar um sistema finaliza quando se obtém em um conjunto consistente de relações matemáticas que permite a descrição qualitativa e quantitativa do sistema, PINTO (1996) [57]. Para a descrição adequada da realidade, a modelagem não dispensa os dados experimentais, muito pelo contrário, apenas estes permitem a validação do modelo desenvolvido.

Um modelo é uma ferramenta muito útil para resumir o conhecimento sobre um processo e pode ser utilizado na simulação e otimização desse, além do projeto de equipamentos e análise de estratégias.

7.1.3 Estimação de parâmetros

Estimar é determinar os parâmetros de um modelo de tal forma que as predições desse sejam as *mais próximas possíveis* dos dados experimentais. O termo "mais próximas possíveis" se refere a otimização de algum índice de desempenho baseado no erro entre as predições do modelo e os valores reais do processo. Um índice de desempenho muito utilizado para sistema discretos é dado pela equação 7.1, ASTROM *et alli* (1984) [58]:

$$J = \sum_{i=1}^{n} g(e(n)) \quad (7.1)$$

Onde:

J é o índice de desempenho;

n - número de dados experimentais;

e(n) - erro entre o modelo e o n-ésimo dado experimental;

g - Uma função do erro, normalmente escolhida como a função quadrática.

A primeira formulação, solução e aplicação de um problema de identificação foi dada por Gauss em sua famosa determinação da órbita do asteróide Ceres.

Entre as técnicas bem estabelecidas que tem sido aplicadas com sucesso na estimação de parâmetro estão os mínimos quadrados, a máxima verossimilhança e correlação cruzada.

Os métodos dos mínimos quadrados e da máxima verossimilhança tem solução analítica apenas para funções lineares nos parâmetros. Caso contrário, os parâmetros devem ser obtidos iterativamente pela resolução de um sistema de equações não lineares. Para resolver tais sistemas pode-se utilizar métodos tal como o Passo Descendente, *Newton* e Gradiente Conjugado entre outros.

7.1.4 Validação do modelo

Após um modelo ser obtido deve-se testá-lo em um grupo de dados diferente do utilizado na estimação de seus parâmetros já que, um modelo que tenha um pequeno erro quadrático de ajuste, para um determinado grupo de dados, pode apresentar um comportamento completamente

diferente em outras situações, ASTROM *et alli* (1984) [58]. Tal comportamento pode ocorrer devido a problemas em qualquer um dos seguintes pontos:

- Método de Otimização;
- Índice de desempenho;
- Estrutura do modelo;
- Amostragem mal planejada ou executada.

7.2 Estudo de Casos

Neste capítulo foi abordada a identificação de dois sistemas utilizados como *benchmarks* em Engenharia Química. O primeiro caso é um *CSTR* encamisado com múltiplos estados estacionários e o segundo, é um bioreator em que as células sofrem inibição pelo aumento da concentração de produto.

O objetivo deste trabalho é fazer predições sem que as saídas reais atrasadas do processo sejam conhecidas, ou seja, utilizando o método de identificação em *paralelo*. Dessa forma, as redes neurais estáticas devem ser descartadas, conforme comentado no capítulos 3, sendo uma rede neural recorrente o modelo empírico adequado a esta situação. Neste trabalho a *RNN* utilizada é composta de duas camadas: Na primeira estão localizados os neurônios de entrada, que apenas passam as perturbações externas para a próxima camada. Na segunda encontram-se os neurônios escondidos e os neurônios de saída. A descrição detalhada da *RNN* foi feita na seção 3.3.

No treinamento de uma *RNN* não é possível utilizar o método *backpropagation* empregado no treinamento de redes estáticas, já que as recorrências presentes na estrutura da *RNN* inviabilizam a obtenção de regras simples baseadas na derivação do erro quadrático de aproximação. Nesta situação, os algoritmos evolucionários são uma boa opção pois não

dependem do cálculo de derivadas. Neste trabalho foi utilizado um algoritmo genético para treinamento da *RNN*.

Cabe ressaltar que outras *RNNs* diferentes da descrita poderiam ser utilizadas para a identificação de sistemas bem como outros métodos de treinamento. Este trabalho mostra *uma* forma de abordar o problema de identificação.

7.3 Algoritmo Genético

Foi utilizado para treinamento da *RNN* o melhor algoritmo genético obtido no capítulo 6, ou seja, com as modificações no *crossover* e elitismo.

7.3.1 Parâmetros

Efetuou-se um estudo preliminar para determinação dos parâmetros do algoritmo genético. Os valores encontrados podem ser vistos abaixo:

- População = 300 indivíduos. Foram observados em estudos preliminares que populações menores que 300 indivíduos não levavam a soluções adequadas. Populações de maior tamanho acarretavam maior tempo computacional sem alterações significativas no resultado final;

- Taxa de *crossover* = 1. Utilizando uma taxa de *crossover* igual a *um* foram obtidos os melhores resultados, da mesma forma que na otimização dos problemas do capítulo 6. Valores inferiores levam a um aumento no tempo computacional sem modificações na solução do problema;

• Taxa de mutação = 0.03. Taxas de mutação elevadas tendem a transformar os algoritmos genéticos tradicionais em pura pesquisa enumerativa e taxas de mutação muito pequenas tendem a diminuir a diversidade da população. No caso do algoritmo genético empregado neste trabalho, valores elevados da taxa de mutação levam a melhores resultados pois evitam a perda de diversidade da população. Cabe ressaltar que o algoritmo em uso é bastante robusto e pode ser utilizado com taxas de mutações elevadas. Este comportamento é função do elitismo empregado, que só permite que novos elementos gerados passem a fazer parte da população se o seu desempenho for superior ao pior elemento da população. Por esta razão, os indivíduos que tiverem seu desempenho diminuído pela elevada mutação sofrida dificilmente serão admitidos na população;

• Faixa de variação dos pesos = [-6,6]. O mesmo utilizado no treinamento da rede *feedforward* (seção 6.4).

7.3.2 Codificação dos pesos da *RNN*

A codificação dos pesos da rede em um cromossomo pode ser vista nas figuras 7.1 e 7.2. Na figura 7.1 mostra-se os pesos estáticos, dinâmicos e os bias que compõem a *RNN* utilizada neste trabalho. As conexões com pesos estáticos apenas repassam a informação recebida na camada de entrada para a camada recorrente, as conexões com pesos dinâmicos são as responsáveis pela capacidade de representação dinâmica da rede e o bias modifica a localização do ponto de inflexão da sigmóide (função de ativação dos neurônios), sendo que, neste ponto concentra-se a região de maior sensibilidade de um neurônio. Na figura 7.2, os pesos podem ser vistos na forma de um cromossomo.

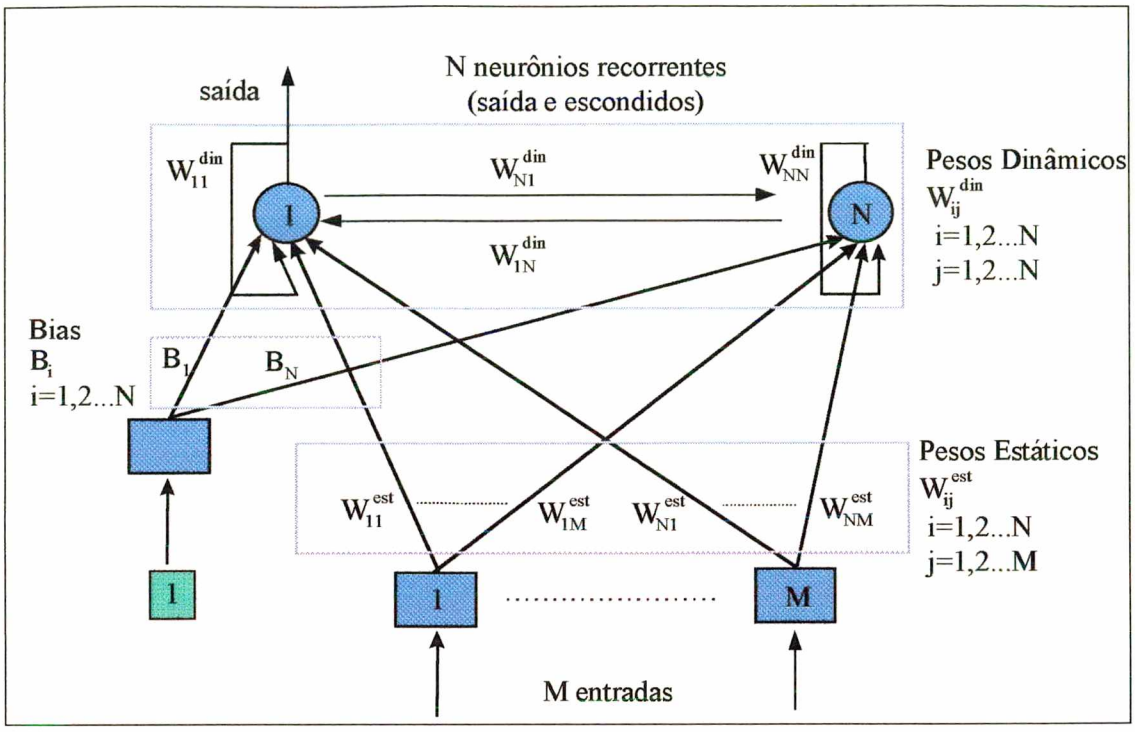

Figura 7. 1 - Arquitetura da *RNN*

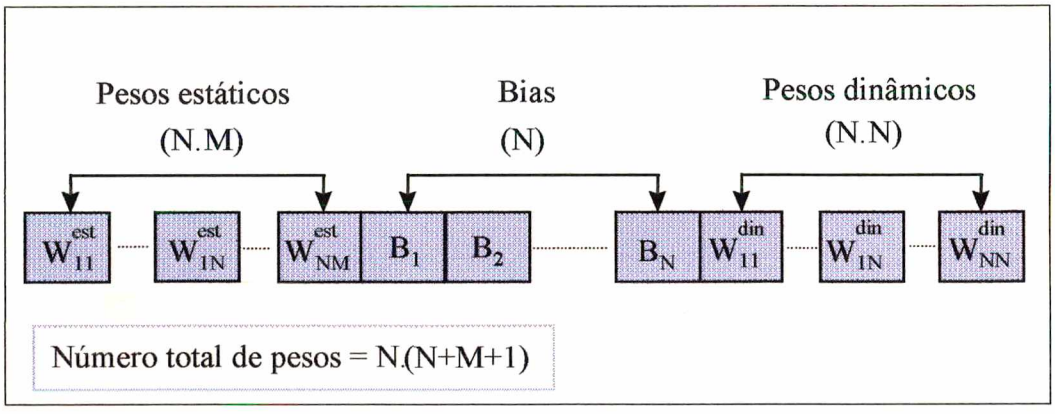

Figura 7. 2 – Codificação da *RNN* na forma de um cromossomo

7.4 Software Desenvolvido para Treinamento da *RNN*

Para o treinamento e teste da *RNN* foi desenvolvido um *software* que apresenta uma interface gráfica de fácil entendimento. A tela principal com alguns parâmetros significativos pode

ser vista na figura 7.3. Este *software* permite um acompanhamento gráfico da evolução do treinamento e, possibilita a intervenção do usuário para alterar operadores e parâmetros do algoritmo genético.

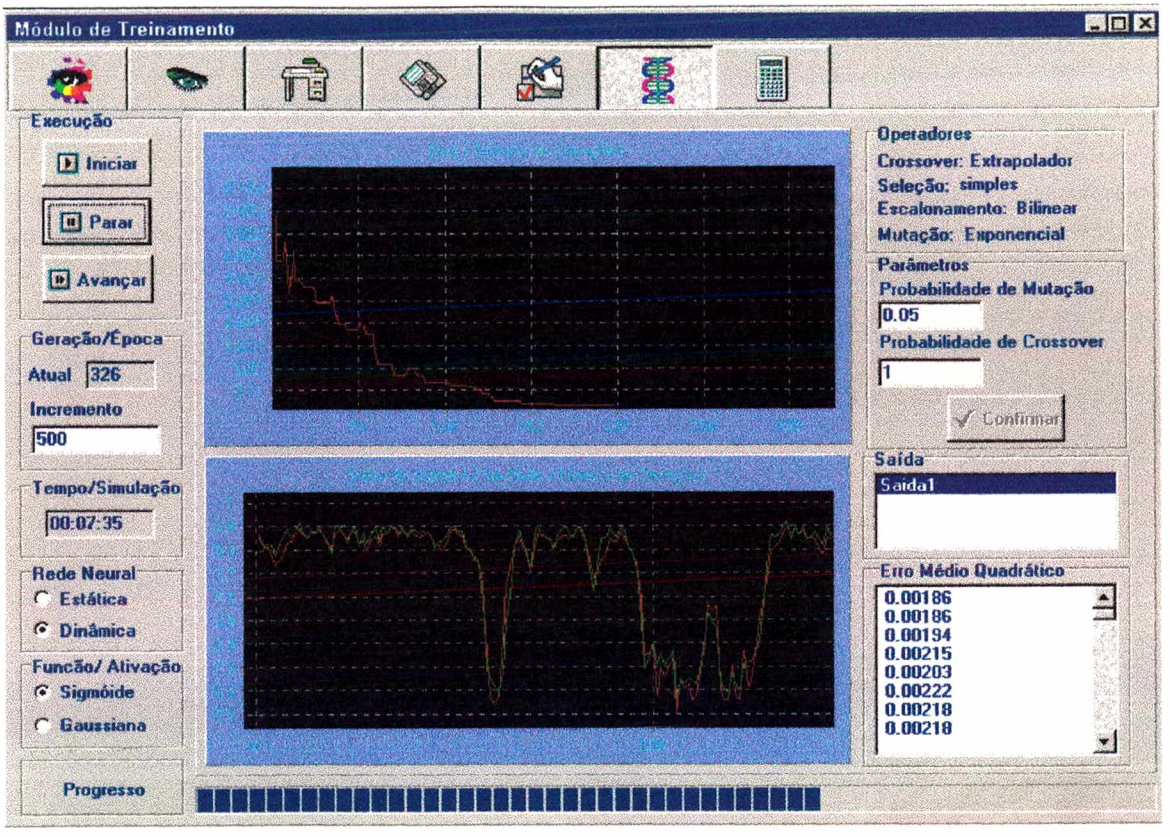

Figura 7. 3 – Software para treinamento da *RNN* - Tela Principal

Todas as simulações foram efetuadas em um microcomputador Pentium® 166 MHz com 48 MGBytes de memória RAM e 4,2 GBytes de disco rígido. O *software* foi dividido em dois módulos, um de treinamento e outro de teste.

Na fase de treinamento efetuam-se as seguintes etapas:

- Leitura do arquivo contendo as perturbações externas (entradas) e as respectivas saídas;
- Definição do número de entradas, neurônios escondidos e de saídas;
- Definição dos operadores e parâmetros do algoritmo genético;
- Definição das faixas de normalização das entradas e saídas.

Após realizado o treinamento pode-se testar a capacidade de representação da *RNN*. O conjunto de pesos utilizado na fase de teste pode ser obtido de duas formas: diretamente do treinamento que corresponde, neste caso, ao indivíduo da população com o melhor desempenho (menor erro quadrático de treinamento) ou, carregado de arquivo. A tela do módulo de teste pode ser vista na figura 7.4.

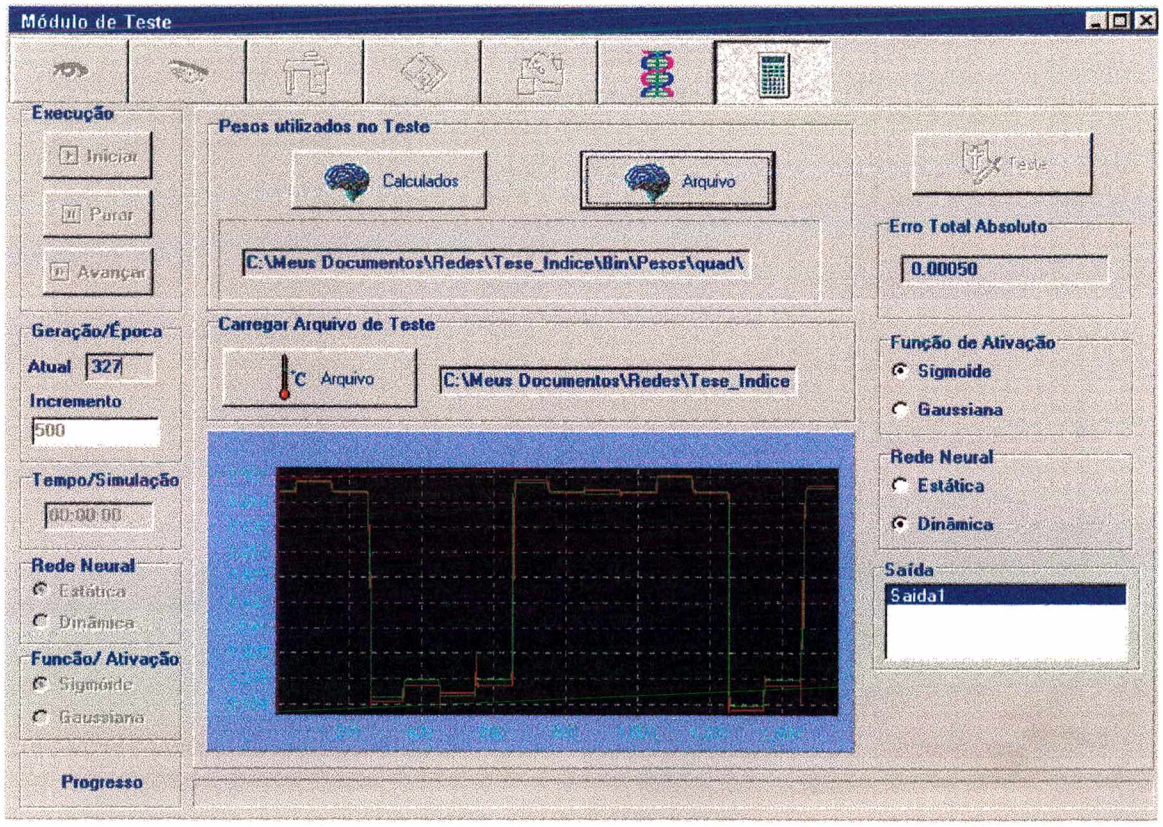

Figura 7. 4 - Software para treinamento da *RNN* - Módulo de Teste

Na fase de teste da *RNN* carrega-se um arquivo que relaciona as saídas do processo às perturbações externas, sendo que, esse deve ser diferente daquele utilizado na etapa de treinamento. O *software* aplica os valores de perturbações externas contidas no arquivo de teste como entradas a rede neural. A *RNN* fornecerá saídas que podem então ser comparadas com as saídas do processo presentes no arquivo de teste. A comparação das saídas da *RNN* e as reais pode ser observada graficamente e através do erro médio quadrático.

Cabe ressaltar que o intervalo de atuação da função de ativação sigmoidal está limitado entre *0* e *1* e portanto, as saídas também devem permanecer nesta faixa. Dessa forma procede-se a normalização dos dados conforme a equações 7.2:

$$Y = a*(y - y_{min})/ (y_{max} - y_{min}) + b \quad (7.2)$$

Onde: $(y - y_{min})/ (y_{max} - y_{min})$ é saída normalizada entre 0 e 1;

y_{min}, y_{max} - valor máximo e mínimo do grupo de dados para determinada saída;

a,b - constantes empregadas na normalização.

Para evitar a saturação dos neurônios, as saídas do processo normalmente não são normalizadas para o intervalo 0 a 1 mas, para um intervalo menor. Por esta razão são introduzidas as constantes *a* e *b*. Neste trabalho, foram utilizadas $a = 0.8$ *e* $b = 0.1$. Dessa forma qualquer saída ficará limitada entre 0.1 e 0.9.

No caso das entradas não há necessidade de normalização porém, este procedimento será efetuado, de acordo com a equação 7.3, para que a faixa de valores de todas as entradas fique semelhante:

$$I =(I - I_{min})/ (I_{max} - I_{min}) \quad (7.3)$$

Onde: I_{max}, I_{min} - valor máximo e mínimo do grupo de dados para uma determinada entrada.

7.5 Identificação de um *CSTR* com camisa de resfriamento

As equações que definem o comportamento do reator foram extraídas de ROQUEIRO & LIMA (1997) [43], mantendo inalterados os parâmetros e por conseguinte, os estados estacionários para os quais foram realizados os estudos. No trabalho de POTTMANN *et alli*

(1992) [59] é apresentada uma aplicação de redes neurais a este processo, sendo descrito como um *benchmark* na avaliação de sistemas de controle e modelos não lineares usados para identificação. O modelo do *CSTR* com camisa de resfriamento pode ser representado através de um sistema de 3 equações diferenciais ordinárias, sendo que, x_1 é a concentração de reagente, x_2 a temperatura no reator e x_3 a temperatura na camisa.

$$\frac{dx_1}{d\tau} = -\phi \cdot k \cdot x_1 + q \cdot (x_{1f} - x_1)$$

$$\frac{dx_2}{d\tau} = \beta \cdot \phi \cdot x_1 - (q + \delta) \cdot x_2 + \delta \cdot x_3 + q \cdot x_{2f}$$

$$\frac{dx_3}{d\tau} = \frac{q_c \cdot (x_{3f} - x_3)}{\delta_1} + \frac{\delta \cdot (x_2 - x_3)}{\delta_1 \cdot \delta_2}$$

$$k = \exp\left[\frac{x_2}{1 + x_2/\gamma}\right]$$

O processo é descrito nas variáveis adimensionais:

$$\beta = \frac{-\Delta H \cdot C_f \cdot \gamma}{\rho \cdot C_p \cdot T_{f0}} \qquad \phi = \frac{\mathrm{V} \cdot \mathrm{k}_0 \cdot \exp(-\gamma)}{Q_0} \qquad \mathrm{x}_2 = \frac{(T - T_{f0}) \cdot \gamma}{T_{f0}}$$

$$\gamma = \frac{\mathrm{E}}{R \cdot T_{f0}} \qquad \tau = \frac{\mathrm{Q}_0 \cdot t}{V} \qquad \mathrm{x}_{2\mathrm{f}} = \frac{(T_f - T_{f0}) \cdot \gamma}{T_{f0}}$$

$$\delta = \frac{U \cdot A}{\rho \cdot C_p \cdot Q_0} \qquad \mathrm{q} = \frac{\mathrm{Q}}{Q_0} \qquad \mathrm{x}_3 = \frac{(T_c - T_{f0}) \cdot \gamma}{T_{f0}}$$

$$\delta_1 = \frac{V_c}{V} \qquad \mathrm{x}_1 = \frac{\mathrm{C}}{C_{fo}} \qquad \mathrm{x}_{3\mathrm{f}} = \frac{(T_{cf} - T_{f0}) \cdot \gamma}{T_{f0}}$$

$$\delta_2 = \frac{\rho_c \cdot C_{pc}}{\rho \cdot C_p} \qquad \mathrm{x}_{1\mathrm{f}} = \frac{\mathrm{C}_\mathrm{f}}{C_{f0}}$$

A carga térmica é definida como:

$$m = q^* x_{2f}$$

com

ΔH	Calor de reação;
C	Composição no reator;
C_f	Composição de alimentação;
C_{f0}	Composição de referência de alimentação;
ρ	Densidade do meio racional;
ρ_c	Densidade do fluido refrigerante;
C_p	Capacidade calorífica do meio reacional;
C_{pc}	Capacidade calorífica do fluido refrigerante;
T	Temperatura do reator;
T_f	Temperatura de alimentação do reator;
T_c	Temperatura do fluido refrigerante;
T_{c0}	Temperatura de alimentação do fluido refrigerante;
E	Energia de ativação;
R	Constante universal dos gases;
U	Coeficiente de troca térmica;
A	Área de troca térmica;
V	Volume do reator;
V_c	Volume da camisa;
k_0	Fator de Freqüência;
Q	Fluxo de Alimentação;
Q_0	Fluxo de Alimentação de referência.

sendo os valores numéricos utilizados:

$\beta = 8.0$	$x_{1f} = 1.0$	$\phi = 0.072$	$x_{2f} = 0.0$
$\delta = 0.3$	$q = 1.0$	$\gamma = 20.0$	$x_3 = 0.0$
$\delta_1 = 0.1$	$\delta_2 = 0.1$	$x_{3f} = -1.0$	$q_c = 1.65102$

Os estados estacionários estáveis são ($x_1 = 0.8933$, $x_2 = 0.5193$, $x_3 = -0.5950$) e

(x_1= 0.1890, x_2= 5.1373, x_3 = -0.6359).

