3
Técnicas da Inteligência Computacional

3.1.
Introdução

Neste capítulo são dados os conceitos fundamentais sobre as técnicas de inteligência computacional empregadas neste trabalho. Primeiramente, é feita uma breve explicação dos modelos de aprendizado empregados como aproximadores: redes neurais do tipo *feed-forward* e recorrentes com aprendizado *back-propagation*, redes RBF, e os modelos *Neuro-Fuzzy*. Em seguida, é descrito de forma sucinta o conceito de Algoritmos Genéticos, descrevendo sua arquitetura e os parâmetros utilizados no processo de evolução. Finalmente são dados alguns conceitos de algoritmos genéticos distribuídos em ambientes de computação paralela.

3.2.
Redes Neurais

Redes Neurais (Haykin, 1999) são modelos computacionais não lineares inspirados na estrutura de neurônios interconectados existente no cérebro humano, capazes de realizar as seguintes operações: aprendizado, associação, generalização e abstração. As redes neurais são compostas por diversos elementos processadores (neurônios artificiais), altamente interconectados, que efetuam operações simples, transmitindo seus resultados aos processadores vizinhos. A habilidade das redes neurais em realizar mapeamentos não-lineares entre suas entradas e saídas as tem tornado prósperas no reconhecimento de padrões (Bishop, 1995) e na modelagem de sistemas complexos.

Devido a sua estrutura, as redes neurais são bastante eficazes no aprendizado de padrões a partir de dados não-lineares, incompletos, com ruído ou mesmo compostos por exemplos contraditórios.

Na literatura pode-se encontrar muitos tipos de redes neurais com diferentes arquiteturas e algoritmos de aprendizado. Neste trabalho serão utilizadas redes com arquitetura *multilayer perceptron*, recorrentes com aprendizado do tipo *back-propagation* e redes de base radial RBF.

3.2.1.
Arquitetura *Multilayer Perceptron*

A arquitetura *Multi-Layer Perceptron* é a mais utilizada nas aplicações de engenharia.

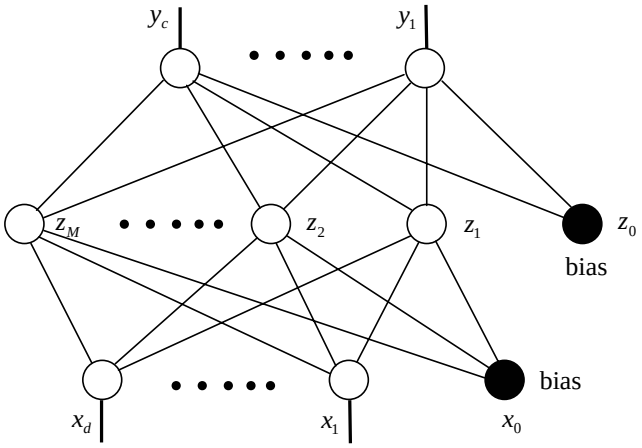

Figura 3. Representação geral da arquitetura *multilayer perceptron*.

De modo geral, esta arquitetura possui um vetor de entradas $\mathbf{x}$ de dimensão d $\mathbf{x} = [x_1, x_2, \ldots, x_d]$; um vetor de c saídas $\mathbf{y} = [y_1, y_2, \ldots, y_c]$; e M neurônios na camada escondida (pode possuir mais de uma camada escondida).

Todos os parâmetros adaptáveis (pesos e *bias*) desta arquitetura são agrupados convenientemente em um único vetor w-dimensional $\mathbf{w} = [w_1, w_2, \ldots, w_W]$ para facilitar tratamentos analíticos.

Além disso, para o cálculo (treinamento) dos parâmetros adaptáveis $\mathbf{w} = [w_1, w_2, \ldots, w_W]$, utiliza-se um conjunto de observações (dados de treinamento) $D = \left\{ \left(x^{(1)}, t^{(1)}\right), \left(x^{(2)}, t^{(2)}\right), \ldots, \left(x^{(N)}, t^{(N)}\right) \right\}$ de algum processo a ser modelado, onde $\mathbf{x} = \left(x^{(1)}, x^{(2)}, \ldots, x^{(N)}\right)$ é o conjunto de dados de entrada e $\mathbf{t} = \left(t^{(1)}, t^{(2)}, \ldots, t^{(N)}\right)$ é o conjunto de dados contendo a saída desejada.