7.5.1 Problema MISO

O problema escolhido consiste na determinação de um modelo empírico baseado em uma *RNN* para predição da concentração do reagente (A) na saída do *CSTR*, sendo que, são aplicadas perturbações na vazão e na carga térmica.

As etapas para identificação do processo, no caso de uma rede neural, são as seguintes:

- Obtenção dos padrões de treinamento;
- Determinação da estrutura e estimação dos pesos da *RNN*;
- Validação do modelo obtido.

Os padrões de treinamento foram obtidos resolvendo o sistema de equações diferenciais que representa o processo a partir de um estado estacionário estável, utilizando o método de *Runge-Kutta* de 4-5 ordem com passo variável. O tempo de amostragem empregado foi de 0,25 segundos (recomendado no trabalho de TAHUATA (1992) [60]). As perturbações para a vazão (q) e a carga térmica (m) adimensionalizadas são degraus com amplitude de até 0.5 em torno dos valores adimensionalizados q = 1,0 e m= 0.0 com largura de 3 intervalos de amostragem (figuras 7.5 e 7.6). As perturbações são suficientes para provocar transições entre os estados estacionários e, junto com as saídas obtidas pela resolução das equações diferenciais (figura 7.7) formam o grupo de dados utilizado no treinamento.

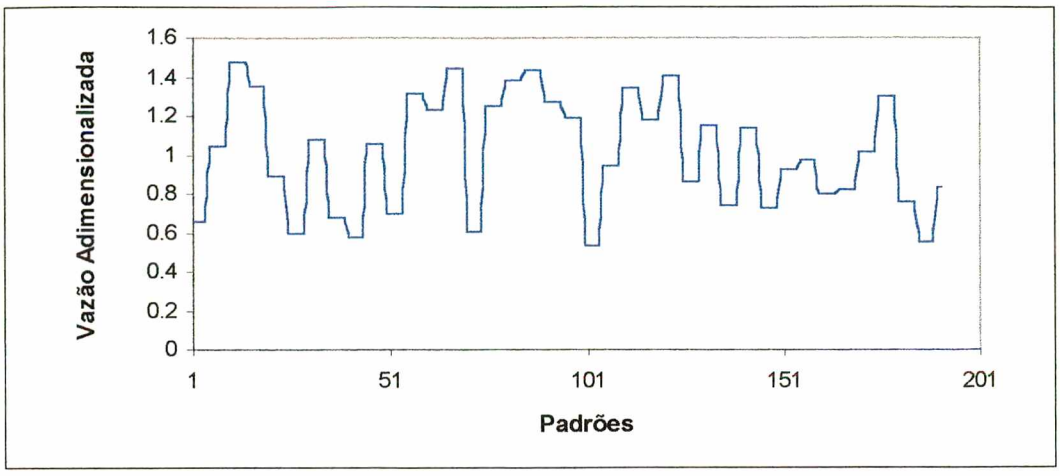

Figura 7. 5 - Perturbações aplicadas na vazão adimensionalizada

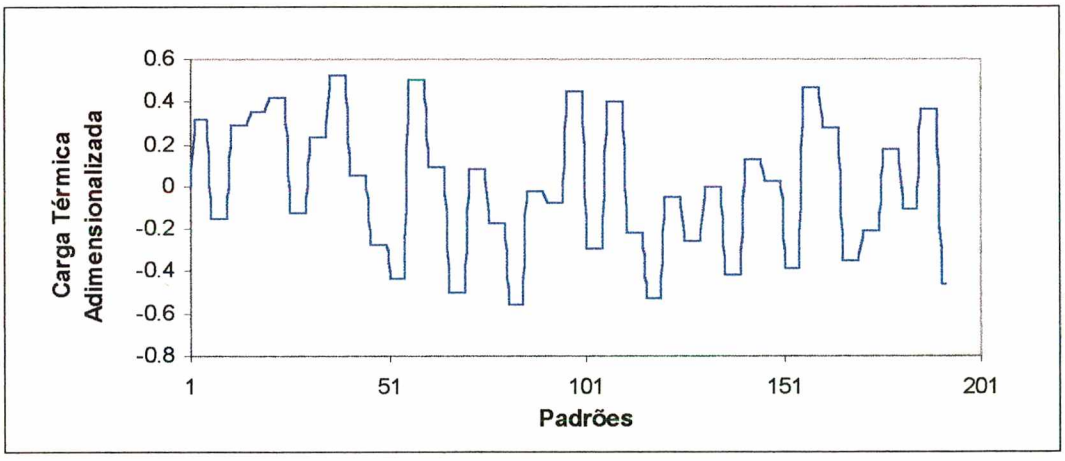

Figura 7. 6 - Perturbações aplicadas na Carga Térmica adimensionalizada

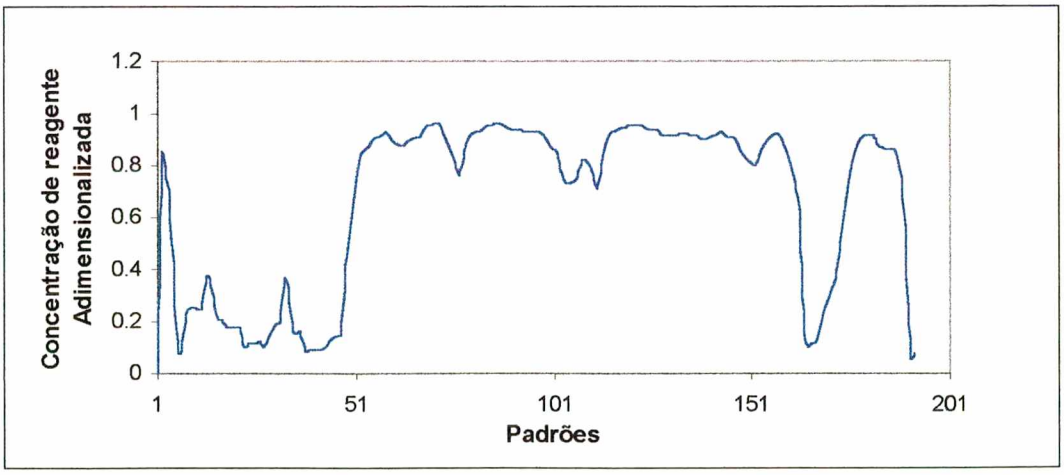

Figura 7. 7 - Concentração de reagente na saída do *CSTR* para as entradas mostradas nas figuras 7.5 e 7.6.

7.5.2 Determinação da Estrutura da *RNN*

Após obtido o grupo de dados de treinamento, deve-se determinar a arquitetura da rede, ou seja, definir o número de neurônios de entrada, escondidos e de saída.

A quantidade de entradas da rede corresponde aos valores atuais das perturbações e um determinado número, a ser determinado, de valores atrasados dessas. Assim, além dos valores atuais da carga térmica e da vazão, deve-se determinar o número adequado de atrasos para estas duas perturbações.

Em relação ao número de neurônios escondidos, não há nenhum método para determinar o valor adequado. Portanto, deve-se variar a quantidade de neurônios escondidos até atingir o resultado desejado (em termos do erro quadrático de aproximação).

O número de neurônios de saída é igual a quantidade de variáveis a serem preditas, que neste caso é a concentração de reagente na saída do reator.

Dessa forma, faltam determinar o número de atrasos nas perturbações aplicadas e o número de neurônios escondidos. Para determinar o número de entradas da rede, admitiu-se inicialmente um valor significativo para o número de neurônios escondidos. Posteriormente, após definido o número de entradas, foi feito um estudo para determinação do número de neurônios escondidos.

O critério de discriminação de modelos utilizado escolhe a menor estrutura possível a partir da qual o erro médio quadrático de treinamento não diminua significativamente.

7.5.2.1 Número de entradas a rede

Os experimentos realizados nesta seção tem por objetivo principal verificar a influência do número de atrasos em cada perturbação no erro médio quadrático de treinamento. Também mostra-se um estudo da convergência do algoritmo genético em função do número de pesos e atrasos da *RNN*.

Para determinar um número inicial de neurônios escondidos procurou-se na literatura algum trabalho envolvendo a identificação de um *CSTR* com uma *RNN*. O trabalho de YOU & NIKOLAOU (1993) [24], que trata da identificação de um *CSTR* com reação de primeira ordem, utiliza uma *RNN* com neurônios sigmóides igual à empregada neste trabalho. A diferença entre este trabalho e o de YOU & NIKOLAOU, é que no último foi identificada a dinâmica em torno de apenas um dos estados estacionários estáveis. Conforme YOU & NIKOLAOU foram necessários 4 neurônios para identificação do sistema. Dessa forma, a arquitetura inicial da *RNN* foi composta de quatro neurônios escondidos, um de saída (concentração de reagente na saída do *CSTR*) e as entradas foram formadas pelas perturbações na vazão e na carga térmica, mais um número de atrasos nessas a ser determinado. Para cada perturbação utilizou-se o mesmo número de atrasos.

Fixado o número inicial de neurônios escondidos, empregou-se a seguinte metodologia para determinação do número de atrasos em cada perturbação: Variando o número de atrasos entre 0 e 2, treinou-se, para cada um desses casos, 10 *RNNs*. Em cada situação anotou-se o erro médio quadrático de treinamento e o número de gerações para convergência do algoritmo genético. Para determinar quando o algoritmo genético convergiu utilizaram-se dois critérios:

- Mudança pouco significativa do erro médio quadrático de treinamento;
- Tendência de grande aproximação da rede em algumas regiões em detrimento a outras.

O resultado dos experimentos pode ser visto na tabela 7.1. Nesta mostra-se o mínimo e a média dos erros médios quadráticos de treinamento para cada atraso, o número de pesos *RNN* treinada e o mínimo e a média dos números de gerações para convergência do algoritmo genético.

Tabela 7.1- Desempenho do algoritmo genético na otimização dos pesos *RNN* em relação ao número de atrasos

Rede Recorrente				*Algoritmo Genético*	
N° de atrasos	*Erro médio quadrático*		*N° pesos*	*Número de Gerações*	
	Média	*Menor*		*Média*	*Menor*
0	0,00382	0,00299	35	143	136
1	0,00301	0,00241	45	176	162
2	0,00283	0,00226	55	185	166

Nota-se na tabela 7.1 que o erro médio quadrático varia pouco de um para dois atrasos em relação a variação de nenhum para um atraso. Este comportamento pode ser melhor visto na figura 7.8, que mostra a influência do número de atrasos na média do erro médio quadrático de treinamento.

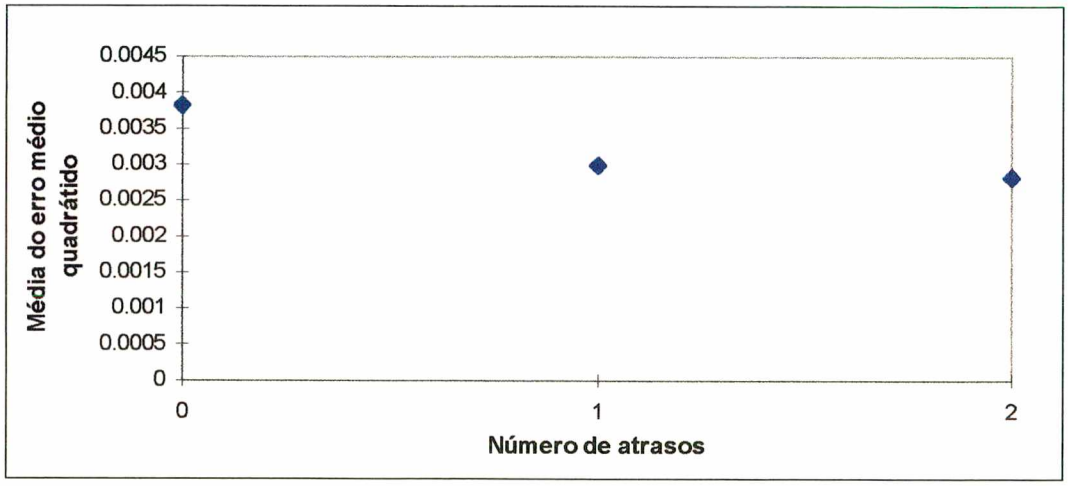

Figura 7. 8 - Média do erro médio quadrático em relação ao número de atrasos nas perturbações aplicadas

Examinando a figura 7.8, nota-se que a partir de um atraso o erro médio quadrático decai muito pouco e por esta razão, foi escolhido um valor de *um* atraso por perturbação. Dessa forma

o número escolhido de entradas da rede foi de quatro (uma entrada para a perturbação na vazão de alimentação, outra para seu atraso e o mesmo para a carga térmica).

Nas figura 7.9 pode ser visto o número de gerações necessários a convergência do algoritmo genético em relação ao número de atrasos. Na figura 7.10, mostra-se o número de gerações em função do número de pesos da *RNN*.

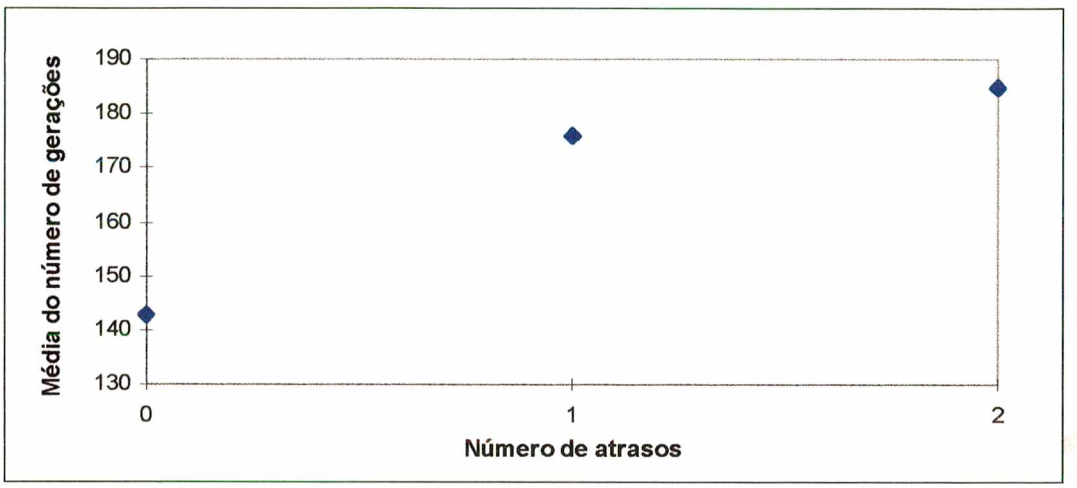

Figura 7. 9 – Média do número de gerações para convergência do algoritmo genético x número de atrasos nas perturbações aplicadas

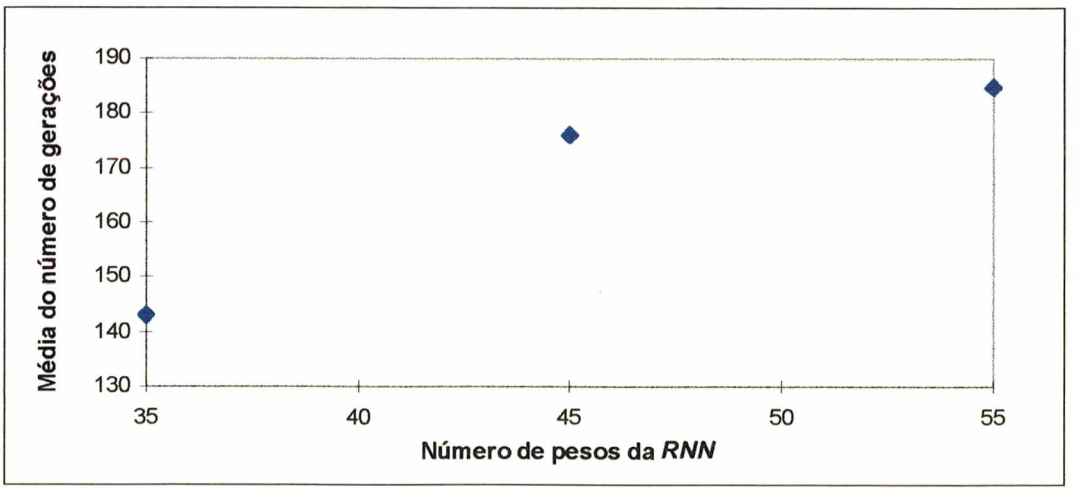

Figura 7. 10 – Média do número de gerações para convergência do algoritmo genético x número de pesos da *RNN*

Analisando as figuras 7.9 e 7.10, percebe-se um aumento pouco expressivo do número de gerações para convergência do algoritmo genético a partir de um atraso. Conclui-se que um aumento das dimensões da rede implica apenas em um pequeno acréscimo do número de gerações.

7.5.2.2 Número de neurônios escondidos

Para determinar o número ótimo de neurônios escondidos variou-se o número desses entre 1 e 5. Em cada caso, treinaram-se 10 *RNNs*. O resultado dos experimentos pode ser visto na tabela 7.2, que mostra a evolução do erro quadrático de treinamento e do número de gerações para convergência do algoritmo genético em relação ao número de neurônios escondidos.

Tabela 7.2 - Desempenho do algoritmo genético na otimização dos pesos da *RNN* em relação ao número de neurônios escondidos

Rede Recorrente				*Algoritmo Genético*	
Neurônios escondidos	*Erro quadrático médio*		*Nº pesos*	*Número de Gerações*	
	Média	*Menor*		*Média*	*Menor*
1	0,00592	0,00318	12	68	41
2	0,00370	0,00329	21	117	111
3	0,00349	0,00295	32	108	96
4	0,00350	0,00300	45	131	110
5	0,00383	0,00269	60	163	109

Na figura 7.11 mostra-se a média do erro médio quadrático de treinamento em relação ao número de neurônios escondidos.

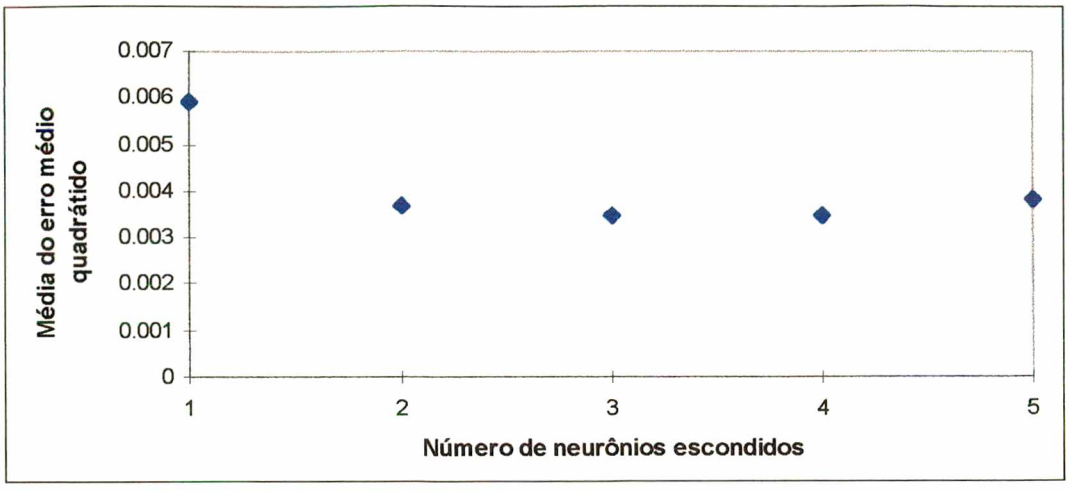

Figura 7. 11 - Média do erro médio quadrático $_x$ número de neurônios escondidos

Analisando a figura 7.11, pode-se verificar que o erro quadrático diminui bastante até 2 neurônios escondidos, decresce pouco até quatro e a partir desse começa a aumentar. Foi escolhido um número de neurônios escondidos igual a 3 pois, a partir deste ponto, o erro não diminui. Dessa forma, a arquitetura da *RNN* obtida do estudo feito possui: quatro entradas, três neurônios escondidos e um de saída.

Nas figuras 7.12 e 7.13, pode ser visto o número de gerações necessárias à convergência em relação ao número neurônios escondidos e ao número de pesos da *RNN* respectivamente.

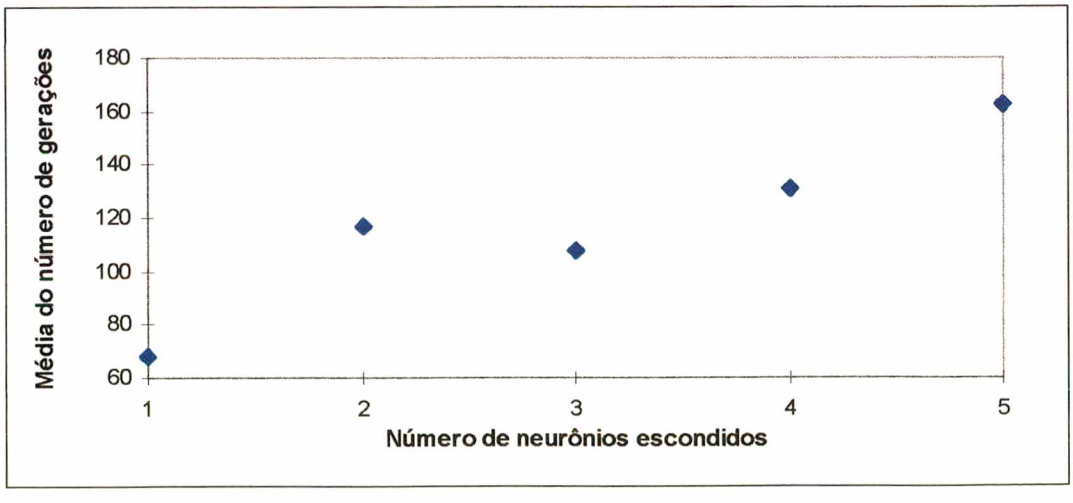

Figura 7. 12 – Média do número de gerações para convergência do algoritmo genético $_x$ número de neurônios escondidos

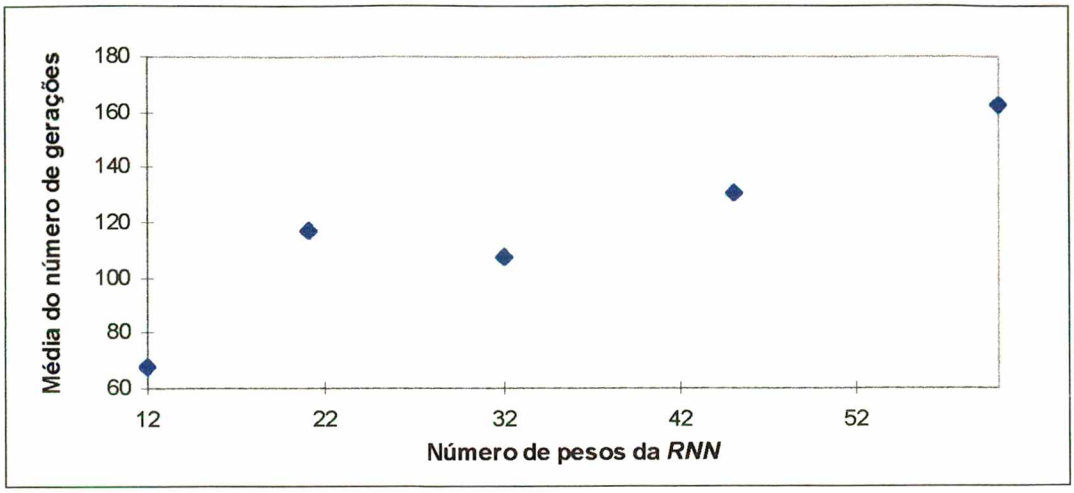

Figura 7. 13 – Média do número de gerações para convergência do algoritmo genético x número de pesos da rede Recorrente

Nas figuras 7.12 e 7.13 percebe-se que o número de gerações, para convergência do algoritmo genético, aumenta até dois neurônios escondidos, diminui em três e aumenta, de forma aproximadamente linear, a partir desse ponto.

7.5.3 Simulação com o modelo do *CSTR*

Para avaliar a capacidade de representação dinâmica da *RNN*, utilizou-se um grupo de dados diferente do empregado para treinamento. O grupo de dados para teste da *RNN* consiste de perturbações efetuadas na vazão (figura 7.14) e, na carga térmica (figura 7.15). A saída do sistema (concentração de reagente na saída do *CSTR*) para as perturbações aplicadas é mostrada na figura 7.16.

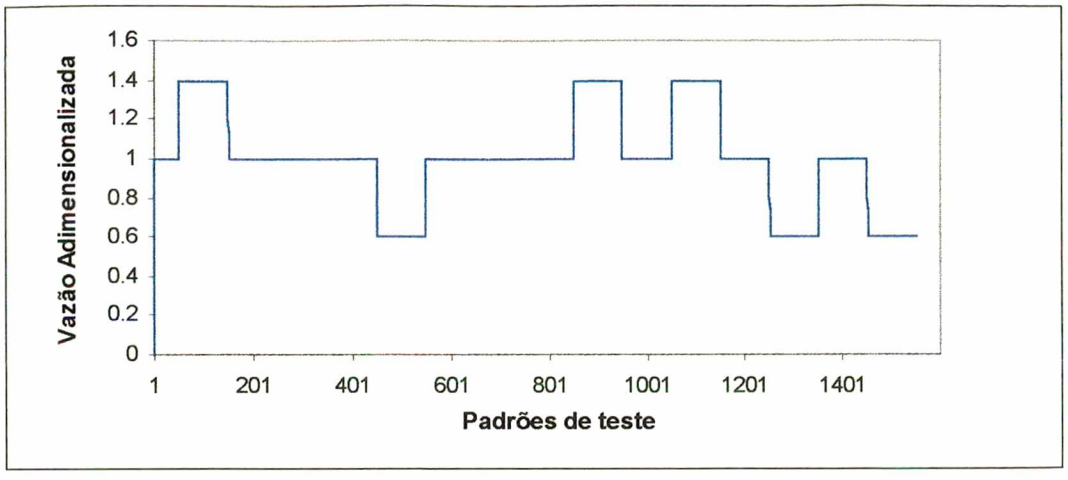

Figura 7. 14 – Perturbações aplicadas na vazão de alimentação adimensionalizada

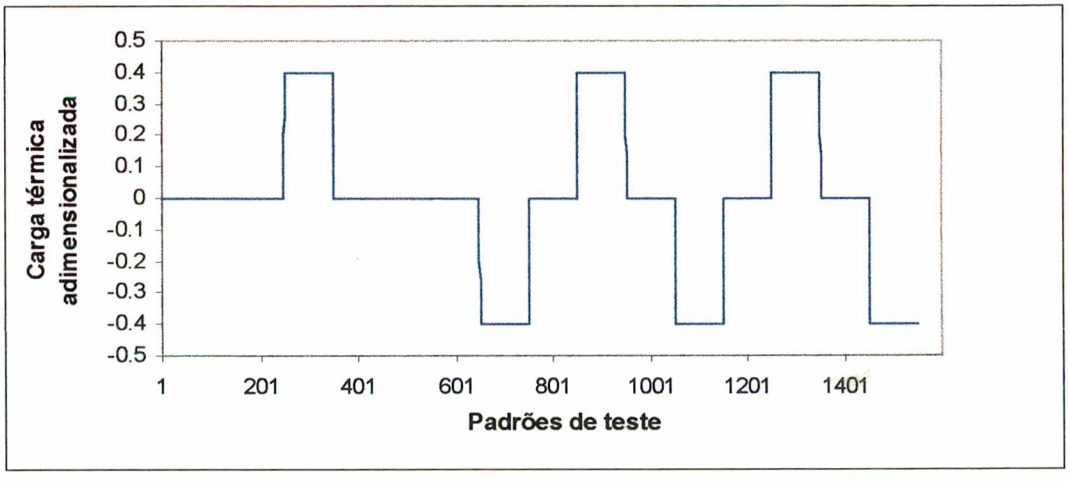

Figura 7. 15- Perturbações aplicadas na carga térmica adimensionalizada

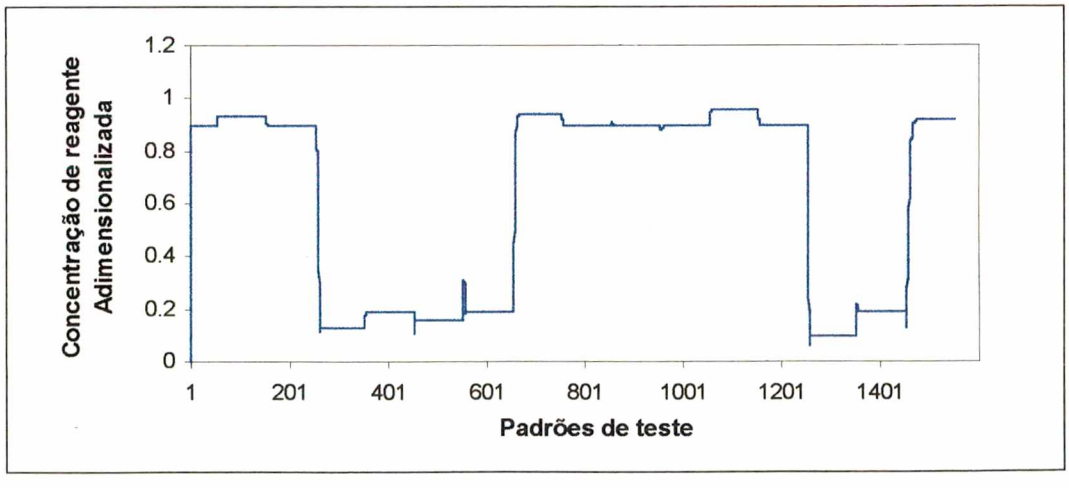

Figura 7. 16 – Concentração de reagente na saída do *CSTR* para as entradas mostradas figuras 7.14 e 7.15

Nas figuras 7.14 e 7.15 nota-se que há todas as combinações possíveis de perturbações degrau. Além disso, os degraus nas entradas externas tem amplitude suficiente para modificar o estado estacionário que corresponde a entradas sem perturbações, isto é, após a perturbação, o processo pode se estabelecer em qualquer um dos dois estados estacionários estáveis. As perturbações para a vazão e a para carga térmica adimensionalizadas tem amplitude de 0.4 em torno dos valores q = 1,0 e m= 0.0.

Na figura 7.16 observam-se as transições entre estados estacionários e a dinâmica em torno de cada estado estacionário. Estas características podem ser observadas claramente no grupo de teste devido a baixa freqüência de mudança dos degraus (50 intervalos de amostragem, no treinamento utilizaram-se 3 intervalos de amostragem).

Na figura 7.17 compara-se a saída da *RNN* e a saída real (figura 7.17). A saída da *RNN* foi obtida aplicando as perturbações na vazão e na carga térmica mostradas nas figuras 7.14 e 7.15. O erro quadrático do teste foi de 0.00050.

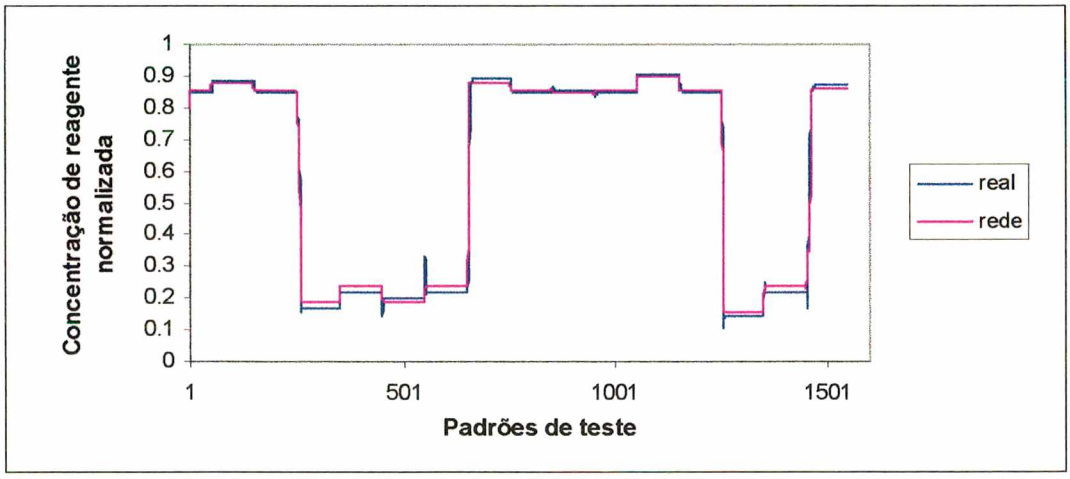

Figura 7. 17 - Comparação entre a saída da rede e a real para a concentração de reagente na saída do *CSTR*

Pode ser observado na figura 7.17, que o modelo neural reproduziu razoavelmente bem a multiplicidade de estados estacionários e as dinâmicas nas vizinhanças desses.

A identificação de um processo com características não lineares e multiplicidade de estados estacionários, como é caso do *CSTR*, não pode ser realizada através do ajuste de modelos lineares. Detalhes podem ser vistos em EMBIRUÇU (1993) [61] que utilizou um modelo linear multivariável.

7.6 Identificação de um Bioreator

Nesta seção é apresentada a identificação de um bioreator onde ocorre um processo de fermentação. O processo é descrito por três equações diferenciais ordinárias de primeira ordem, correspondendo aos balanços de células(x), substrato(s) e produto (p). As equações, os valores dos parâmetros e o estado estacionário do processo foram extraídos do trabalho de HENSON & SEBORG (1991) [62].

$$\dot{x} = -dx + \mu x$$

$$\dot{s} = -d(s_f - s) - \frac{1}{y_{x/s}}\mu x$$

$$\dot{p} = -dp + (\alpha\mu + \beta)x$$

$$\mu = \frac{\mu_m\left(1 - p/p_m\right)s}{k_m + s + s^2/k_i}$$

Onde:

d - taxa de diluição;

s_f - concentração de substrato na alimentação - variável entrada;

μ - taxa específica de crescimento;

$y_{x/s}$ - rendimento em células/substrato consumido;

α e β - parâmetros cinéticos;

μ_m- taxa de crescimento específica máxima;

p_m - constante de saturação do produto;

k_m- constante de saturação do substrato;

k_i - constante de inibição do substrato.

Parâmetros:

$y_{x/s}$ 0.4 g/g;

α 2.2 g/g;

β $0.2h^{-1}$;

μ_m $0.48h^{-1}$;

p_m 50 g/l;

k_m 1.2g/l;

k_i 22 g/l;

s_f 20g/l (variável de entrada);

d 0.202g/l (variável de entrada).

Estado estacionário:

x 6.0 g/l;

p 19.14 g/l;

s 5.0 g/l (variável de saída).

Para resolução do sistema de equações foi utilizado o método *Runge-Kutta* 4-5 ordem com passo variável. O processo a ser identificado tem duas entradas (vazão de alimentação e concentração de substrato na alimentação) e uma saída (concentração de substrato na saída do bioreator).

7.6.1 Problema MISO

O problema consiste na identificação da concentração de substrato na saída do bioreator, dado um conjunto de perturbações aplicadas na vazão e na concentração de substrato da alimentação.

Para o treinamento da *RNN*, gerou-se um grupo de dados a partir da resolução das equações diferenciais do bioreator, e de um conjunto de perturbações na vazão e na concentração de substrato na alimentação. Estas perturbações são degraus com amplitude de até 20 % em torno do estado estacionário (s_f = 20 g/l e d = 0.202 l/h), e largura de 3 intervalos de amostragem (ver Figuras 7.18 e 7.19). A concentração de substrato na saída do bioreator para as perturbações efetuadas pode ser vista na figura 7.20.

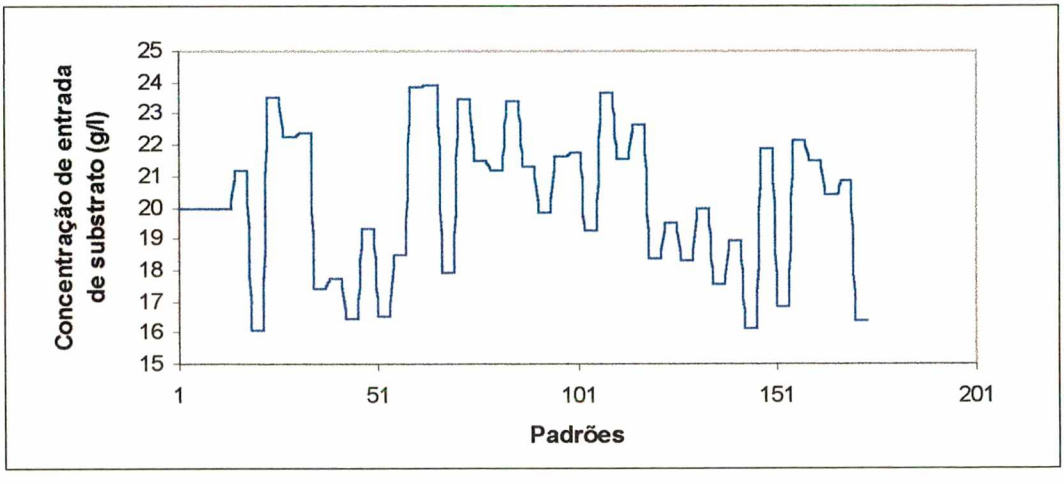

Figura 7. 18 - Perturbações aplicadas na concentração de substrato alimentado ao bioreator

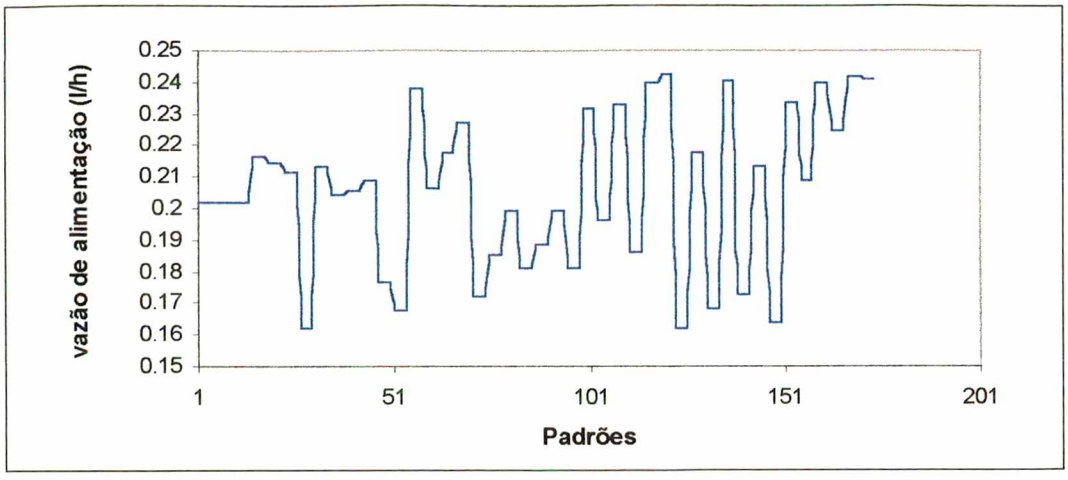

Figura 7. 19 - Perturbações aplicadas na vazão de alimentação

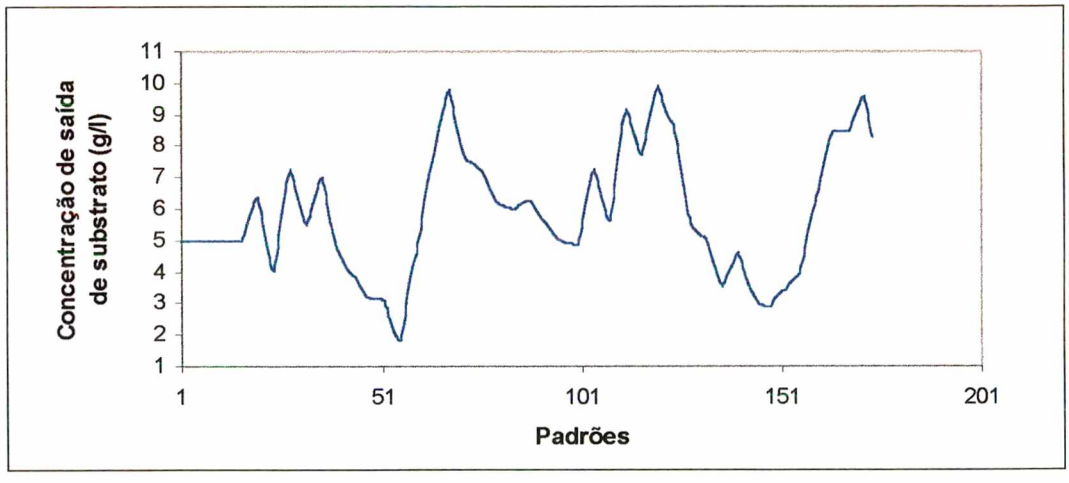

Figura 7. 20 – Concentração de substrato na saída do bioreator para as perturbações mostradas nas figuras 7.18 e 7.19

7.6.2 Determinação da Estrutura da Rede

Para determinação da melhor estrutura da *RNN* foi utilizado o mesmo procedimento efetuado para o *CSTR*. Inicialmente, determinou-se o número de atrasos para cada perturbação fixando o número de neurônios escondidos. Posteriormente, foi efetuado um estudo para a determinação do número adequado de neurônios escondidos.

7.6.2.1 Número de entradas a rede

Na determinação do número de entradas da rede, foi admitido inicialmente um número de neurônios escondidos igual a três. Portanto, a estrutura inicial da *RNN* utilizada no treinamento foi composta de um neurônio de saída, três neurônios escondidos e de um número de atrasos, a ser determinado, nas perturbações aplicadas.

Para determinar o número de entradas variou-se o número de atrasos entre 0 e 2. Para cada atraso, foram efetuados 10 ensaios anotando-se o erro médio quadrático entre a saída da rede e a saída real, e o número de gerações necessárias para a convergência do algoritmo genético. O critérios de parada para o algoritmo genético são os mesmos que os utilizados no *CSTR*.

Os resultados finais da simulação podem ser vistos na tabela 7.3. Nesta pode-se notar que o erro médio quadrático de treinamento atinge o seu mínimo para um atraso.

Tabela 7.3 - Desempenho do algoritmo genético na otimização dos pesos *RNN* em relação ao número de atrasos

Rede Recorrente				*Algoritmo Genético*	
N° de atrasos	*Erro quadrático médio*		*N° pesos*	*Número de Gerações*	
	Média	*Menor*		*Média*	*Menor*
0	0,00115	0,00080	24	254	243
1	0,00045	0,00032	32	245	154
2	0,00073	0,00059	40	282	232

Nas figura 7.21 mostra-se o mínimo e a média dos erros médios quadráticos de treinamento em função dos atrasos. Examinando a figura percebe-se que *um* atraso corresponde ao valor ótimo. Dessa forma o número de entradas para *RNN* será quatro, duas entradas consistem dos valores atuais das perturbações na vazão e na concentração de substrato e as outras duas são os atrasos defasados em um intervalo de amostragem dessas.

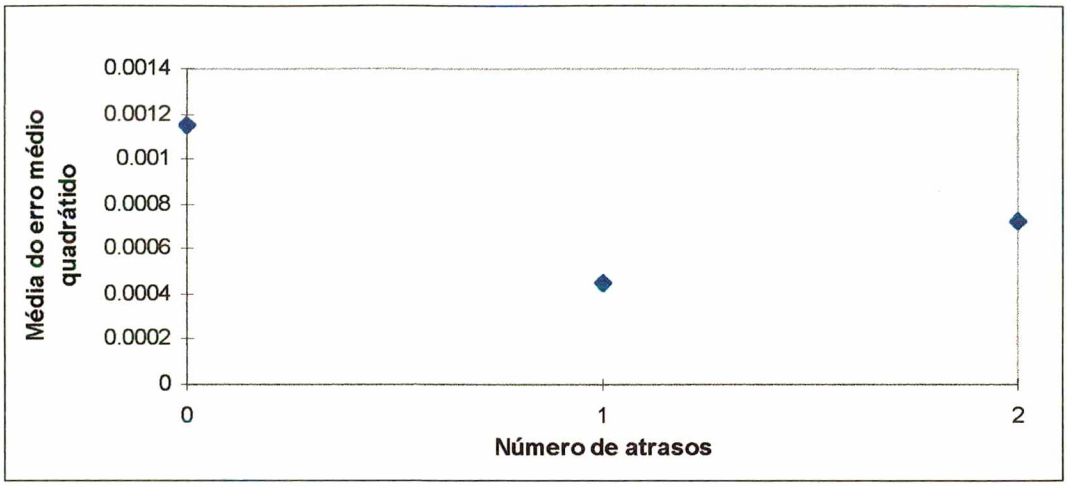

Figura 7. 21 -Média do erro médio quadrático $_x$ número de atrasos nas perturbações aplicadas

Nas figuras 7.22 e 7.23 pode ser visto o número de gerações necessárias à convergência do algoritmo genético em relação ao número de atrasos e de pesos respectivamente.