3.2.2.
Redes Recorrentes

Como o próprio nome indica, são redes neurais que possuem arquitetura recorrente com uma ou mais conexões de retro alimentação local ou global. A mais conhecida destas redes é a rede de Hopfield (Hopfield, 1982). Redes recorrentes podem ser utilizadas como memória associativa ou para

mapeamento entrada-saída. Para esta última aplicação temos as redes *Non-linear AutoRegressive with eXogenous inputs* NARX (Ljung, 1987), redes com realimentação local (Frasconi *et al*, 1992), redes de Elman (Elman, 1990), redes Williams-Zipser e de Jordan (Williams, 1988). Cabe ressatar que este tipo de redes possui bom desempenho para treinar sistemas que variam no tempo como mostrado no trabalho de Pari (1999). No escopo deste trabalho, são utilizadas as redes de Elman

3.2.2.1. Redes de Elman

As redes de Elman utilizam retro alimentação global, como mostra a Figura 4. Neste caso existe realimentação de cada um dos neurônios da camada escondida para todos os neurônios da camada escondida, isto é, cada neurônio escondido recebe o vetor de estados "anterior" completo $a(k-1)$. Esta topologia é denominada rede completamente recorrente e conhecida como rede de Elman (Elman, 1990). A pesar do exemplo mostrado na Figura 4 conter apenas uma entrada e uma saída, essas redes são genéricas, podendo ter várias entradas e saídas.

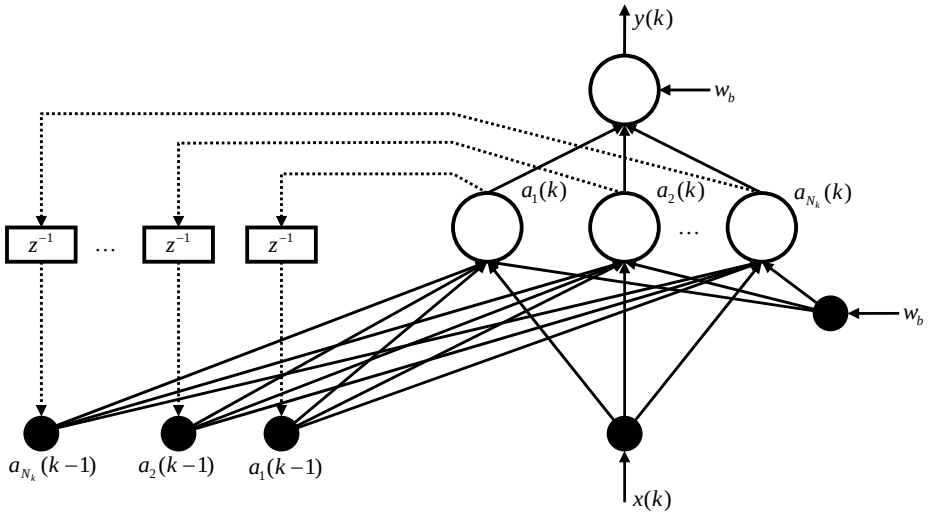

Figura 4. Rede recorrente de Elman.

3.2.3. Algoritmo de Aprendizado *Back-Propagation*

Minsky e Papert (1969) mostraram que uma rede de duas camadas *feed-forward* supera muitas limitações existentes em redes de uma camada, mas na época não tinham uma solução para o problema de ajustar os pesos da entrada e das camadas escondidas.

Rumelhart et al. (1986) apresentaram uma resposta a esta questão e respostas similares foram publicadas um pouco antes por Werbos (1974), Parker (1985) e Cun (1985).

A idéia principal desde algoritmo de aprendizado é que os erros das unidades da camada escondida sejam determinados retro-propagando os erros da camada de saída; motivo pelo qual, este método toma o nome de regra de aprendizado *back-propagation*.

Este algoritmo pode ser aplicado em redes com qualquer número de camadas escondidas. Cabe ressaltar que, para uma rede *feed fordward* foi mostrado que uma única camada escondida é suficiente para aproximar com precisão arbitrária qualquer função contínua, desde que haja uma quantidade suficiente de processadores na camada escondida e que a função de ativação destes processadores seja não linear (Hornik, 1989). Na maioria das aplicações, usa-se uma rede *feed-forward* com uma única camada escondida e com função de ativação sigmóide (Cybenco, 1989) nos neurônios da camada escondida e com função de ativação linear nos neurônios da camada de saída.

O algoritmo funciona em duas fases:

1. O p-ésimo padrão de entrada $x^{(p)}$ é inserido e propagado em sentido direto através de toda a rede para calcular os valores de saída $y_o^{(p)}$, onde $y_o^{(p)}$ é o valor de saída do processador o da camada de saída para a entrada do padrão $x^{(p)}$.