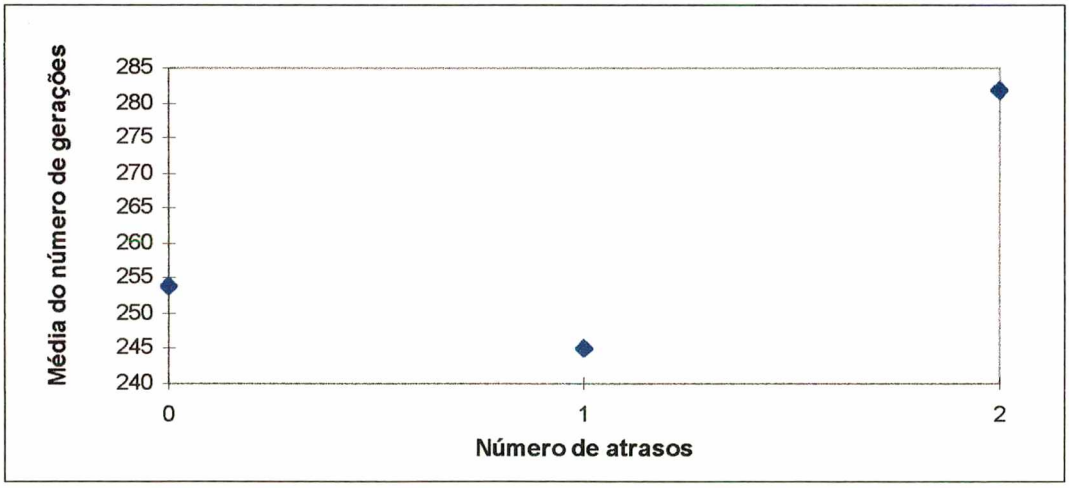

Figura 7. 22 – Média do número de gerações para convergência do algoritmo genético $_x$ número de atrasos nas perturbações aplicadas

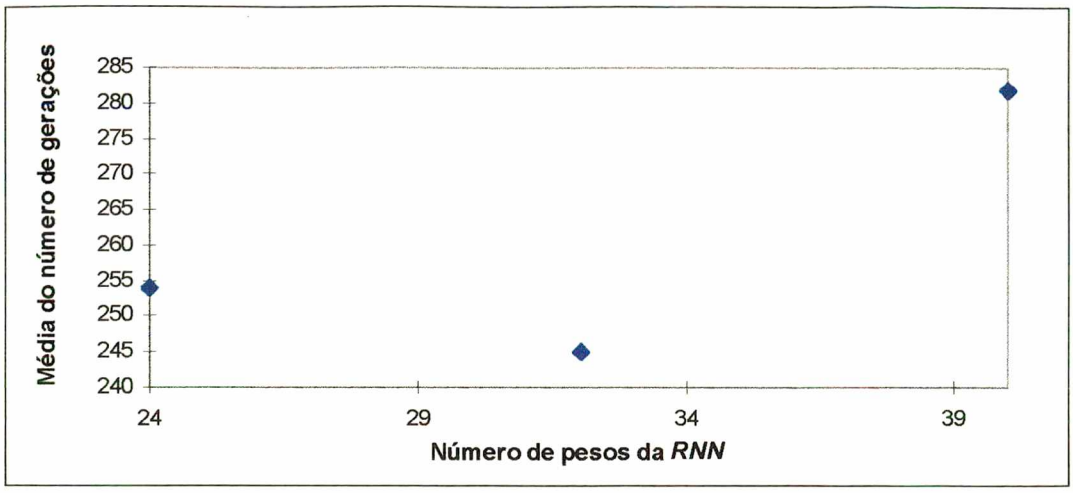

Figura 7. 23 – Média do número de Gerações para convergência do algoritmo genético $_x$ número de pesos da *RNN*

Analisando as figuras 7.22 e 7.23, nota-se um aumento pouco expressivo do número de gerações para convergência do algoritmo genético.

7.6.2.2 Determinação do número de neurônios escondidos

Para determinar o número de neurônios escondidos foi realizado um estudo variando a quantidade desses entre 1 e 5. Em cada caso foram efetuados 10 ensaios anotando-se o mínimo e a média dos erros médios quadráticos de aproximação, e o número de gerações necessárias para convergência do algoritmo genético.

Na tabela 7.4 mostra-se os erros quadráticos de cada estrutura de *RNN*, além do número de gerações necessárias para convergência do algoritmo genético.

Tabela 7.4- Desempenho do algoritmo genético na otimização dos pesos da *RNN* em relação ao número de neurônios escondidos

Rede Recorrente				*Algoritmo Genético*	
Neurônios escondidos	*Erro quadrático médio*		*N°pesos*	*Número de Gerações*	
	Média	*Menor*		*Média*	*Menor*
1	0,00131	0,00101	12	208	150
2	0,00092	0,00050	21	240	186
3	0,00049	0,00041	32	231	185
4	0,00046	0,00035	45	311	303
5	0,00050	0,00044	60	321	311

Analisando a tabela 7.4 percebe-se que o erro quadrático médio de treinamento diminui consideravelmente até 3 neurônios. Para analisar melhor o comportamento do erro médio quadrático mostra-se na figura 7.24 a média desse em função do número de neurônios escondidos.

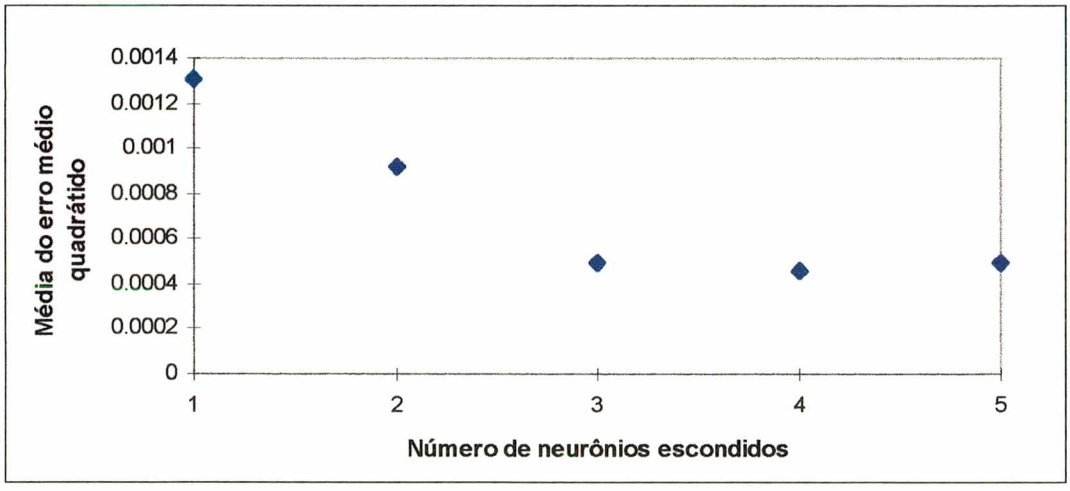

Figura 7. 24 - Média do erro médio quadrático $_x$ número de neurônios escondidos

Na figura 7.24 pode-se verificar que o erro quadrático diminui até 3 neurônios escondidos, permanece constante até quatro e começa aumentar a partir deste ponto. Dessa forma, o número ótimo de neurônios escondidos, segundo o critério definido, é três.

Nas figuras 7.25 e 7.26 mostra-se a influência do número de neurônios e pesos na convergência do algoritmo genético.

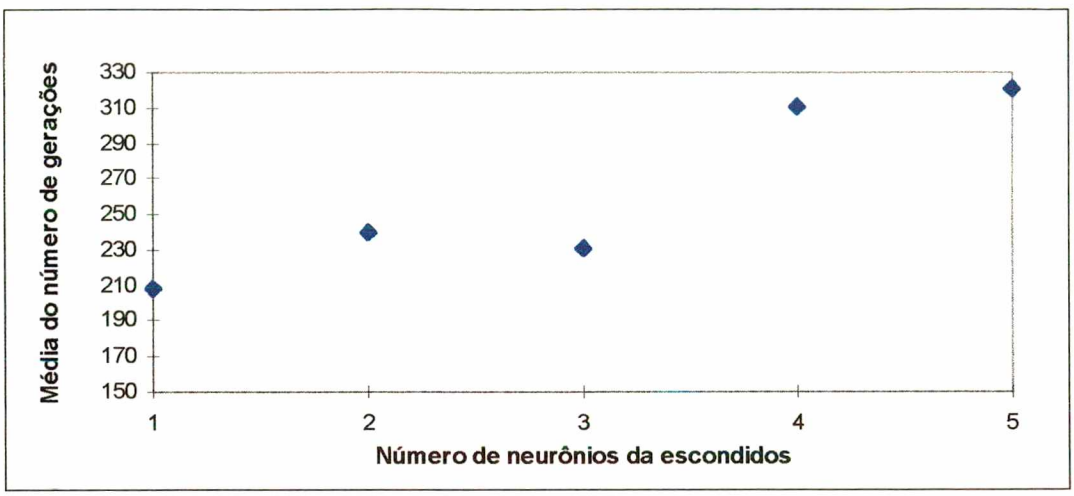

Figura 7. 25 - Média do número de gerações para convergência do algoritmo genético $_x$ número de neurônios escondidos

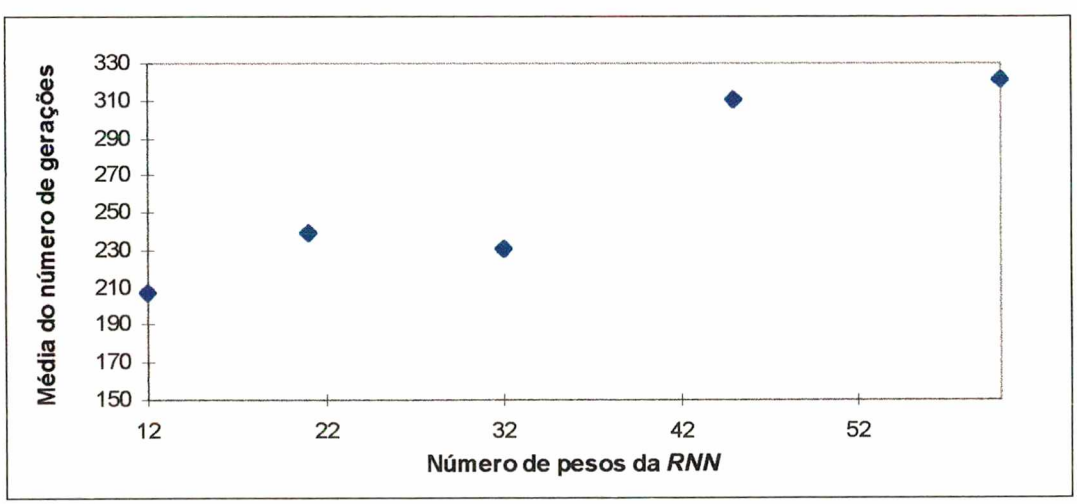

Figura 7. 26 – Média do número de gerações para convergência do algoritmo genético $_x$ número de pesos da *RNN*

Nas figuras 7.25 e 7.26 percebe-se que o número mínimo de gerações necessárias para a convergência do algoritmo foi obtido para uma *RNN* com 3 neurônios escondidos. Esta estrutura é também a que obteve o menor erro quadrático de treinamento. Dessa forma, aumentar o número de neurônios escondidos não necessariamente irá melhorar o desempenho da *RNN*.

Pode-se concluir do estudo feito, que a estrutura ideal a identificação do bioreator é composta de quatro entradas, três neurônios escondidos e um neurônio de saída.

7.6.3 Simulação com o modelo do Bioreator

Para simulação do bioreator através da *RNN* gerou-se um grupo de dados formado de perturbações degrau na concentração de substrato na entrada do reator e na vazão de alimentação (figuras 7.27 e 7.28). As perturbações efetuadas na vazão e na concentração de substrato têm amplitude de 8% em torno dos valores correspondentes ao estado estacionário (s_f = 20 e d = 0.202). Aplicando essas perturbações no modelo fenomenológico obtém-se a concentração de substrato na saída do bioreator (figura 7.29).

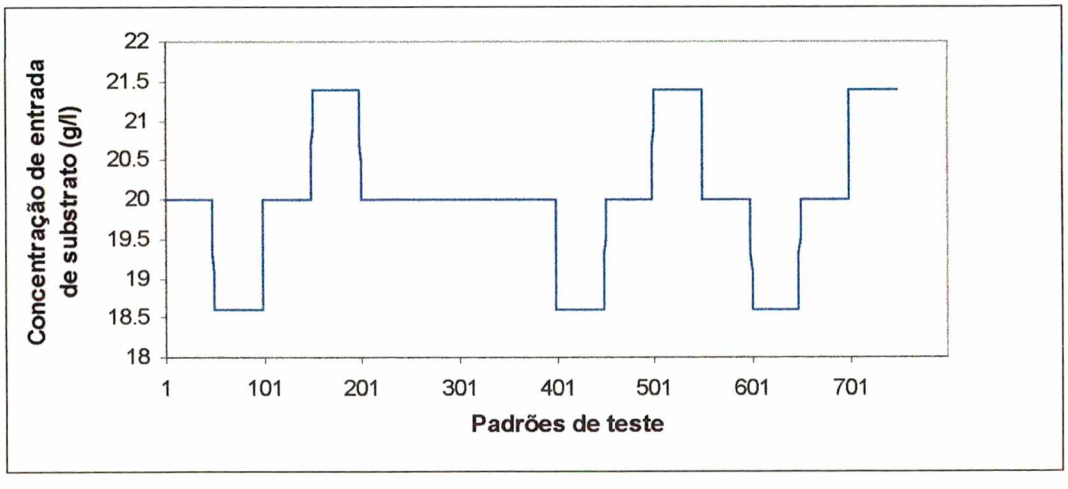

Figura 7. 27 - Perturbações aplicadas na concentração de substrato alimentado ao bioreator

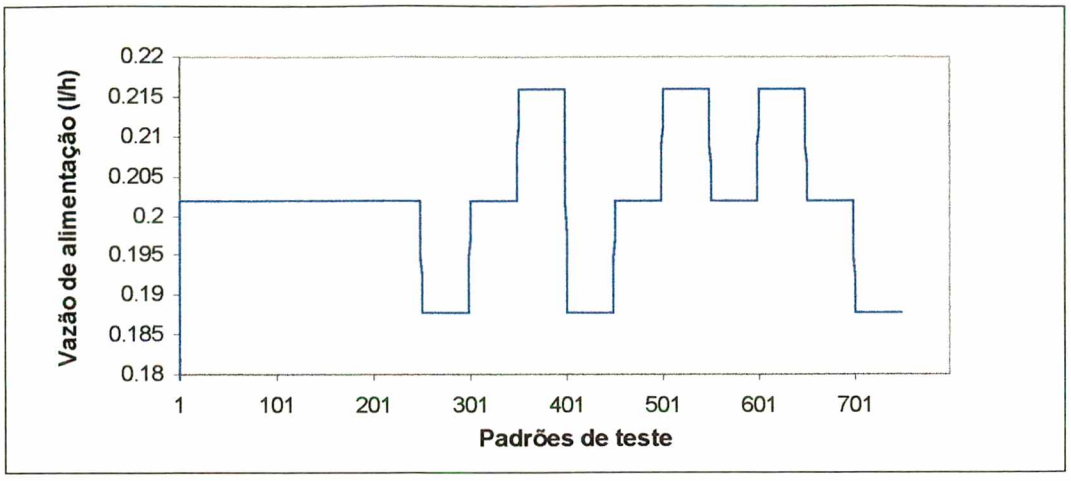

Figura 7. 28 – Perturbações aplicadas na vazão de alimentação

Nas figuras 7.27 e 7.28 pode-se verificar que há todas as combinações possíveis de perturbações degrau. Também, todos os degraus possuem largura bem maior que no treinamento (em torno de 50 intervalos de amostragem) o que permite observar a não linearidade de ganhos na vizinhança do estado estacionário (figura 7.29).

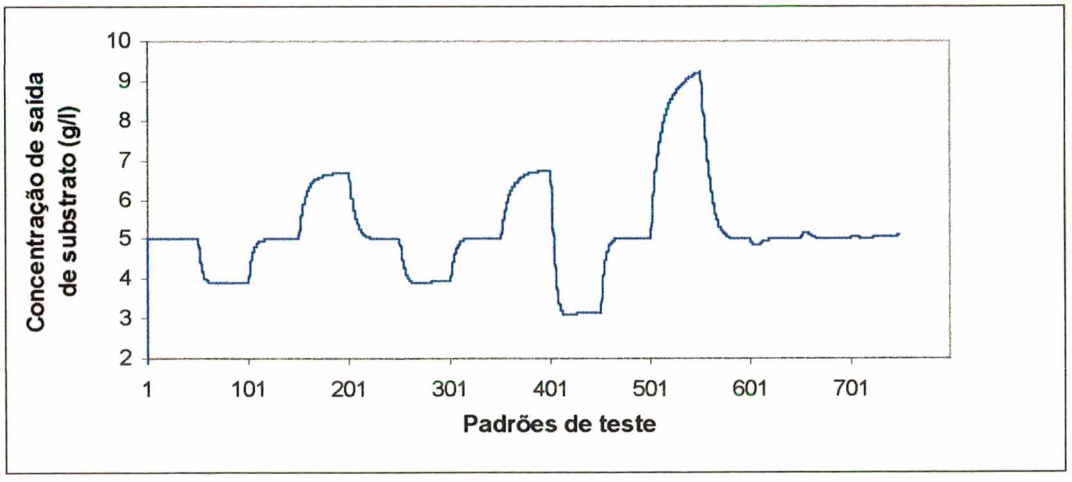

Figura 7. 29 - Concentração de substrato na saída do bioreator para as perturbações mostradas nas figuras 7.27 e 7.28

Aplicando as perturbações das figuras 7.27 e 7.28 na *RNN* obtém-se a concentração de substrato na saída do reator. A saída gerada pela rede é comparada com a saída real (figura 7.30). O erro médio quadrático do teste foi de 0.00055.

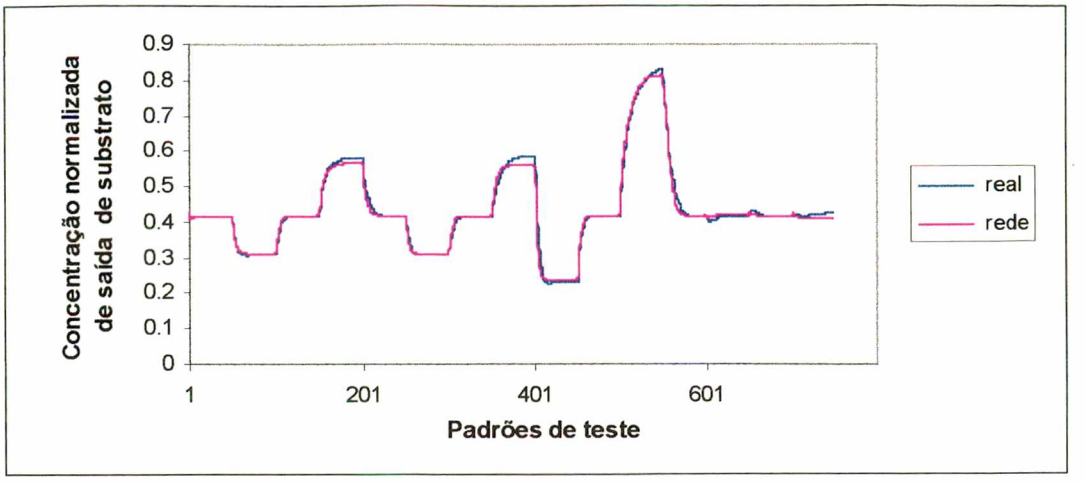

Figura 7. 30 - Comparação entre a saída da rede e a real para a concentração de substrato na saída do bioreator

Pode ser observado na figura 7.30 que a *RNN* consegue reproduzir bem o estado estacionário e as não linearidades em torno desse.

7.7 Conclusões

Neste capítulo foram identificados dois processos: o primeiro foi um *CSTR* com múltiplos estados estacionários e o segundo um bioreator com não linearidade de ganhos.

O *CSTR* com camisa de resfriamento possui dois estados estacionários estáveis e um instável. As perturbações utilizadas para identificação e simulação obrigam as variáveis do processo a percorrer trajetórias nas vizinhanças dos dois estados estacionários. Após fixada uma estrutura adequada, foi efetuada um simulação com o modelo e pode-se comprovar a capacidade da *RNN* em identificar a multiplicidade de estados estacionários.

O bioreator apresenta ganhos diferentes na variável de saída do processo para perturbações de igual amplitude e sinal oposto. Neste caso também foi comprovada a capacidade da *RNN* de simular um processo não linear.

Tanto para a identificação do *CSTR* quanto do bioreator, determinou-se inicialmente a estrutura da rede, ou seja, o número de atrasos nas perturbações externas e o número de neurônios escondidos através de um critério baseado no erro médio quadrático de treinamento. Nos dois casos observou-se que o erro quadrático não se modifica ou até mesmo aumenta a partir de um certo número atrasos e neurônios escondidos. O mesmo comportamento foi observado para o número de gerações necessárias a convergência do algoritmo genético, indicando que o aumento do tamanho da *RNN* não necessariamente leva a um melhor desempenho.

Cabe ressaltar que o algoritmo genético utilizado no treinamento da *RNN*, para identificação do *CSTR* e do bioreator, foi o mesmo empregado no treinamento da rede estática no capítulo 6. Estes problemas tem um equacionamento completamente diferente quando resolvidos de forma analítica.

8 CONTROLE PREDITIVO COM REDES NEURAIS RECORRENTES

8.1 Introdução

Uma das técnicas de controle para sistemas não lineares mais utilizadas atualmente é o controle preditivo. Nesta seção serão destacadas apenas as idéias básicas de controle preditivo; uma descrição mais detalhada pode ser encontrada em GARCIA *et alli* (1989) [63] e CAMACHO & BORDONS (1994) [64].

As idéias comuns aos diferentes esquemas de controle preditivo são:

• As saídas futuras de um processo entre os horizontes N_1 e N_2, chamados de *horizontes mínimo* e *máximo de predição*, são preditas a cada instante de tempo k utilizando o modelo do processo. Estas saídas preditas $\bar{y}(k+i\,|\,k)$ para $i = N_1...N_2$ dependem dos valores conhecidos no instante k (entradas e saídas passadas) e dos futuros sinais de controle $U_j(k+i\,|\,k)$, com $j = 1...N$ (número de variáveis manipuladas) e $i = 0...N_u$ (*horizonte de controle*), que devem ser determinados;

• O conjunto de futuras ações de controle é calculado otimizando algum critério de forma a manter o processo o mais próximo possível de uma trajetória de referência $y_{ref}(k+i)$, que pode ser o próprio *set point* ou, em geral, uma trajetória suave de mudança de *set point*;

• Normalmente aplica-se ao processo apenas o primeiro sinal de controle da seqüência calculada a cada intervalo de tempo. Esta estratégia é adotada pois a qualidade da predição do modelo deteriora a medida que aumenta o *horizonte de predição*.

O critério de otimização mais empregado é o erro quadrático entre a saída predita e a estipulada pela trajetória de referência. Em muitos casos, pode-se também incluir o esforço de controle na função objetivo, CANCELIER (1997) [45]. O critério de otimização é mostrado na equação 8.1.

$$J(N_1,N_2,N_u)=\sum_{i=N_1}^{N_2}\delta(i)\left[\bar{y}(k+i\mid k)-y_{ref}(k+i)\right]^2+\sum_{j=1}^{N}\sum_{i=1}^{N_u}\lambda(j,i)\left[\Delta U_j(k+i-1)\right]^2 \tag{8.1}$$

Com uma trajetória de referência definida pela equação:

$$y_{ref}(k+i)=\alpha y_{ref}(k+i-1)+(1-\alpha)y_{sp} \quad \text{para i=1...N}_2 \tag{8.2}$$

$$\text{com} \quad y_{ref}(k)=y(k).$$

Onde:

$\lambda(j,i)$ - Fatores de penalização para $j=1..N$ e $i=1..N_u$;

α - Fator de ajuste da trajetória de referência;

y - Variável controlada;

y_{sp} - *Set point*, ponto ao qual o sistema deve ser conduzido.

O objetivo ao minimizar a equação 8.1 é fazer que a saída futura $\bar{y}(k+i\mid k)$ siga a referência $y_{ref}(k+i)$ e, ao mesmo tempo, o esforço de controle $\Delta U_j(k+i-1)$ seja minimizado. Observando a equação 8.1, percebe-se que essa possui alguns graus de liberdade (N_1, N_2, N_u, $\delta(i)$, $\lambda(j,i)$) que podem ser modificados para obter o comportamento desejado do sistema controlado. O significado de N_1 e N_2 é bastante intuitivo. Eles indicam o intervalo de tempo que é desejável que a saída siga a referência. Se N_1 é definido com um valor alto significa que os erros nos primeiros instantes não são importantes. Os coeficientes $\delta(i)$ e $\lambda(j,i)$ são seqüências de parâmetros que ponderam o comportamento futuro, normalmente estas sequências assumem valores constantes ou o formato de funções exponenciais. Por exemplo, é possível obter uma ponderação exponencial para $\delta(i)$ em função do horizonte de predição utilizando-se a expressão 8.3:

$$\delta(i) = \beta^{N_2 - i} \quad (8.3)$$

Se um valor entre 0 e 1 é atribuído a β, os erros serão ponderados em forma crescente para valores futuros, levando o sistema a percorrer trajetórias mais suaves com menor esforço da controle. Se β é escolhido maior que 1, então os erros nos primeiros intervalos de tempo são mais penalizados e o controlador pode adotar ações mais bruscas.

Na minimização da função custo, a maioria dos métodos de projeto para controladores preditivos utiliza uma trajetória de referência $y_{ref}(k+i)$ que não necessariamente coincide com o *set point*. Geralmente utiliza-se uma trajetória suave gerada a partir do valor atual da saída e do *set point* (equação 8.2). O parâmetro α desta expressão deve ser selecionado no intervalo [0,1]. Analisando a equação 8.2 percebe-se que quanto mais próximo a 1 for o valor de α mais suave será a trajetória de referência.

Neste trabalho foi utilizada uma abordagem clássica de controle preditivo para comprovar a viabilidade do uso de *RNNs* treinadas por algoritmos genéticos. O treinamento da rede neural foi realizado utilizando o método de identificação em *paralelo*, e o controlador implementado em tempo real. O algoritmo genético foi utilizado no controlador preditivo tanto para treinamento da *RNN* quanto para otimização da equação 8.1.

8.2 Utilização de Redes Neurais como Modelos para Controladores Preditivos

O modelo do processo é o principal componente do controlador preditivo e deve ser capaz de capturar a dinâmica do processo, de modo a predizer o comportamento futuro desse.

As redes neurais, conforme comentado, podem ser utilizadas como modelos de processos não lineares. Cabe ressaltar que existem algumas diferenças entre a utilização de redes estáticas e

dinâmicas para projeto de controladores preditivos. Esta diferença será descrita para um processo genérico com uma entrada, uma saída mais tempo morto. No entanto, a mesma abordagem poderá ser utilizada para tratar de processos com mais de uma entrada ou saída.

8.2.1 Modelos de Predição Baseados em Redes Estáticas

Um processo com tempo morto possuindo uma entrada e uma saída pode ser representado por uma equação do tipo:

$$\bar{y}(k+1) = F(y(k), u(k-d)) \quad (8.4)$$

Onde:

k = intervalo de amostragem atual;

$\bar{y}(k+1)$ = saída predita no intervalo de tempo $k+1$;

$y(k)$ = valor atual da saída do processo;

d = tempo morto/ número de intervalos de amostragem;

$u(k-d)$ = ação de controle aplicada a d intervalos de amostragem passados.

Fazendo a substituição $k = k+d$ na expressão (8.4) obtém-se:

$$\bar{y}(k+d+1) = F(y(k+d), u(k)) \quad (8.5)$$

Deve-se notar que a predição efetuada pela equação 8.5 necessita do valor de $y(k+d)$ que é obviamente desconhecido no instante k. Por esta razão deve-se fazer um estimativa para $y(k+d)$, que pode ser obtida substituindo-se $k = k-1$ na equação 8.5. A equação resultante é:

$$\bar{y}(k+d) = F(y(k+d-1), u(k-1)) \quad (8.6)$$

Substituindo (8.6) em (8.5), obtém-se a equação 8.6:

$$\bar{y}(k+d+1) = F(F(y(k+d-1), u(k-1)), u(k)) \quad (8.6)$$

Repetindo o processo *d* vezes chega-se a:

$$\bar{y}(k+d+1) = F(F(F.....(y(k), u(k-d).....), u(k-1)), u(k)) \quad (8.7)$$

A equação 8.7 pode ser colocada na forma:

$$\bar{y}(k+d+1) = G(y(k), U(k)) \quad (8.8)$$

Onde: $U(k)$ = Vetor de ações de controle desde o instante k-d até k;

G = função obtida da composição da F consigo própria d vezes.

Deve-se notar que na expressão (8.8) o vetor $U(k-1)$ (ações de controle até o instante (k-1)) e $y(k)$ são conhecidos. A única variável que pode ser manipulada é o valor atual da ação de controle $u(k)$.

Dessa forma, cabe ressaltar que:

• Para predizer a saída futura de um processo modelado por (8.4) com uma rede estática, deve-se treinar a rede com $y(k+d+1)$ como saída e como entradas o vetor $U(k)$ e o valor real da saída do processo $y(k)$. Caso se deseje conhecer toda a curva de saída até o instante $k+d+1$, ao invés apenas de $y(k+d+1)$, deve-se treinar a rede com as respectivas $d+1$ saídas, ou seja, de $y(k+1)$ até $y(k+d+1)$ (vetor $Y(k+d+1)$);

• A predição de $d+1$ passos à frente leva a um aumento da complexidade do problema. A relação inicial $\bar{y}(k+1) = F(y(k), u(k-d))$ foi transformada em outra mais complexa $\bar{y}(k+d+1) = F(F(F.....(y(k), u(k-d).....), u(k-1)), u(k))$ através da composição da função F consigo própria d vezes. Dessa forma, qualquer não linearidade presente no processo é acentuada pelo aumento do *horizonte de predição*.

8.2.2 Modelos de Predição Baseados em Redes dinâmicas

Para fazer predições de um processo com uma rede estática efetuou-se a composição da função F, equação 8.4, consigo própria aumentando a complexidade final do problema. Porém, como a capacidade da predição de uma rede neural recorrente é maior pode-se empregar outra metodologia.

Partindo da equação 8.4, repetida novamente aqui:

$$\bar{y}(k+1) = F(y(k), u(k-d)) \quad (8.9)$$

Faz-se uma predição para o instante $k+2$ (equação 8.10), faz-se $k = k+1$ na equação 8.9, ou seja:

$$\bar{y}(k+2) = F(\bar{y}(k+1), u(k-d+1)) \quad (8.10)$$

Deve-se notar que no instante k o valor real de $y(k+1)$ é desconhecido e, portanto, foi substituído por $\bar{y}(k+1)$, valor predito pela *RNN* no instante $k+1$. Repetindo o processo até o instante d obtém-se:

$$\bar{y}(k+3) = F(\bar{y}(k+2), u(k-d+2))$$
$$\vdots \qquad (8.11)$$
$$\bar{y}(k+d+1) = F(\bar{y}(k+d), u(k))$$

Dessa forma, para predizer a saída futura de um processo modelado por (8.4) com uma rede dinâmica, deve-se treinar a rede com *y(k+1)* como saída e *u(k-d)* como entrada. O vetor de saída *Y(k+d+1)* é conseguido realimentando a saída predita pela rede nela própria *d* vezes.

Comparando as abordagens estática e dinâmica percebe-se que a rede recorrente é equivalente a uma rede estática de menor tamanho. Sendo que está diferença deve-se acentuar com aumento do horizonte de predição:

• Para uma rede estática o número de entradas é *y(k)+ U(k)*, ou seja, *(d+2)* entradas (uma entrada correspondente a *y(k)* mais *d+1* entradas de *U(k)*). O número de saídas, caso se deseje toda a trajetória de saída desde o instante *k+1* até *k+d+1*, é *Y(k+d+1)* (que é composto de *d+1* elementos). No caso da rede dinâmica, há apenas uma entrada *u(k-d)* e uma saída *y(k+1)*. Para conseguir o vetor de saídas *Y(k+d+1)*, basta realimentar o valor predito pela rede *d* vezes;

• Conforme mostrado para a rede estática, a composição da função do sistema *F* nela própria, aumenta a dificuldade de identificação. Portanto, serão necessários mais neurônios na(s) camada(s) escondida(s) de uma rede estática quando comparada a uma *RNN*. Uma ilustração das duas abordagens pode ser vista nas figura 8.1 e 8.2.

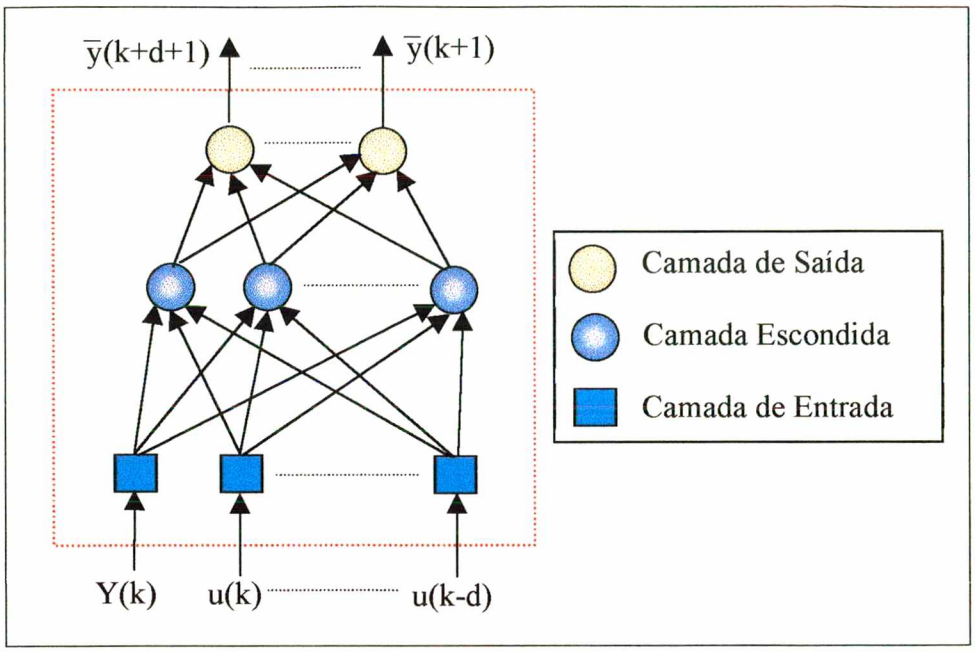

Figura 8. 1 - Ilustração de uma rede estática utilizada como modelo de predição

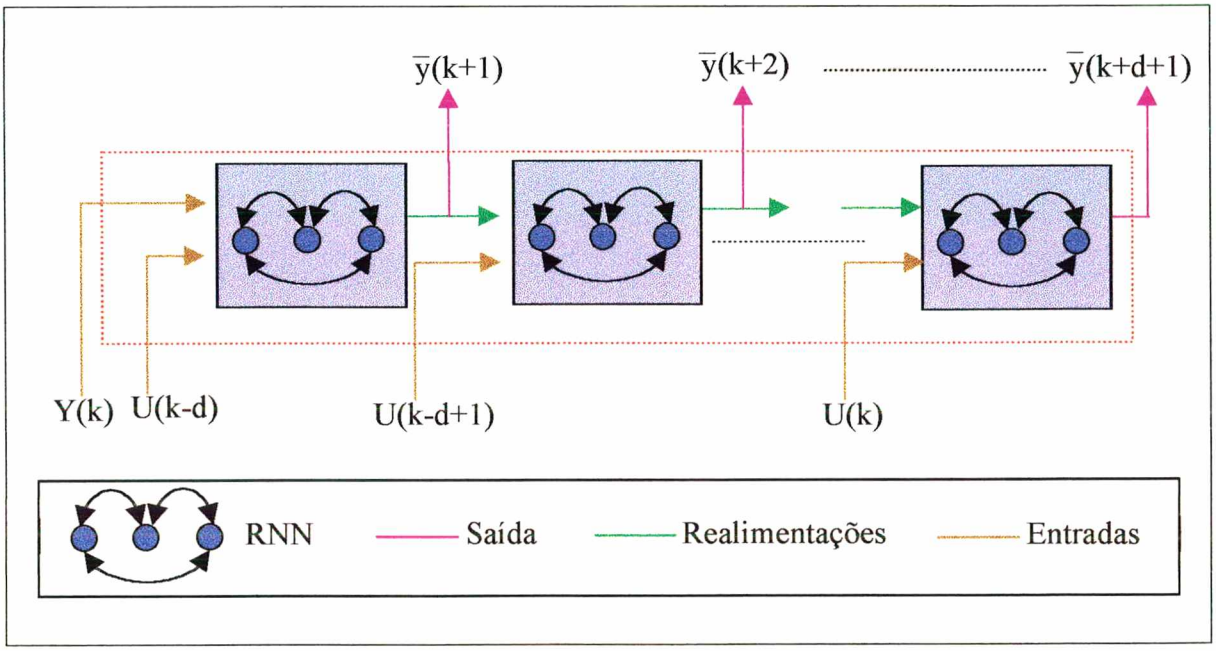

Figura 8. 2 - Ilustração de uma rede dinâmica utilizada como modelo de predição

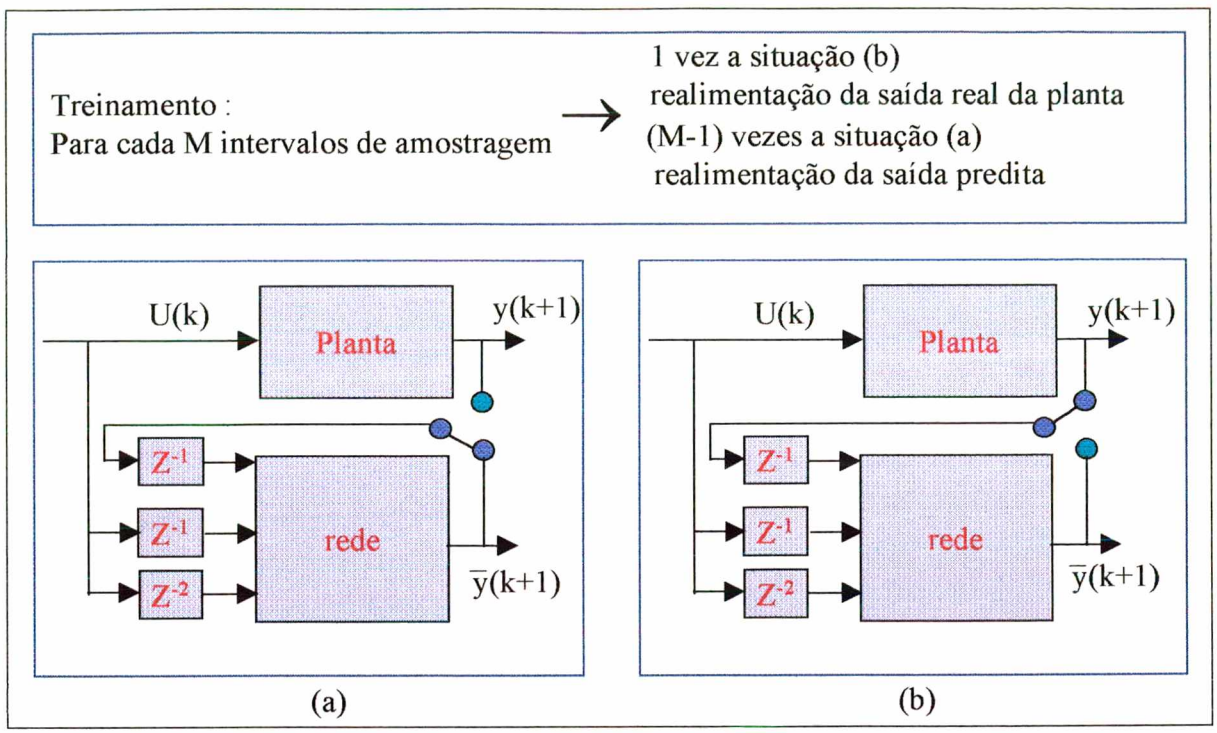

Figura 8. 3 - Treinamento modificado para *RNN*

(a) Realimentação da saída predita pela *RNN*; (b) Realimentação da saída da planta

8.2.4 Formação dos padrões

A formação dos padrões de forma a eliminar o tempo morto, na modelagem e controle de processos, foi um dos principais avanços mostrados por MAZZUCO (1996) [44]. A determinação do valor do tempo morto é essencial na formação dos padrões.

A tabela 8.1 ilustra o deslocamento temporal dos componentes de um padrão que, sem perda de generalidade, será admitido possuir um tempo morto (T_m) igual a dois intervalos de amostragem. Também, admite-se que o processo possui apenas uma entrada e uma saída.

Tabela 8.1- Formação dos Padrões

Instante (k)	Y_k	U_k
1	Y_1	U_1
2	Y_2	U_2
3	Y_3	U_3
4	Y_4	U_4
5	Y_5	U_5
6	Y_6	U_6
7	Y_7	U_7

Na tabela 8.1, pode-se observar que no instante que a ação de controle é aplicada, por exemplo em k=1 (U_1), o efeito na variável controlada só acontecerá $(T_m+1)=3$ intervalos de amostragem adiante, ou seja, em Y_4 como visto na tabela. Dessa forma as perturbações aplicadas devem estar defasadas em no mínimo $Tm+1$ intervalos de amostragem em relação à saída.

8.2.5 Incorporação do Erro de Predição no Modelo Neural

Partindo de um sistema que pode ser modelado pela equação 8.4 $y(k+1)=F(y(k),u(k-d))$, treina-se uma rede neural para aproximar a expressão F, ou seja:

$$F(y(k),u(k-d)) \approx H(y(k),u(k-d)) \quad (8.12)$$

Onde: $H(y(k),u(k-d))$ é saída predita pela rede neural.

E a equação de predição fica:

$$\bar{y}(k+1) = H(y(k), u(k-d)) \quad (8.13)$$

Em um grupo de dados obtido a partir de perturbações aplicadas ao processo pode não haver informação suficiente para uma identificação adequada, sendo que, uma rede neural treinada com tal grupo de dados apresentará erros de predição em determinadas regiões do processo. Portanto, a equação 8.13 deve ser modificada para levar em conta os erros de predição, ou seja:

$$y(k+1) = H(y(k), u(k-d)) + E(k+1) \quad (8.14)$$

Onde E é o erro de predição, que é a diferença entre o valor real do processo no instante $k+1$ e a saída estimada pela rede. Este termo é particularmente importante na correção de erros de predição de estado estacionário, que de outra forma causariam *off-set* dentre outros problemas. O erro de predição em $k+1$ não é conhecido no instante k. Porém, normalmente considera-se esse igual ao erro de predição cometido no instante k, GARCIA *et alli* (1989) [63] e SEBORG *et alli* (1989) [65], ou seja:

$$y(k+1) = H(y(k), u(k-d)) + E(k) \quad (8.15)$$

A equação 8.15 será utilizada posteriormente no controle de um tanque cônico-cilíndrico.

8.3 Otimização da Função Custo do Controlador Preditivo Utilizando Algoritmos Genéticos

Para um controlador baseado em uma rede neural recorrente, a otimização da função objetivo (equação 8.1) por métodos do tipo gradiente é uma tarefa difícil. Para evitar o cálculo das derivadas pode-se recorrer aos algoritmos genéticos. Neste caso deve-se codificar o vetor de ações de controle presentes e futuras na forma de um cromossomo.

8.3.1 Codificação do Vetor de Ações de Controle

A figura 8.4 mostra um cromossomo formado pelo vetor de ações de controle presentes e futuras. O número de incógnitas da função custo a ser determinado é igual ao horizonte de controle N_u vezes o número de variáveis manipuladas N.

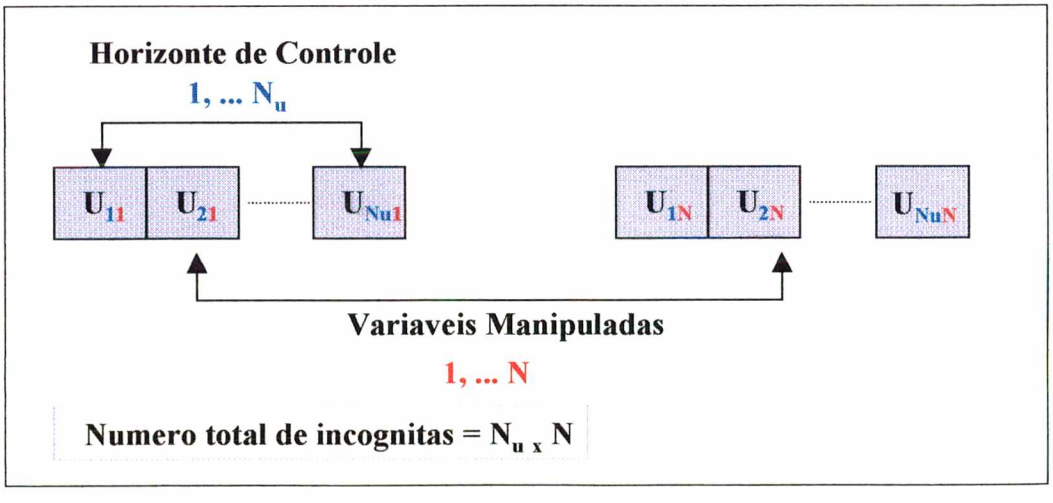

Figura 8. 4 -Vetor de ações de controle escrito na forma de um cromossomo

8.3.2 Restrições das Ações de Controle

Em situações reais todos os processos estão sujeitos a restrições. Os atuadores tem um campo limitado de ação, como no caso de válvulas que são limitadas às posições totalmente aberta ou fechada, bem como uma determinada taxa máxima de variação. Características construtivas, problemas de segurança, ambientais ou até mesmo faixa de trabalho de sensores podem causar limitações nos valores assumidos pelas variáveis do processo, como por exemplo, níveis em tanques, fluxos em canalizações ou limites de pressão e temperatura de reatores. Além disso, as condições operacionais são normalmente definidas por restrições baseadas em motivos econômicos. Todos estes fatores introduzem restrições na função custo do controlador.

No controle do tanque cônico-cilíndrico, a ser descrito posteriormente, a restrição na função objetivo refere-se ao limite do atuador. Neste caso, o sinal de controle pode variar entre [1,5] Volts. O algoritmo genético pode facilmente tratar desta restrição, bastando apenas definir o valor mínimo de cada gene como 1 e o máximo como 5.

8.4 Materiais e Métodos

Um controlador utilizando a *RNN* como preditor e o algoritmo genético como otimizador *on-line* da função custo foi utilizado para controlar o nível de um tanque cônico-cilíndrico. Este sistema faz parte de um conjunto de sistemas do Laboratório de Controle de Processos (LCP/ UFSC).