2. A saída $y_o^{(p)}$ é comparada com o valor desejado $t_o^{(p)}$, obtendo-se um valor de erro $\delta_o^{(p)}$ em cada neurônio na camada de saída. Esta fase envolve a propagação em forma reversa do valor do erro através de toda a rede e o cálculo da atualização dos pesos em todos os processadores como segue:

- O peso de uma conexão w_{kj} é ajustado em um valor proporcional ao produto do erro no processador k que recebe a entrada y_j e a saída do processador j que enviou o sinal através da sua conexão:

$$\Delta_p w_{kj} = \gamma \delta_k^{(p)} . y_j^{(p)} \tag{8}$$

onde

w_{kj}	Peso da conexão entre processadores j e k
γ	Taxa de aprendizado
$\delta_k^{(p)}$	Erro no processador k para o padrão $x^{(p)}$

$y_j^{(p)}$ Saída do processador j para o padrão $x^{(p)}$

Se o processador k pertence à camada de saída o, o erro é :

$$\delta_o^{(p)} = \left(t_o^{(p)} - y_o^{(p)}\right)F'\left(s_o^{(p)}\right) \tag{9}$$

onde

$t_o^{(p)}$ Valor desejado do processador de saída o para o padrão de entrada $x^{(p)}$

$F'\left(s_o^{(p)}\right)$ 1ª derivada da função de ativação da soma ponderada $S_o^{(p)}$ de todas as entradas do neurônio de saída o para o padrão $x^{(p)}$

- Dado que a função de ativação $F(\bullet)$ é sigmoidal, tem-se que:

$$y^{(p)} = F(s^{(p)}) = \frac{1}{1+e^{-s^{(p)}}} \tag{10}$$

- Calculando a 1ª derivada de $F(s^{(p)})$:

$$F'(s^{(p)}) = y^{(p)}(1-y^{(p)}) \tag{11}$$

- Assim, o erro para um processador o da camada de saída pode ser escrito como:

$$\delta_o^{(p)} = \left(t_o^{(p)} - y_o^{(p)}\right)y_o^{(p)}\left(1-y_o^{(p)}\right) \tag{12}$$

Se o processador k pertence à camada escondida h, o erro é calculado recursivamente a partir dos erros dos processadores aos quais ele está conectado diretamente e dos seus pesos. Para o caso de função de ativação sigmoidal tem-se:

$$\delta_h^{(p)} = F'\left(s_h^{(p)}\right)\sum_{o=1}^{N_o}\delta_o^{(p)}w_{oh} = y_h^{(p)}\left(1-y_h^{(p)}\right)\sum_{o=1}^{N_o}\delta_o^{(p)}w_{oh} \tag{13}$$

onde

$\delta_h^{(p)}$ Erro do processador da camada escondida h para o padrão $x^{(p)}$

$F\left(s_h^{(p)}\right)$ Ativação no processador da camada escondida para o padrão $x^{(p)}$

w_{oh} Peso da conexão entre o processador da camada escondida e o processador da camada de saída

N_o Número de processadores da camada de saída

3.2.3.1. Termo de momento

Termo adicionado ao processo de atualização dos pesos com o objetivo de acelerar a convergência, evitando oscilações.

A atualização dos pesos fica da seguinte forma:

$$\Delta w_{kj}(t+1) = \gamma \delta_k^{(p)} . y_j^{(p)} + \alpha \Delta w_{kj}(t) \quad (14)$$

onde

α Taxa de momento

$t, t+1$ última e próxima iterações

Apesar do grande êxito do algoritmo de aprendizado *back-propagation*, existem algumas deficiências, começando pelo longo tempo de treinamento devido principalmente a valores de taxa de aprendizado e momento não adequados. Outras falhas que existem são: paralisia da rede e o problema de cair no mínimo local.

3.2.3.2. Validação Cruzada

O objetivo do aprendizado *Back-Propagation* é obter um mapeamento de entrada-saída. Uma rede bem treinada significa que esta aprendeu o modelo suficientemente bem para ter boas estimativas no futuro. A partir desta visão, o processo de aprendizado é uma escolha da parametrização da rede para o conjunto de dados, isto é, escolher a partir de algumas parametrizações "candidatas" a melhor segundo algum critério.

Desta forma, a validação cruzada (Stone, 1974, 1978) é um princípio atraente onde o conjunto de dados é particionado em dados para treinamento e dados para teste, sendo que os dados de treinamento são subdivididos em dados para estimação e dados para validação.

A idéia é utilizar o conjunto de treinamento para avaliar o desempenho dos modelos candidatos e assim, escolher o melhor. O subconjunto de treinamento

permite selecionar o modelo e o subconjunto de validação permite validar o modelo escolhido. Já com o conjunto de testes é verificada a generalização do modelo, isto para evitar o efeito de sobre-ajuste (*overfitting*).

Assim, a validação cruzada é útil tanto para escolher a melhor arquitetura de rede, isto é a seleção de modelos, ou para decidir a hora de finalizar o processo de treinamento (ajuste dos parâmetros).

3.2.4. Redes RBF

Para entender este tipo de rede, devem ser explicadas as funções de Base Radial (*Radial Basis Functions*) que são um tipo especial de funções que cumprem a seguinte característica: sua resposta cresce ou decresce, de forma monotônica à medida que o argumento se afasta de um determinado valor central (Orr, 1996, 1999). O ponto central, o parâmetro de escala de distância e a forma da função são parâmetros do modelo que, caso fosse linear, teriam valores fixos. Uma função de base radial bem conhecida é a curva *gaussiana* que, para dimensão 1, é dada pela expressão:

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right) \tag{15}$$

com parâmetros:

c	centro
r	raio

Na Figura 5 a seguir, mostra-se uma função *gaussiana* RBF definida para valores c = 0 e r = 1.

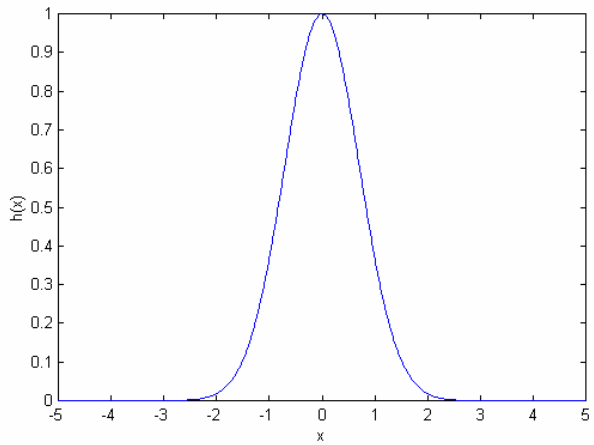

Figura 5. Função Gaussiana para c = 0, r = 1

Uma gaussiana RBF decai de forma monotônica de acordo com a distância ao centro. Outra função RBF muito utilizada é a multiquadrática que, para dimensão 1, tem a seguinte expressão:

$$h(x) = \frac{\sqrt{r^2 + (x - c)^2}}{r^2} \tag{16}$$

Mostra-se na Figura 6 a seguir a função multiquadrática correspondente.

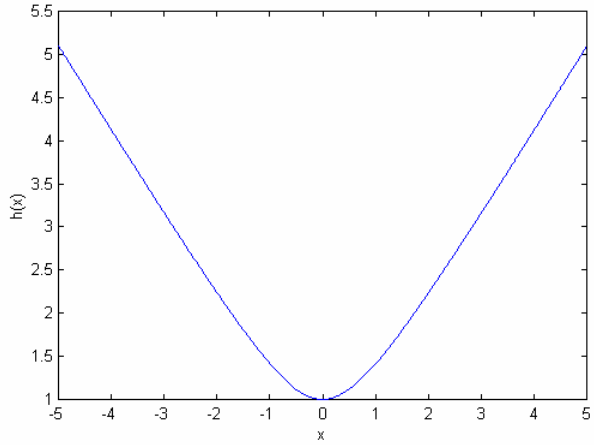

Figura 6. Função multiquadrática para c = 0, r = 1

Neste caso a função *cresce* de forma monotônica à medida que se afasta do centro. Outro aspecto das funções RBF é quanto a serem locais ou globais, o que se refere ao grau de variabilidade da resposta. Como pode ser percebido nas figuras anteriores, no caso da *gaussiana*, a variabilidade só existe em uma vizinhança do centro, e no caso da multiquadrática esta possui variabilidade em todo o domínio.

3.2.4.1. Redes Neurais baseadas em Funções RBF

As funções radiais quando utilizadas dentro de uma arquitetura de rede neural com uma ou várias camadas intermediárias geram uma rede neural baseada em funções radiais. No artigo de (Broomhead & Lowe, 1988), as redes baseadas em funções radiais (rede RBF) tomam a forma clássica de funções radiais dentro de uma rede *feed-forward* com uma camada escondida, como ilustrado na figura a seguir.