O tanque cônico-cilíndrico apresenta a não linearidade de ganho inerente à mudança de geometria e há um forte ruído na medição do nível. Outra não linearidade é dada pela válvula de controle, que é do tipo igual percentagem e possui histerese (figura 8.5).

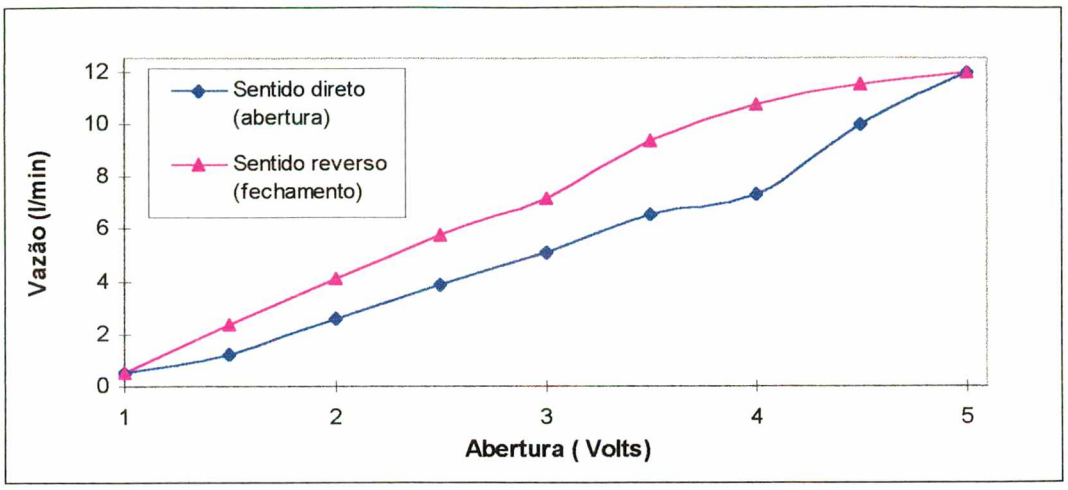

Figura 8. 5 - Dinâmica da histerese presente na válvula de controle

8.4.1 Descrição do Tanque Cônico-Cilíndrico

O tanque cônico é constituído de um tanque de PVC com duas seções de comprimento aproximadamente equivalentes, sendo alimentado pela rede hidráulica do laboratório, constituída de tubulações em PVC de 12,7 mm de diâmetro, uma bomba centrífuga de 0.25 cv de potência, uma válvula de controle com acionamento pneumático para tubulações 19,05 mm de diâmetro e válvulas manuais. O detalhamento do sistema é mostrado nas figuras 8.6 e 8.7.

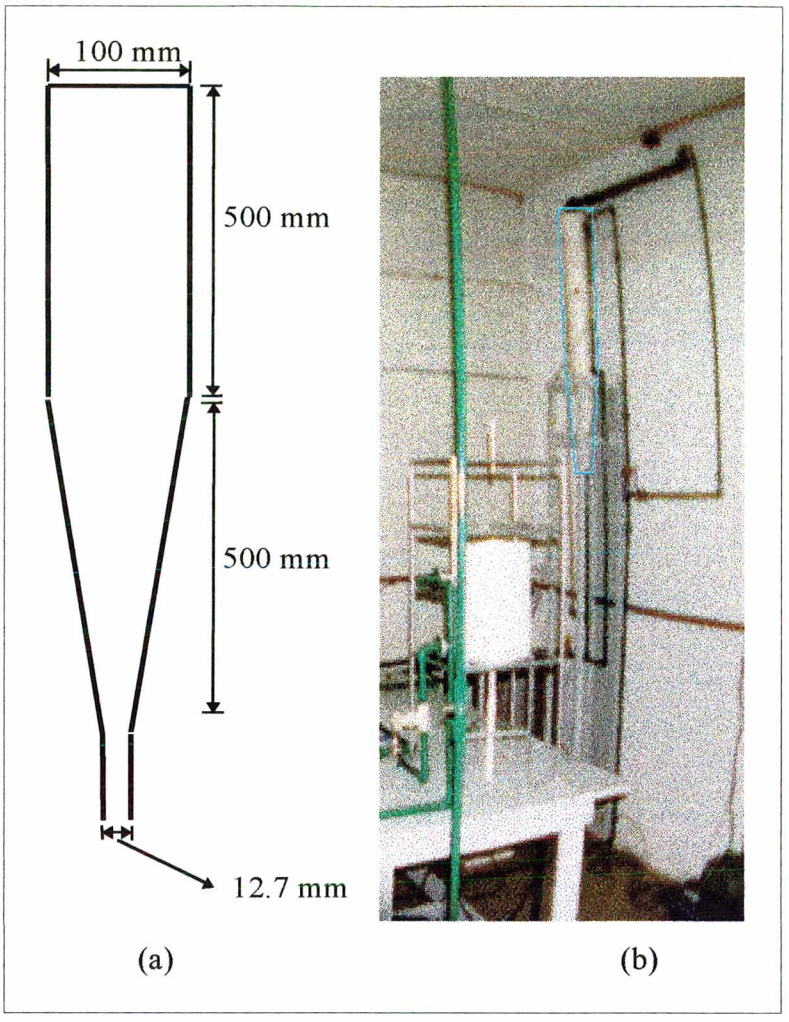

Figura 8. 6 - Tanque Cônico-Cilindrico

(a) Dimensões; (b) Vista em detalhe do equipamento no LCP

A parte inferior da seção cônica do tanque, mostrado na figura 8.6, está ligada ao sensor de pressão situado a 0,7 m do tanque, isto significa que o sistema deve ser inicializado manualmente em 0,7 m para o início dos testes. Uma descrição da calibração do sensor de pressão pode ser encontrada no apêndice 4.

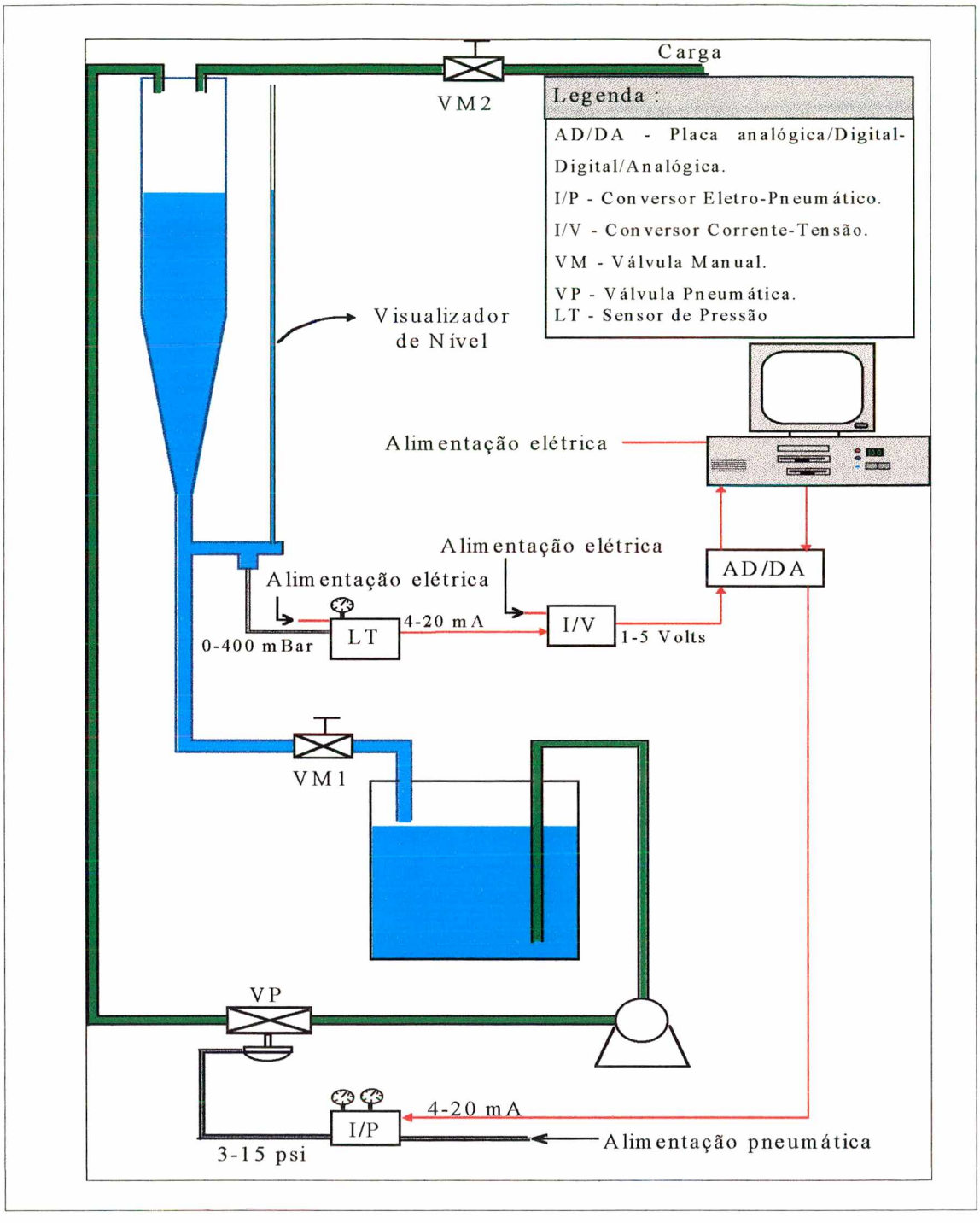

Figura 8. 7 - Sistema de Tanque Cônico-Cilíndrico

8.4.2 Implementação Computacional do Controlador

A implementação computacional de uma estratégia de controle depende de duas ferramentas, *hardware* e *software* especiais dedicados. O *hardware* utilizado é um conjunto de duas placas, uma de terminais e uma de aquisição de dados (AD/DA). A placa AD/DA utilizada é da marca DataTranslation e possui um resolução de 12 *bits*, 24 canais de entrada, 16 digitais e 8 *single ended* e 2 canais de saída analógicos, com faixa de entrada e saída de 1-5 volts. Suas funções são completamente programáveis. A placa possui ainda documentação completa para a implementação em *software*. Para efetuar o controle do tanque cônico-cilíndrico foi desenvolvido um *software* para ambiente *Windows*. Sua interface gráfica é intuitiva e de fácil utilização.

Basicamente, o *software* de controle opera pela ação de um temporizador. Assim, inicialmente, procede-se a configuração dos dados de *hardware*, dos parâmetros do controlador neural e da curva de calibração. A curva de calibração é obtida em malha aberta e representa a relação linear entre as medidas realizadas em volts e as medidas reais do sistema. No caso do tanque de nível, isto é representado em um gráfico do tipo Volt $_{x}$ Metro e, os valores dos coeficientes da equação da reta obtidos são inseridos no *software*. Uma ilustração do painel principal do *software* de controle pode ser vista na figura 8.8.

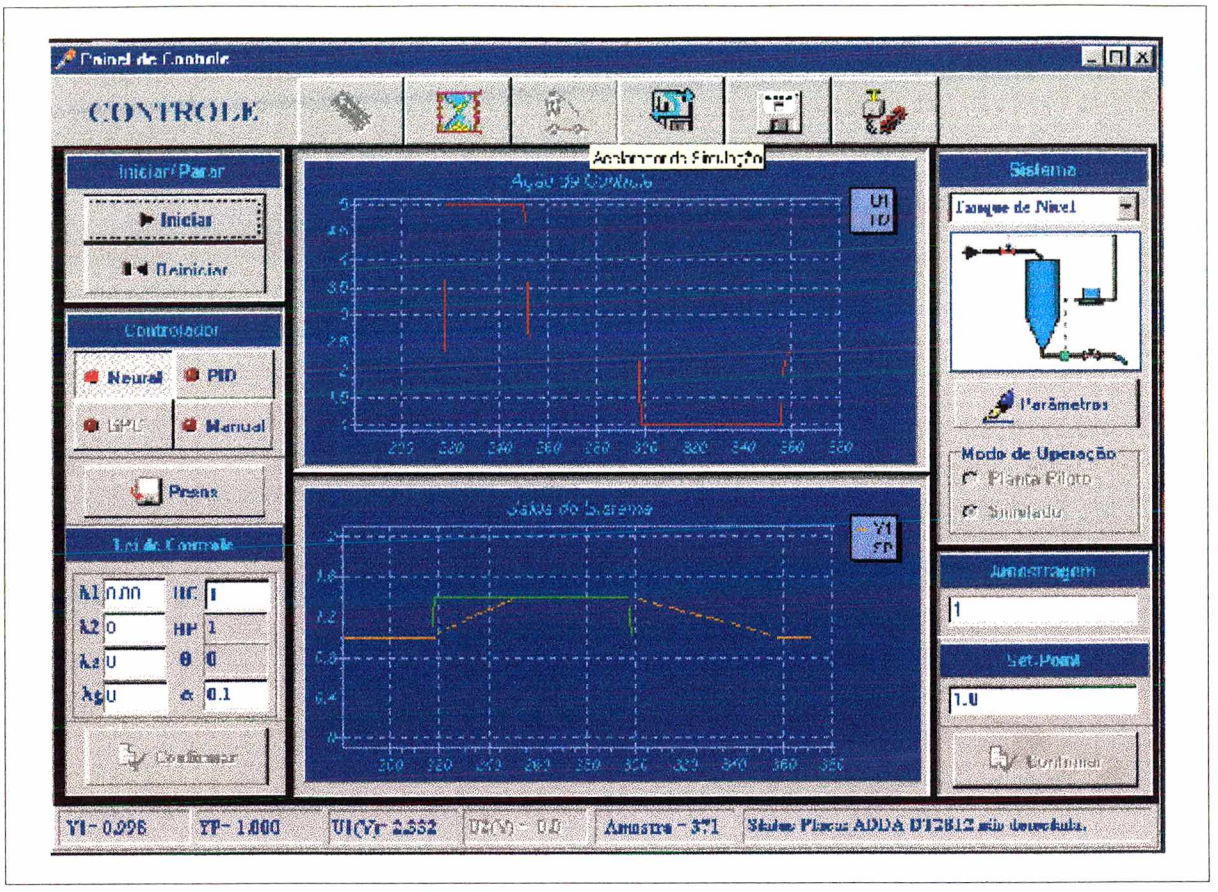

Figura 8. 8 – *Software* de controle - Painel Principal

O *software* também permite a utilização de outros controladores, tal como o *PID*, e possibilita a manipulação da ação de controle manualmente, recurso este utilizado na obtenção do grupo de dados para o treinamento de redes neurais.

Para a implementação das estratégias de controle propostas utilizou-se um microcomputador *Pentium* 75 MHz com 24 MBytes de memória RAM e 1.2 GBytes de disco rígido operando com os sistemas operacionais MS DOS 6.0 e MS Windows® 95.

As especificações de todos os equipamentos utilizados encontram-se no apêndice 5.

8.5 Resultados e Discussões

Neste capítulo serão abordadas todas as fases do projeto do controlador neural para o tanque cônico-cilíndrico que são:

- Obtenção dos dados experimentais (grupo de dados para treinamento e teste da *RNN*);
- Formação dos padrões da *RNN*;
- Seleção da melhor arquitetura;
- Validação do modelo neural obtido;
- Avaliação do desempenho do controlador.

8.5.1 Obtenção dos Dados experimentais

Dois grupos de dados foram utilizados: um para treinamento da *RNN* e outro para teste. Os grupos de dados foram coletados *off-line* e o tempo de amostragem utilizado foi de 5 s. O grupo de dados utilizado no treinamento da *RNN* foi obtido efetuando perturbações *degrau*, entre 1 a 5 Volts, no sinal enviado a válvula de controle. Estes limites correspondem aos estados de completamente fechado e aberto da válvula de controle. Nas figuras 8.9 e 8.10 mostra-se o grupo de dados formado pelas perturbações aplicadas na válvula de controle (entrada à rede) e o nível do tanque cônico-cilíndrico (saída), sendo que, as perturbações efetuadas produziram variações no nível entre 0.75 a 1.55 metros.

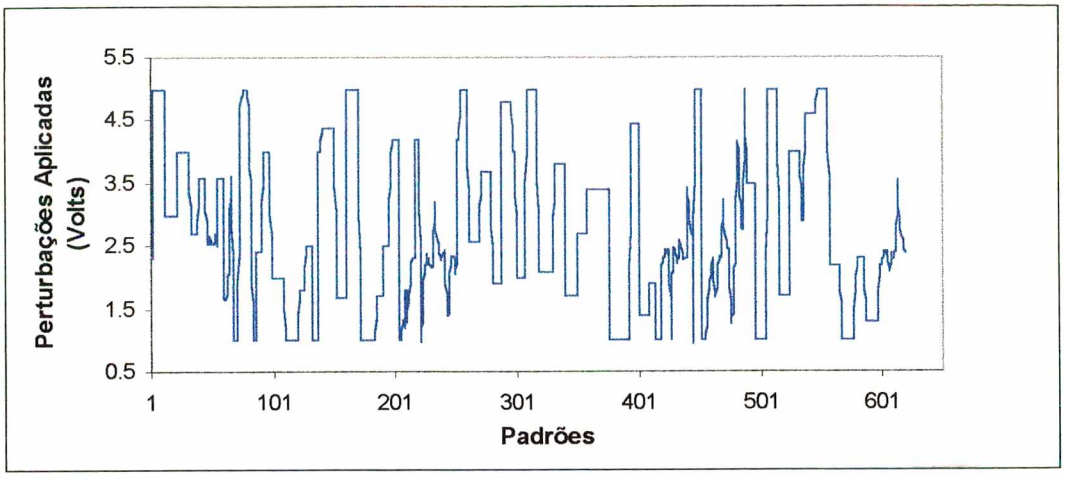

Figura 8. 9 – Perturbações aplicadas na válvula de controle (Treinamento)

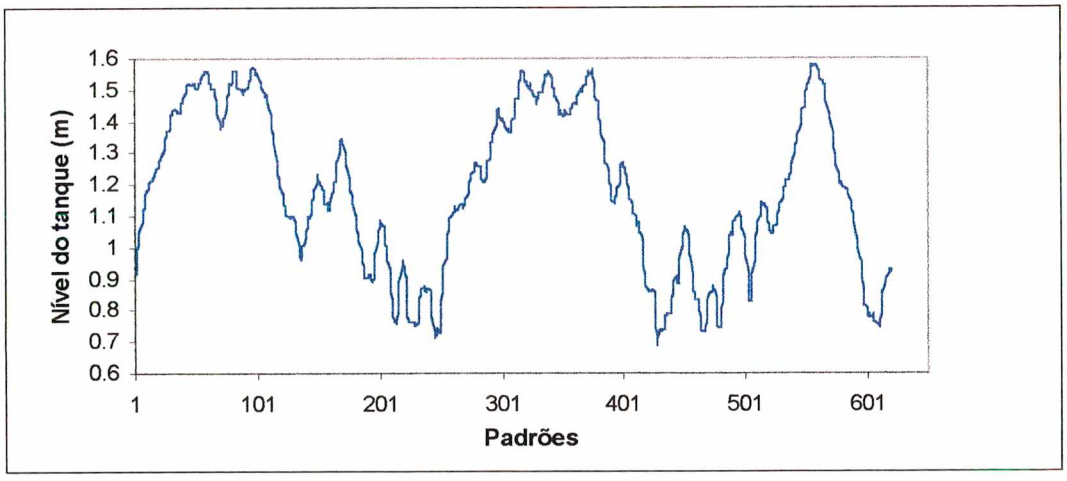

Figura 8. 10 – Variações do nível do tanque causadas pela aplicação das perturbações mostradas na figura 8.10 (Treinamento)

Para obtenção do grupo de dados de teste (figuras 8.11 e 8.12), utilizou-se a mesma estratégia empregada na obtenção dos dados de treinamento.

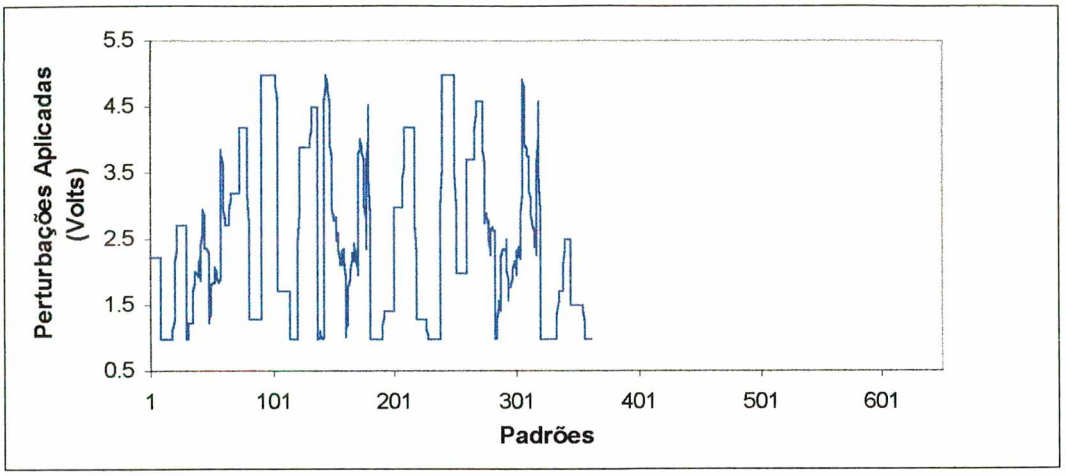

Figura 8. 11 - Perturbações aplicadas na válvula de controle (Teste)

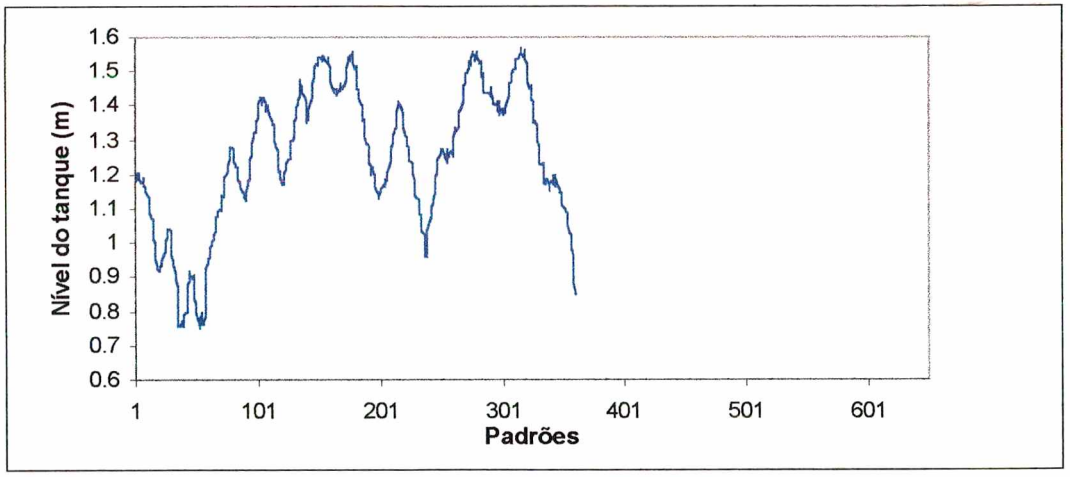

Figura 8. 12 -Variações do nível do tanque causadas pela aplicação das perturbações mostradas na figura 8.11 (Teste)

8.5.2 Formação dos padrões

Aplicando perturbações ao sistema observou-se que as respostas a estas ocorriam após 2 intervalos de amostragem, portanto, os padrões devem ser formados de modo que a ação de controle fique atrasada em 2 intervalos de amostragem em relação ao nível, ou seja:

Tabela 8.2 - Formação dos Padrões

Instante (k)	Y_k	U_k
1	Y_3	U_1
2	Y_4	U_2
3	Y_5	U_3
4	Y_6	U_4
5	Y_7	U_5

8.5.3 Seleção da melhor arquitetura

Para garantir capacidade de representação dinâmica à *RNN*, realimentou-se a rede apenas com um valor real, obtido dos dados coletados da planta, para cada cinco valores preditos. Dessa forma a *RNN* possui, em princípio, capacidade de predição de seis intervalos de amostragem. Cabe ressaltar que essa capacidade de predição é superior a dois intervalos de amostragem, conforme observada no sistema, e portanto, suficiente para utilização da *RNN* como modelo do controlador preditivo.

O procedimento utilizado para determinação da arquitetura ideal da *RNN* foi o descrito no capítulo 7, baseado no erro quadrático de treinamento. A melhor estrutura determinada consistiu de:

Número de neurônios de saída = 1. A saída é o nível do tanque;
Número de entradas = 2. As entradas da *RNN* são a abertura da válvula de controle e o valor atrasado dessa;
Número de neurônios escondidos = 3.

8.5.4 Validação do Modelo Neural

Dentre os vários conjuntos de pesos obtidos, durante a etapa de treinamento para a melhor arquitetura obtida, escolheu-se, para ser utilizado como modelo do controlador preditivo, aquele que apresentou o menor erro quadrático de predição para o grupo de dados de teste. Na figura 8.13 pode-se observar a saída real e a predição da rede neural. O erro médio quadrático de predição do teste foi de 0.00034.

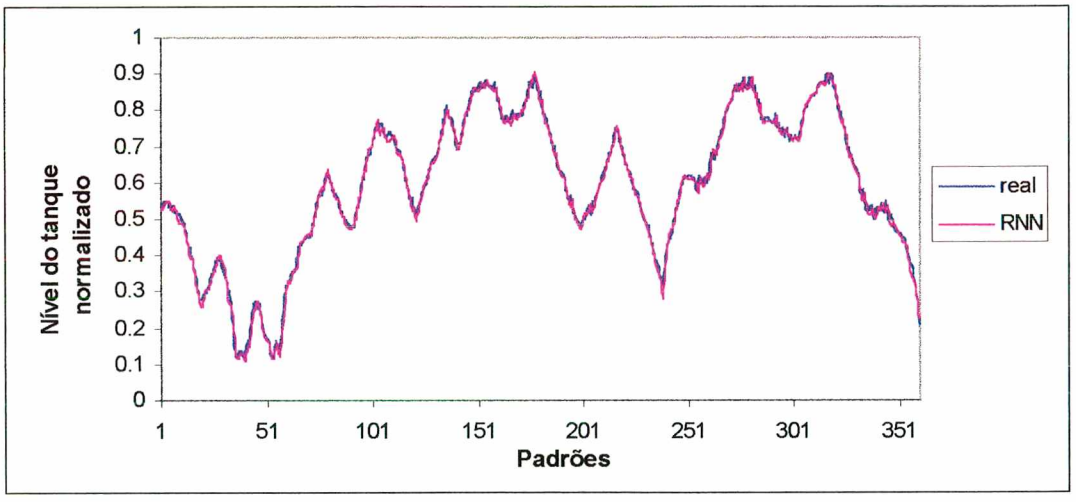

Figura 8. 13 - Comparação entre o nível real do tanque e o predito pela *RNN*

Cabe ressaltar que o teste foi efetuado, de forma semelhante ao treinamento, realimentando o valor real do processo a cada seis intervalos de amostragem, nos outros cinco intervalos de amostragem realimentou-se a saída predita pela própria *RNN*.

8.5.5 Controle do Tanque Cônico

O desempenho do controlador neural foi testado para perturbações no *set point*, sendo que, seu desempenho foi comparado, em relação as variáveis controlada e manipulada, a um controlador *PID*. Na figuras 8.14 e 8.15 são apresentadas as ações de controle do controlador baseado na *RNN* e do *PID* respectivamente, e nas figuras 8.16 e 8.17 é possível observar as transições de *set point* para os dois controladores na faixa de 0.8 e 1.45 m

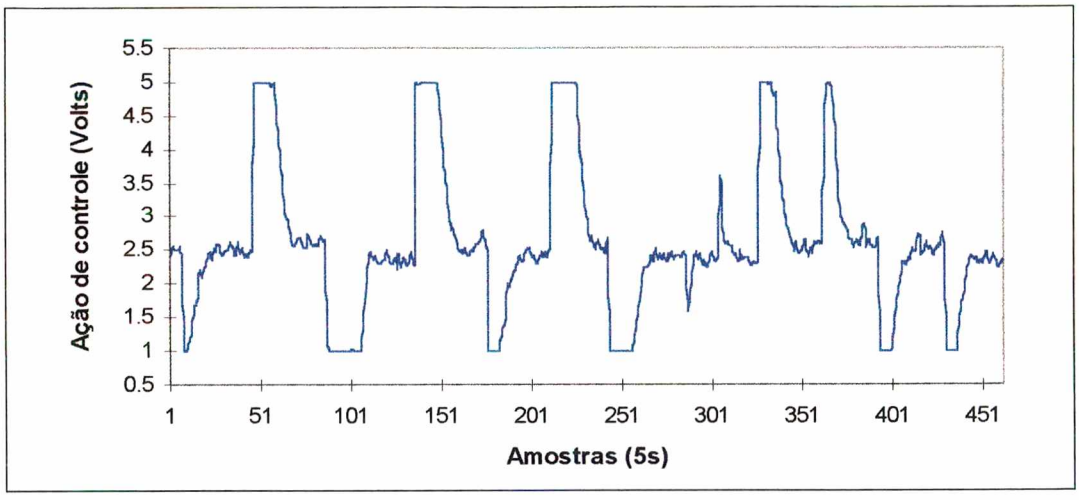

Figura 8. 14 – Ações de controle tomadas pelo controlador baseado na *RNN*

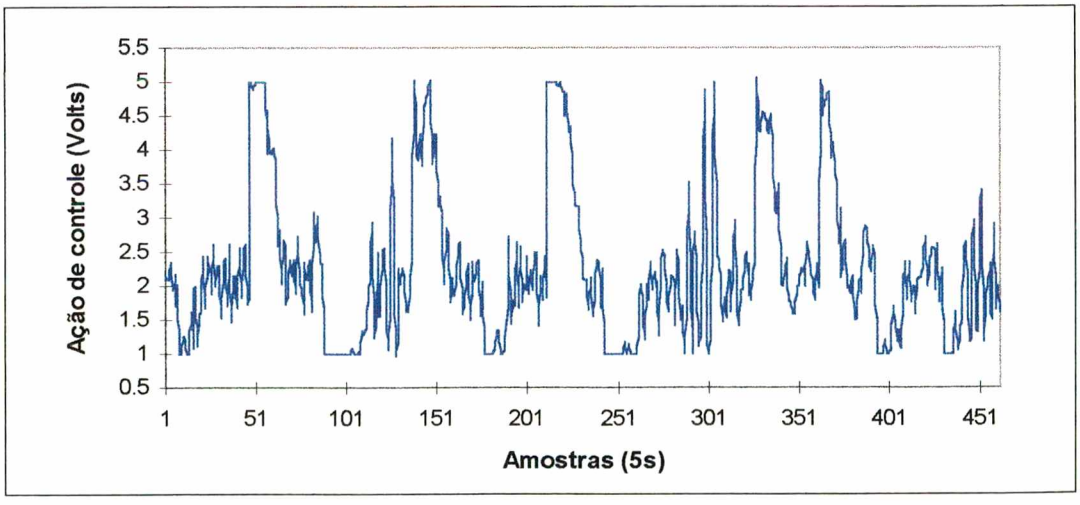

Figura 8. 15 - Ações de controle tomadas pelo controlador *PID*

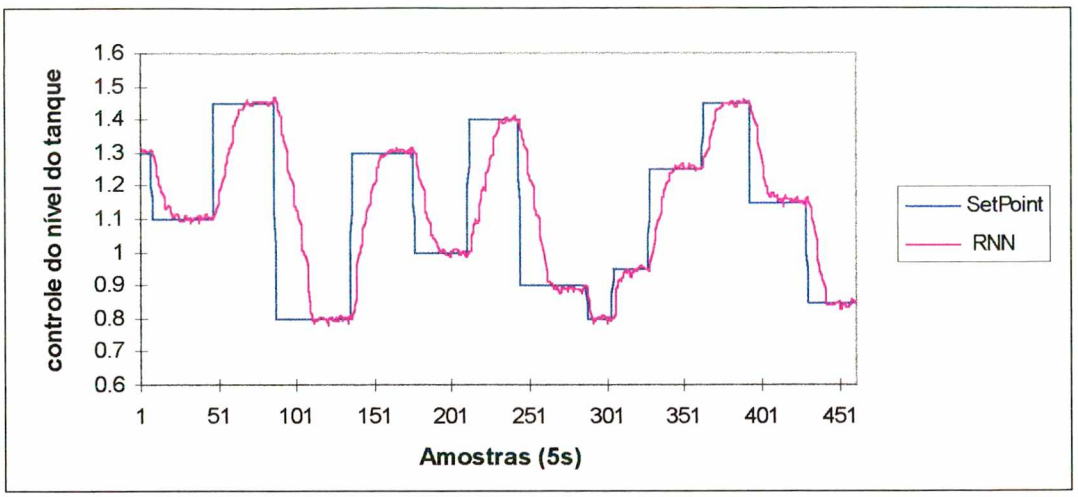

Figura 8. 16 - Transições de *set point* efetuadas pelo controlador baseado na *RNN*

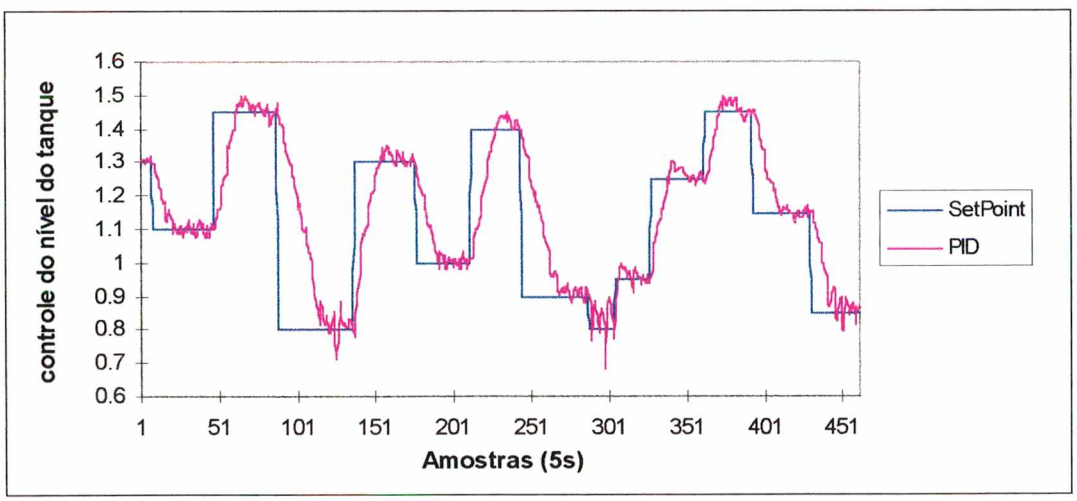

Figura 8. 17 - Transições de *set point* efetuadas pelo controlador PID

Pode-se observar nas figuras 8.14 a 8.17 que o controlador baseado na *RNN* foi mais rápido, não apresentou *overshoot* ou *off-set* e utilizou ações de controle mais suaves. O controlador *PID*, por outro lado, apresentou *overshoot* nas transições em que o valor do *set point* foi aumentado e variações significativas das ações controle. A diferença de desempenhos é mais acentuada no início da seção cônica onde a dinâmica do sistema é mais rápida, obrigando o controlador a tomar ações de controle precisas para produzir transições eficientes de *set point.*

Os parâmetros do controlador neural e do *PID* foram ajustados experimentalmente de forma a maximizar o desempenho de cada controlador. Os valores dos parâmetros são mostrados abaixo.

Parâmetros do Controlador baseado na *RNN*
$\lambda = 0.025$
$\alpha = 0.65$
horizonte de controle = 4 intervalos de amostragem
limite inferior do horizonte de predição = 1
limite inferior do horizonte de predição = 4

Parâmetros do Controlador *PID*
$K_c = 18$
$T_i = 30$
$T_d = 0.5$

Na otimização da função objetivo do controlador, observou-se que o algoritmo genético convergiu, em média, em um tempo de 500 ms. O tempo gasto é aproximadamente 10 vezes menor que a amostragem utilizada. Portanto, comprova-se a capacidade de otimização do algoritmo genético para uma aplicação em tempo real.

8.6 Conclusões

Neste capítulo foi descrita a utilização de redes dinâmicas e estáticas para identificação de um processo genérico com uma entrada, uma saída mais tempo morto. As principais diferenças observadas foram:

- A rede dinâmica equivale a uma rede estática de menor tamanho. Sendo que está diferença aumenta com o aumento do horizonte de predição;

- Quando se utiliza uma rede estática para efetuar predições a vários intervalos de amostragem futuros, essa deve aprender não apenas a função que modela o processo, mas a composição da função com ela própria para um número de intervalos igual ao *horizonte de predição*. A função resultante é mais complexa que a original, acentuando qualquer não linearidade presente na função original. No caso de uma rede dinâmica, não há aumento da complexidade do problema devido à realimentação da própria saída.

Foi desenvolvido um controlador preditivo baseado na *RNN*. No treinamento da *RNN* como modelo para o controlador utilizou-se uma estratégia em que os valores reais da saída do processo e os preditos pela rede eram realimentados convenientemente.

O controlador foi testado no controle do tanque de nível e comparado com um controlador *PID*. Para modificações no *set point* o controlador neural foi mais rápido, não apresentou *overshoot* e *off-set* e utilizou ações de controle mais suaves. O controlador *PID* apresentou *overshoot* nas transições em que o valor do *set point* foi aumentado e variações significativas das ações controle.

Cabe ressaltar que o algoritmo genético foi utilizado tanto no treinamento da *RNN* quanto na otimização da função custo do controlador preditivo. Os bons resultados obtidos incentivam a aplicação de algoritmos genéticos e *RNNs* no controle preditivo de processos mais complexos.

9 CONCLUSÕES E SUGESTÕES

O objetivo principal deste trabalho foi implementar operadores genéticos em codificação real e analisar a viabilidade do seu uso em aplicações de identificação de processos com redes neurais recorrentes. O algoritmo genético com melhor desempenho foi utilizado para implementar um controlador preditivo de um sistema não linear.

Também foi realizado um estudo de algoritmos genéticos para otimizar alguns problemas cujas soluções eram conhecidas. Os resultados obtidos foram analisados avaliando o erro de otimização e a diversidade da população em função do número de gerações necessárias para atingir a precisão desejada. A diversidade da população em codificação binária pode ser calculada facilmente porém, em codificação real não há uma forma equivalente. No entanto, como os problemas testados tinham ótimo conhecido, propôs-se uma técnica para cálculo da diversidade da população em codificação real baseada nos valores de funções objetivo correspondentes ao ótimo, a média da população e o melhor indivíduo dessa em cada geração.

Do estudo realizado foi possível obter o melhor conjunto operadores genéticos, dentre os mais freqüentemente citados na literatura e alguns operadores propostos neste trabalho. Um resultado importante foi a determinação das características presentes nos operadores que promoveram um aumento da eficiência e robustez dos algoritmos genéticos. Baseado em tais características foram propostas algumas modificações no melhor algoritmo genético avaliado e foram comparados os desempenhos dos algoritmos, com e sem modificações, no treinamento de uma rede neural estática (*feedforward*). Cabe ressaltar que o melhor resultado foi obtido com o algoritmo genético modificado que é composto dos operadores propostos neste trabalho.

O algoritmo genético modificado foi também comparado ao método *backpropagation* no treinamento da uma rede *feedforward*. Os resultados obtidos com o algoritmo genético foram ligeiramente superiores aos conseguidos com o método clássico. No entanto, deve-se ressaltar que os algoritmos genéticos são métodos de otimização de aplicação geral enquanto o método *backpropagation* é específico ao treinamento de redes *feedforward*.

Como último teste, o algoritmo genético modificado foi comparado a um *software* de uso geral (*Gaot*) na otimização de alguns *benchmarks* de otimização. O algoritmo genético modificado foi superior em todas as otimizações ao *software Gaot*, atingindo uma grande precisão na otimização dos *benchmarks*.

Como resultado de uma análise comparativa, comprovou-se que uma rede neural recorrente (*RNN*) equivale a uma rede estática de menor tamanho, sendo que, esta diferença aumenta com o incremento do *horizonte de predição*. Além disso, quando se utiliza uma rede estática para efetuar predições a vários intervalos de amostragem, essa deve aprender não apenas a função que modela o processo, mas a composição da função com ela própria para um número de intervalos igual ao *horizonte de predição*. A função resultante é mais complexa que a original, acentuando qualquer não linearidade presente. No caso de uma rede dinâmica, não há aumento da complexidade do problema devido a realimentação da saída predita pela própria rede.

Para verificar se o algoritmo genético é um método de otimização adequado ao treinamento de *RNNs* utilizadas como modelos empíricos não lineares foram realizadas duas experiências: a identificação de um *CSTR* com múltiplos estados estacionários, e a de um bioreator com não linearidade de ganhos. Tanto para o *CSTR* quanto para o bioreator determinou-se inicialmente a arquitetura da rede, ou seja, o número de atrasos nas perturbações externas e o número de neurônios escondidos, através de um critério baseado no erro médio quadrático de treinamento. Nos dois casos observou-se que o erro quadrático apresentava uma região de mínimo em função do número atrasos e neurônios escondidos. O mesmo comportamento foi observado considerando o erro quadrático em relação ao número de gerações necessárias à convergência do algoritmo genético. Após determinada a arquitetura, foi efetuada um simulação com o modelo de rede e pode-se comprovar a capacidade dessa na identificação de processos com dinâmica complexa.

O algoritmo genético utilizado no treinamento da *RNN* para identificação do bioreator e do *CSTR* foi o mesmo utilizado no treinamento da rede *feedforward* mostrando a versatilidade do método de otimização.

Finalmente, foi desenvolvido um controlador preditivo baseado em uma *RNN*. O controlador foi validado, em relação a perturbações *set point*, no controle de nível de um tanque cônico-cilíndrico. No treinamento da *RNN* como modelo de predição do controlador utilizou-se uma estratégia em que os valores reais da saída do processo e os preditos pela rede eram realimentados de forma conveniente. O desempenho do controlador foi bastante superior ao obtido com um *PID*, tanto para a variável controlada quanto para a manipulada. Cabe ressaltar que o algoritmo genético foi utilizado no controlador preditivo para obtenção do modelo neural e na otimização da função custo. Os bons resultados obtidos incentivam a aplicação de algoritmos genéticos e *RNNs* no controle de processos mais complexos.

A seguir são apresentadas algumas sugestões para continuidade deste trabalho:

Algoritmos Genéticos

No algoritmo genético, novos indivíduos são produzidos a cada geração pela combinação dos mesmos genes em diferentes indivíduos da população. Para melhorar o desempenho do algoritmo genético em problemas onde exista uma forte interação entre variáveis poder-se-ia pensar em alguma modificação do *crossover*, ou mesmo outro operador, de forma que diferentes genes possam interagir mais entre eles.

Redes Neurais

Uma *RNN* completamente interconectada, como a utilizada neste trabalho, possui um grande número de parâmetros livres (pesos) em relação ao número de neurônios processadores. O uso de arquiteturas de redes adaptadas à modelagem empírica de sistemas dinâmicos poderia reduzir o volume de cálculo necessário ao ajuste dos parâmetros.

Uma das dificuldades do treinamento de redes neurais, recorrentes ou estáticas, é a determinação do número de neurônios escondidos ou do número de neurônios na camada escondida respectivamente. Este número deve ser suficiente para identificar o processo ou sistema mas não muito grande pois haveria uma perda da capacidade de generalização da rede. Para

solucionar este problema, poder-se-ia utilizar o algoritmo genético para síntese (determinação tanto da arquitetura quanto do valor dos pesos) da rede.

Função Objetivo

A função objetivo mais utilizada no treinamento de redes neurais é o erro quadrático de aproximação, devido às restrições dos métodos de treinamento do tipo gradiente. O erro quadrático de aproximação é adequado em problemas estáticos, tal como o reconhecimento de padrões e a aproximação de funções, mas pode não sê-lo em problemas dinâmicos, com por exemplo a identificação de sistemas. Neste último caso, é muito comum obter uma diminuição do erro quadrático de treinamento enquanto o erro quadrático no grupo de dados de teste aumenta. Seria interessante estudar outras funções objetivos mais adequadas a este tipo de problema, contínuas ou não, aproveitando a flexibilidade do algoritmo genético como método de otimização.

10 APÊNDICES

10.1 APÊNDICE 1 - Funcionamento do *SGA* para uma Geração

Será apresentado um exemplo extraído de GOLDBERG (1989) [52] que mostra o funcionamento do *SGA* para *uma* geração. O problema consiste na otimização da função $f(x) = x^2$ onde x pode variar entre 0 e 31.

Para aplicar o algoritmo genético devem-se codificar as variáveis do problema em *strings* de comprimento fixo. Neste problema, há apenas uma variável 'x' que será codificada como uma *string* binária com 5 *bits* de comprimento. Definida a função objetivo e a forma de codificação será efetuada, a seguir, a simulação de uma geração do *SGA*.

Inicialmente seleciona-se a população aleatoriamente. O tamanho da população será de 4 indivíduos para esta simulação. Uma possível população é mostrada na tabela 10.1.

Tabela 10.1-População Inicial

Número da *string*	População Inicial	Valor real
1	01101	13
2	11000	24
3	01000	8
4	10011	19

Em seguida, as *strings* são decodificadas em valores reais e estes são utilizados no cálculo da função objetivo. O *fitness* será assumido igual ao valor da função objetivo. Assim, exemplificando para o terceiro elemento da tabela 4.1 tem-se:

$$x = 8 \rightarrow f(x) = x^2 \rightarrow 64 = \text{valor do } \textit{fitness}$$

Para transformar o *fitness* de cada *string* em probabilidade de seleção, basta dividi-lo pelo somatório do *fitness* de todas as *strings*:

$$P_i = \frac{F_i}{\sum_{j=1}^{N} F_j}$$

Onde: P_i = probabilidade de seleção do i-ésimo elemento;

F_i = *Fitness* do i-ésimo elemento;

N = número de elementos da população.

Exemplificando para o primeiro elemento da tabela 4.1, tem-se

$$P_1 = \frac{F_1}{(F_1 + F_2 + F_3 + F_4)} = \frac{13^2}{(13^2 + 24^2 + 8^2 + 19^2)} = \frac{169}{(169 + 576 + 64 + 361)} = 0.144$$

O valor do *fitness*, da probabilidade de seleção além de alguns dados estatísticos para todos os elementos da população, podem ser vistos na tabela 10.2.

Tabela 10.2 -Uma geração na simulação do algoritmo genético

Número da *string*	População Inicial	Valor x	*Fitness*	Probabilidade seleção (P_i)	Cópias Esperadas ($N*P_i$)	Cópias Selecionadas
1	01101	13	169	0.144	0.576	1
2	11000	24	576	0.492	1.968	2
3	01000	8	64	0.054	0.216	0
4	10011	19	361	0.310	1.24	1
	Soma		1170	1.00	4.00	4.0
	Média		293	0.25	1.00	1.0
	Máximo		576	0.49	1.97	2.0

Uma geração no processo do algoritmo genético começa com a seleção. Seleciona-se a nova população utilizando a *rolleta* 4 vezes. Na simulação atual, as *strings* 1 e 4 produzem uma cópia, a *string* 2 produz duas e a *string* 3 nenhuma, como mostrado na tabela 2. Comparando o número de cópias selecionadas de cada *string* com o número esperado (tamanho da população (N) x probabilidade de seleção (P_i)) obtém-se o que era previsto: os melhores pais geram mais cópias, os médios alocam o mesmo número e os mais fracos não produzem descendência.

Das *strings* selecionadas são feitas cópias que devem ser arranjadas em pares. No algoritmo genético, a composição dos pares é feita de forma aleatória entre as *strings* previamente selecionadas.

A seguir escolhe-se, também aleatoriamente, para cada par o ponto de *crossover*.

Observando a tabela 10.3, verifica-se que o primeiro cruzamento é efetuado entre as strings 2 e 1.

Tabela 10.3 - Novos indivíduos após uma geração

Strings selecionadas	Posição na População	Ponto de *Crossover*	Nova população	Valor real	*Fitness*
0110|1	2	4	01100	12	144
1100|0	1	4	11001	25	625
11|000	1	2	11011	27	729
10|011	4	2	10000	16	256
		Soma			1754
		Média			439
		Maior			729

Por sorteio obtém-se a posição 4 como ponto de *crossover* (da esquerda para a direita). O resultado deste cruzamento é;

Pais		Filhos
0110|1		01100
	→	
1100|0		11001

As duas *strings* restantes são cruzadas no ponto 2. As *strings* resultantes também podem ser vistas na tabela 10.3.

O último operador, mutação, é aplicado *bit* a *bit*. Assumindo que a probabilidade de mutação é de 0.001. Com 20 posições (população de 4 *strings* x 5 *bits/string*) transferidas poder-se-ia esperar 20*0.001 = 0.02 *bits* sofrendo mutação a cada nova geração. A simulação deste processo indica que nenhuma posição é modificada de 0 para 1 ou vice-versa.

Após completada a formação de uma nova geração a população deve ser avaliada. Para fazer isto simplesmente decodificam-se as novas *strings* geradas pelo *SGA* e calcula-se valor do *fitness*. Os resultados de uma única simulação são mostrados no lado direito da tabela 10.3.

Comparando-se as tabelas 10.2 e 10.3 pode-se notar que o *fitness* do melhor indivíduo e da média melhoraram na nova geração. O *fitness* médio da população passou de 293 para 439 em uma geração. O *fitness* máximo aumentou de 576 para 729 durante o mesmo período.

Obter conclusões a partir de uma única simulação de um processo estocástico é extremamente arriscado porém, percebe-se que o algoritmo genético testa e premia as seqüências de *bits* que levam ao melhoramento da população. A melhor *string* da primeira geração [11000] recebeu duas cópias devido a seu desempenho superior à média. Quando esta se combina com outra *string* [10011] no ponto de *crossover* 2, uma das *strings* [11011] criadas é superior aos pais que a geraram.

10.2 APÊNDICE 2 – Coeficientes da função De' Jong 5

I / J	A_{IJ}	
	1	2
1	-32	-32
2	-16	-32
3	0	-32
4	16	-32
5	32	-32
6	-32	-16
7	-16	-16
8	0	-16
9	16	-16
10	32	-16
11	-32	0
12	-16	0
13	0	0
14	16	0
15	32	0
16	-32	16
17	-16	16
18	0	16
19	16	16
20	32	16
21	-32	32
22	-16	32
23	0	32
24	16	32
25	32	32

10.3 APÊNDICE 3 - Grupo de dados utilizado no problema de estimação de parâmetros

H(x,y)	x	y
4.455076	-0.47063	0.348267
3.956293	-0.9232	-0.33448
3.655326	-0.22776	-0.4225
4.834007	0.361438	0.102978
5.616624	0.415035	0.484669
5.260328	-0.44957	-0.84309
7.995677	-0.07937	0.935865
6.376821	0.362924	0.697027
5.042147	0.400753	0.219523
5.009043	-0.89881	-0.82782
3.481979	-0.20245	0.187626
6.451141	-0.06554	0.761977
7.374457	0.157908	-0.7629
6.907085	0.706234	0.347188
6.882065	0.332794	0.791802
4.732443	-0.1957	0.488943
8.023647	0.381887	0.935891
3.576443	0.00791	0.109353
7.24903	0.169275	0.870134
5.072173	-0.98495	-0.80646

10.4 APÊNDICE 4 - Calibração do Sensor de Pressão

O objetivo da calibração dos elementos sensores é obter uma relação entre a variável a ser medida, no caso do tanque de cônico-cilíndrico esta é a altura, e o sinal enviado pelo sensor. A calibração obtida, realmente, não se refere ao sensor especificamente, mas sim ao conjunto de elementos de medição, ou seja, neste caso ao sensor de pressão e o conversor I/V. Desta forma, uma relação Voltagem $_x$ Altura é obtida.

10.4.1 Procedimento para Determinação da Curva de Calibração

O *software* de controle pode ser configurado para ler as informações da planta em volts, assim inicialmente o *software* dever ser configurado e em seguida a válvula Manual, VM1 (figura 8.3), deve ser fechada e a válvula de controle aberta até que o tanque esteja completamente cheio. O tanque estando cheio, a válvula de controle deve ser fechada e a válvula VM1 aberta e fechada em diferentes posições do tanque, anotando-se a altura do tanque e a voltagem em cada instante. Assim, o grupo de dados de Voltagem e Altura é obtido. Finalmente, plota-se os valores obtidos em um gráfico Volts $_x$ Metros e procede-se uma aproximação linear. A calibração obtida para o conjunto sensor de pressão-conversor I/V está abaixo (o coeficiente de correlação da reta é 0.9995):

$$Altura(m) = -1.7966 + Volts * 1.0041$$

10.5 APÊNDICE 5 – Especificações dos equipamentos utilizados

Válvula de controle

Marca: Hiter - série 201, atuador DN0021-AC.

Características:

- Igual percentagem ar abre, falha fecha;
- Sinal de entrada 3 a 15 psi;
- Parte interna em aço inox 316, corpo em aço carbono;
- Conexões tipo rosca ¾ in.

Conversores Eletropneumáticos

Marca: Hélix tipo P11-1111-2;

Sinal de entrada: 4 a 20 mA;

Sinal de saída: 3 a 15 psi;

Alimentação: 20 psi.

Filtros Reguladores de Pressão

Marca: Hélix - tipo F11-11;

Pressão máxima de Alimentação: 250 psi;

Saída: 20 psi.

Amplificador

Marca: Microquímica - modelo 308-A;

Características:

- Ganho programável: 0 a 9.000 vezes;
- Entrada: 0 -10 mV;
- Alimentação: 220 V.

Sensor de pressão

Marca: Contrisul;

Características:

- Entrada: 0 a 400 mBar;
- Saída: 4 a 20 mA;
- Alimentação 20 Vcc;
- Corpo em aço carbono, parte interna em aço inox 316.

Bomba Centrífuga

Marca: Schneider, modelo 02.01;

Potência ¼ CV, 3400 RPM;

Alimentação 220 V.

Microcomputador

Microcomputador tipo Pentium 75 MHz.

Configuração:

- Unidade de disco rígido de 1,7 GBytes;
- Memória RAM de 16MBytes;

- Monitor SuperVGA color;
- Sistema operacional MS DOS 6.0 e Windows™ 95.

Placa Analógica-Digital/Digital- Analógica

Interface de aquisição de dados para microcomputador tipo PC/XT/AT, marca DataTranslation, modelo DT2812, barramento ISA/EISA, com capacidade de transferência via DMA.

Características:

- 16 canais de entrada analógicos em modo comum ou 8 canais em modo diferencial;
- Resolução 12 *bits*;
- Taxa de amostragem de 60 KHz;
- Faixa de entrada de 0 a 1.25 V, 2,5 V, 5 V, 10 V, +/- 1,25 V, +/- 2.5 V. +/- 5 V, +/-10 V,
- 2 Canais de saída analógica;
- Conversão na faixa 10 microsegundos a 3 min;
- 2 contadores/temporizadores de 16 *bits* dedicados a contagem de eventos e medidas de frequências;
- 1 contador/temporizador de 16 *bits* programável;
- 16 canais de saída digital;
- 16 canais de entrada digital;
- 3 canais compartilhados com contadores/temporizadores.

Conversor Tensão-Corrente

Conversor isolador de sinal marca ICI Instrumentação e Controle Industriais Ltda.;

Características:

- Configuração para termo-resistências, termopares e sinais padronizados;
- Sinal de saída: 0 a 20mA, 4 a 20 mA ou 0 a 10 Volts (optoisolado), configuráveis;
- 2 níveis de alarme configuráveis;
- Fonte auxiliar para transmissor: 5, 10, 12, 24 Vcc;
- Alimentação: 110 ou 220 Vac.

11 BIBLIOGRAFIA

[1] RUMELHART, D.; HINTON, E.G. & WILLIAMS, R.J. - **Learning Internal Representations by Error Propagation**, 1986.

[2] VAN DER SMAGT, P.P. - **Mimisation Methods for Training FeedForward Neural Networks,** Neural Networks, vol. 7, n° 1, p. 1-11, 1994.

[3] ROBITAILLE, B. ; MARCOS, B. ; VEILETTE, M. & PAYRE, G. - **Modified Quasi-Newton Methods for Training Neural Networks,** Computers Chem. Eng., vol. 20, n° 9, p. 1133-1140, 1996. l

[4] RUMELHART,D. & McCLELLAND, J. - **Parallel Distributed Processing: Explorations in the Microstructure of Cognition:Foundations**, v1,. MIT Press, Cambridge, MA.1986.

[5] WILLIAMS, R. & ZIPSER, D. - **A Learning Algorithm for Continually Running Fully Recorrent Neural Networks**, Neural Computation, n°1, p. 270- 277, 1989.

[6] PEARLMUTTER, B. A. - **Gradient Calculations for Dynamic Recurrent Neural Networks: a Survey,** IEEE Transactions on Neural Networks, vol. 6, n° 5, p. 1212-1228, 1995.

[7] PETERSON, C. & ANDERSON, J. - **A Mean Field Theory Learning Algorithm for Neural Networks**, Complex Systems, n° 1, 1987.

[8] VAN DEN BOUT, D. & MILLER, T - **Improving the Performance of the Hopfield-Tank Neural Network Through Normalization and Annealing**, Biological Cybernetics, n° 62, vol. 129, 1989.

[9] SAYED, O. - **Applying Genetic Algorithms to Recurrent Neural Networks for Learning Network Parameters and Architecture**, Master's Thesis, Department of Electrical Engineering, Case Western Reserve University, May 1995.

[10] MONTANA, D. J. - **Neural Network Weight Selection Using Genetic Algorithms-Intelligent Hybrid Systems**, Edited by S. Goonatilake and S. Khebal, © John Wiley e Sons Ltda, p. 85-103, 1995.

[11] MANDISCHER, M. ; GEYER, H. & ULBIG P. - **Neural Networks and Evolutionary Algorithms for the Prediction of Thermodynamic Properties for Chemical Engineering**, In Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning, University of New South Wales, vol. 1, Edited by X. Yao and R. I McKay, C. S. Newton, J.-H. Kim and T. Furuhashi, 1998.

[12] RISSANEN, J. – **Stochastic Complexity in Statistical Inquiry**, World Scientific, N. J. 1989.

[13] ANGELINE, P. ; GREGORY, S. & JORDAN, P. - **An Evolutionary Algorithm that constructs Recurrent Neural Networks,** IEEE Transactions on Neural Networks, vol. 5, p. 54-65, 1994.

[14] MANDISCHER, M. - **Evolving Recurrent Neural Networks with Non-binary Encoding**, IEEE Proceedings of the International Conference on Evolutionary, Computing (ICEC), Perth, Australia, 1995.

[15] SALUSTOWICZ, R. - **A Genetic Algorithm for the Topological Optmization of Neural Networks**, Master's Thesis, Technische Universitat Berlin, Berlin, Germany, 1995.

[16] KWOK, T. Y. & YEUNG, D. Y. - **Constructive Feedforward Neural Networks for Regression Problems: A Survey**, Technical Reports HKUST-CS95-43, Department of Computer Science, Hong Kong University of Science and Technology, September 1995.

[17] RUDOLPH, S. - **On a Genetic Algorithm for Selection of Optmally Generalizing Neural Network Topologies**, Proceedings of the 2nd International Conference on Adaptative Computing in Engineering Design and Control, University of Plymouth, U. K. p. 79-86, 1996.

[18] FREITAS, J. F. G. ; NIRANJAN, M. & GEE, A. H. - **The EM Algorithm and Neural Networks for Nonlinear State Space Estimation**, Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR313, May 1998.

[19] FREITAS, J. F. G. ; NIRANJAN, M. ; GEE, A. H. & DOUCET, A. - **Sequential Monte Carlo Methods for Optimisation of Neural Networks Models**, Cambridge University Engineering Department, Technical Report CUED/F-INFENG/TR 328, November 1998.

[20] PACHECO, S. S. & THOME, A. G. - **"Turbo Shake"- Um Avanço ao Método "Shake" no Treinamento de Redes Neurais**, III Congresso Brasileiro de Redes Neurais, Florianópolis, SC, p. 51-56, 1997.

[21] RUDOLPH, S. - **On A Data-Driven Model Identification Technique Using Artificial Neural Networks** – Proceedings EUROMECH 373 Colloquium on Modeling and Control of Adaptive Mechanical Structures, Magdeburg, Germany, Fortschritt-Berichte VDI, Reihe 11, n° 268, March, 1998.

[22] BURROWS, T. L. & NIRANJAN. - **The Use of Feedforward and Recurrent Neural Networks for System Identification**, Cambridge University Engineering Department, Technical Report CUED/F-INFENG/TR 158, December 1993.

[23] ROISENBERG, M ; BARRETO, J. M. & AZEVEDO, F. M. - **Feedforward and Recurrent Neural Networks Complexity Power: A Comparison Based on a Concrete Example**, III Congresso Brasileiro de Redes Neurais, Florianópolis, SC, p. 1-6, 1997.

[24] YOU, Y. & NIKOLAOU, M. - **Dynamic Process Modeling with Recurrent Neural Networks**, AIChE Journal, vol. 39, n°10, p. 1654-1666, October 1993.

[25] PARK, J. K. ; DRANOFF, J. S. & MAH, R. S. H. - **On-Line Runaway Prediction Using Recurrent Neural Networks**, Proceedings of PSE, p. 1269-1274, 1994.

[26] HUB, L. & JONES, J. D. - **Early On-line Detection of Exothermic Reactions**, Plant/ Operation Process, 5(4), 221, 1986.

[27] SU, H.B. ; FAN, L.T. & SCHLUP, J.R. - **Monitoring the process of curing of epoxy/graphite fiber composites with a recurrent neural network as a soft sensor**, Engineering Applications of Artificial Intelligence, Elsevier Sci Ltd, Exeter Engl, vol. 11, n°2, p.293-306, Apr 1998.

[28] ROVITHAKIS, G. A. ; CHRISTODOULOU, M. A. - **Adaptative Control of Unknown Plants Using Dynamical Neural Networks**, IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, n° 3, p. 400-412, March 1994.

[29] VENUGOPAL, K.P.; PANDYA, A.S. & SUDHAKAR, R. - **A Recurrent Neural Network Controller and Learning Algorithm for On-line Learning Control of Autonomous Underwater Vehicles**, Neural Networks, vol. 7, n° 5, p.833-846, 1994.

[30] HARTH, E. & PANDYA, A.S. - **Dynamic of Alopex Process: Applications to Optimization problems**, In L. M. Ricciardi, Ed. Biomathematics na related computacional problems. Amsterdam: Reidel Publications, 1987.

[31] DELGADO, A.; KAMBHAMPATI, C. & WARWICK, K. - **Dynamic recurrent neural network for system identification and control**, IEE Proceedings: Control Theory and Applications, vol. 142, n° 4, p. 307-314, Jul 1995.

[32] AHMED, M. S. & TASADDUQ, I. A. - **Neural Servocontroller for nonlinear MIMO Plant**, IEE Proc.- Control Theory Appl., vol. 145, n° 3, p. 277-290, May 1998.

[33] BRDYS, M.A.; KULAWSKI, G.J. & QUEVEDO, J. - **Recurrent networks for nonlinear adaptive control**, IEE Proceedings: Control Theory and Applications, vol. 145, n° 2, p. 177-189, Mar 1998.

[34] LINKENS, D. A & NYONGESA, H. O. - **Genetic Algorithms for Fuzzy Control**, IEE Proc. Control Theory and Applications, vol. 142, n° 3, p. 161-175, May 1995.

[35] LINKENS, D. A & NYONGESA, H. O. - **Learning Systems in Intelligent Control: an Appraisal of Fuzzy, Neural and Genetic Algorithm Control Applications**, IEE Proc. Control Theory and Applications, vol. 143, n° 4, p. 367-386, July 1996.

[36] ZUO, W. - **Multivariable Adaptative Control for a Space Station Using Genetic Algorithms**, IEE Proc. Control Theory and Applications, vol. 142, n° 2, p. 81-87, March 1995.

[37] FOGARTY, T. C. & BULL, L. - **Optimisation Individual Control Rules and Multiple Communicating Rule-Based Control Systems with Parallel Distributed Genetic Algorithms**, IEE Proc. Control Theory and Applications, vol. 142, n° 3, p. 211-222, May 1995.

[38] DOWNING, C. J. ; CORK, R. T. C. ; BYRNE B. ; COVENEY, K. & MARNANE, W.P - **Controller Optimisation and System Identification using Genetic Algorithms,** Proceedings of Irish DSP and Control Colloquim, p. 45-52, 1996.

[39] TREBI-OLLENU, A. & WHITE, B. A. - **Multiobjective Fuzzy Genetic Algorithm Optimisation Approach to Nonlinear System Design**, IEE Proc. Control Theory and Applications, vol. 144, n° 2, p. 137-142, March 1997.

[40] DELGADO, A. ; PUIGJANER, L. ; SERRA, M. ; WILKENDORF, F. & MEHLHORN, A. I. – **Neural Networks and Genetic Algorithms Modeling Chemical Process**, III Congresso Brasileiro de Redes Neurais, Florianópolis, SC, p. 161-166, 1997.

[41] CONG, S.; CHEN, B. & HE, X. - **Neural Networks Based Nonlinear System Coordinated Control**, 5 th IAFC Sysposium on Dynamics and Control of Process Systems, DYCOPS-5, Corfu, Greece, June 8-10, p. 496-501, 1998.

[42] PHIMISTER, J.R. ; FRAGA, E.S. & SEIDER, W.D. - Plantwide Controller Tuning Using a Multiobjective Genetic Algorithm, **5 th IAFC Sysposium on Dynamics and Control of Process Systems, DYCOPS-5**, Corfu, Greece, June 8-10, p. 371-376, 1998.

[43] ROQUEIRO, N. & LIMA, E. L. - **Wavelet Theory and Harmonic Analysis in Applied Scienses** – Birkhauser Boston, p. 265-297, Cambridge, MA/USA,1997.

[44] MAZZUCO, M. M. - **Implementação de um Controlador Preditivo Baseado em um Modelo Neural Associado a um Sistema Especialista**, Dissertação de Mestrado, Departamento de Engenharia Química, Universidade Federal de Santa Catarina, Florianópolis, 1996.

[45] CANCELIER, A. – **Controle Preditivo de Reatores Semi-batelada**, Dissertação de Mestrado, Departamento de Engenharia Química, Universidade Federal de Santa Catarina, Florianópolis, 1997.

[46] ZUPAN, J. & GASTEIGER, J. - **Neural Networks for Chemists**, VHC Publishers, New York, NY (USA), 1993.

[47] HENSON, M. A. & SEBORG, D. E. - **Nonlinear Process Control**, Prantice Hall, 1997.

[48] HAIKIN, S. - **Neural Networks – A Comprehensive Foundation**, Ontario: IEEE Computer, Society Press, 1994.

[49] FAUSETT, L. - **Fundamental of Neural Networks - Architetures, Algorithms and Applications**, Prantice Hall International, Inc, 1995.

[50] BÄCK, T. ; HAMMEL, U. & SCHWEFEL, H. - **Evolutionary Computation: Comments on the History and Current State**, IEE Transactions on Evolutionary Computation, vol. 1, n° 1, p.3-17, April 1997.

[51] HOLLAND, J. H. - **Adptation in Natural and Artificial Systems, An Introdutory Analysis with Applications to Biology, Control and Artificial Intelligence**,1975.

[52] GOLDBERG, D. E. - **Genetic Algorithms in Search, Optimization, and Machine Learning**, Addison-Wesley, New York, 1989.

l

[53] HERRERA, F.; LOZANO, M.; VERDEGAY, J. L. - **Algoritmos Genéticos: Fundamentos, Extensiones y Aplicaciones**, Technical Report #DECSAI-94105, Granada, ES:E.T.S. de Ingeniería Informática: Universidad de Granada, Abril 1994.

[54] POHLHEIM, H. - **GEATbx: Genetic and Evolutionary Algorithm *Toolbox* for use with Matlab**. http://www.systemtechnik.tuilmenau.de/~pohlheim/GA_Toolbox/index.html, 1997.

[55] HOUCK, C.R. ; JOINES, J.A. & KEY, M.G. - **A Genetic Algorithm for Função Optimization: A Matlab Implementation,** NCSU-IE TR 95-09, http://www.ie.ncsu.edu/mirage/, 1995.

[56] HSIA, T. C. - **System Identification – Least-Squares Methods**, Lexington Books, D. C. Heath and Company, Lexington, Massachusetts, Toronto, 1977.

[57] PINTO, J. C. - **Apostila da Disciplina de Planejamento Experimental para Estimação de Parâmetros**, Programa de Engenharia Química/COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil, 1996.

[58] ASTROM, K. J. & WITTENMARK, B - **Computer Controlled Systems –Theory and Design**, Prantice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1984.

[59] POTTMANN, M. & SEBORG, D. E. - **Identification of non-linear process using reciprocal multiquadratic functions**, J. Proc. Control, vol. 2, p. 4-189, 1992.

[60] TAHUATA, T. - **Relatório Interno** – COPPE / UFRJ,1992.

[61] EMBIRUÇU, M. - **Controle de processos não lineares,** Tese de Mestrado – COPPE/UFRJ, 1993.

[62] HENSON, M. A. & SEBORG, D. E. - **An internal model control strategy for nonlinear systems**, AIChe Journal, vol 37, n° 7, p. 1065 -1991, 1991.

[63] GARCIA, C. E. ; PRETT, D. M. & MORARI, M. - **Model Predictive Control: Theory and Practice – a Survey**, Automatica, vol. 25, p. 335-348, 1989.

[64] CAMACHO, E. F. & BORDONS, C. - **Model Predictive Control in the Process Industry**, 1994.

[65] SEBORG, D. E. ; EDGAR, T. F. & MELLICHAMP, D. A. - **Process Dynamics and Control**, New York, ed. John Wiley & Sons, 1989.