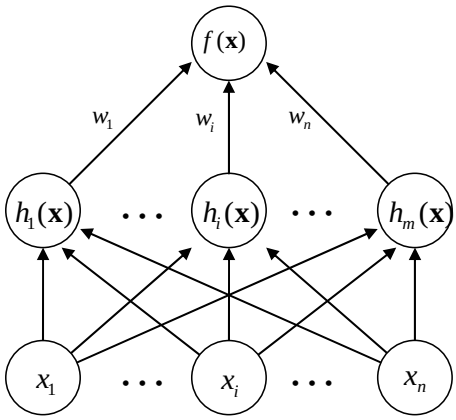

Figura 7. Rede RBF típica

Na Figura 7, cada um dos n componentes do vetor de entradas $\mathbf{x}$, alimenta cada uma das m funções radiais, cujas saídas são combinadas linearmente usando os pesos w_j para formar a saída $f(\mathbf{x})$.

3.2.4.2. Treinamento das redes RBF

O treinamento visa minimizar o erro quadrático médio entre a saída da rede e a saída desejada. Para isto devem se estabelecer os parâmetros ajustáveis da rede que são os pesos, os centros e os raios das funções radiais.

Uma forma bastante prática de treinar redes RBF é a seguinte:

Partindo de uma rede de duas camadas, a primeira com funções radiais e a segunda de saída com neurônios de ativação linear, de forma similar à rede mostrada na Figura 7.

Inicialmente, a camada de funções radiais não possui neurônios. Segue-se o seguinte algoritmo até alcançar o erro quadrático médio desejado ou o número máximo de neurônios permitido:

- *recall* da rede;
- achar o padrão de entrada com máximo erro;
- inserir um novo neurônio com função radial $h_i(\mathbf{x})$ com pesos iguais ao padrão anteriormente achado;
- atualizar os pesos w_i da camada linear (saída) minimizando o erro;

3.2.4.3.
Propriedades aproximativas das redes RBF

Na literatura de redes neurais, existem discussões sobre as propriedades aproximativas para *perceptrons* de múltiplas camadas. As redes baseadas em funções radiais mostram boas propriedades como aproximadores de funções, comparáveis às propriedades dos *perceptrons* de múltiplas camadas. Isto é, a família de redes RBF é suficientemente extensa para aproximar qualquer função contínua sobre um conjunto compacto.

3.3.
Modelos *Neuro-Fuzzy* Hierárquicos

Sistemas *neuro-fuzzy* (Jang, 1993, 1995) são sistemas híbridos, pois utilizam mais de uma técnica de identificação de sistemas para solução de um problema de modelagem. Essa mistura de técnicas se reflete na obtenção de um sistema melhorado em termos de interpretação, de aprendizado, de estimação de parâmetros, de generalização, etc. Os sistemas *neuro-fuzzy* combinam a capacidade de aprendizado das redes neurais artificiais com o poder de interpretação lingüística dos sistemas de inferência *fuzzy*. A idéia básica de um sistema *neuro-fuzzy* é implementar um sistema de inferência *fuzzy* numa arquitetura paralela distribuída de tal forma que os paradigmas de aprendizado comuns às redes neurais possam ser aproveitados nessa arquitetura híbrida.

O principal objetivo de se criar um modelo *neuro-fuzzy* hierárquico é que o mesmo possui, devido a seu particionamento do espaço de entrada, capacidade ilimitada de criar e expandir sua própria estrutura. Além disso, este particionamento reduz a limitação dos modelos *neuro-fuzzy* convencionais quanto ao reduzido número de entradas. Dentre os tipos de particionamento hierárquico existentes, o particionamento binário (*Binary Space Partitioning* BSP), de onde provém o sistema neuro-fuzzy hierárquico com particionamento binário NFHB (De Souza, 1999), foi selecionado.

Este método de particionamento é descrito a seguir.

3.3.1.
Particionamento BSP

No particionamento BSP, o espaço é dividido sucessivamente, de forma recursiva, em duas regiões. Este particionamento pode ser representado por uma árvore binária (BSP *tree*) que ilustra as sucessivas sub-divisões do espaço

n-dimensional em subespaços fechados. O processo de construção desta árvore toma um subespaço e o divide por um hiperplano que passa pelo interior deste subespaço. Isto resulta em dois novos subespaços que podem ser posteriormente particionados pela aplicação do mesmo método (recursividade).

O hiperplano divisor deste espaço n-dimensional é, neste caso, um objeto de dimensão $n-1$, que pode ser usado para dividir o espaço em duas partes não necessariamente iguais. A Figura 8, ilustra, para o caso bidimensional, um exemplo deste tipo de particionamento (a) e sua respectiva árvore representativa (b).

Conforme ilustrado, cada partição final é representada por nós-folhas na árvore BSP. Os nós interiores representam as diversas partições intermediárias realizadas. Este particionamento também é conhecido como K-d tree ou particionamento H-V (horizontal-vertical).

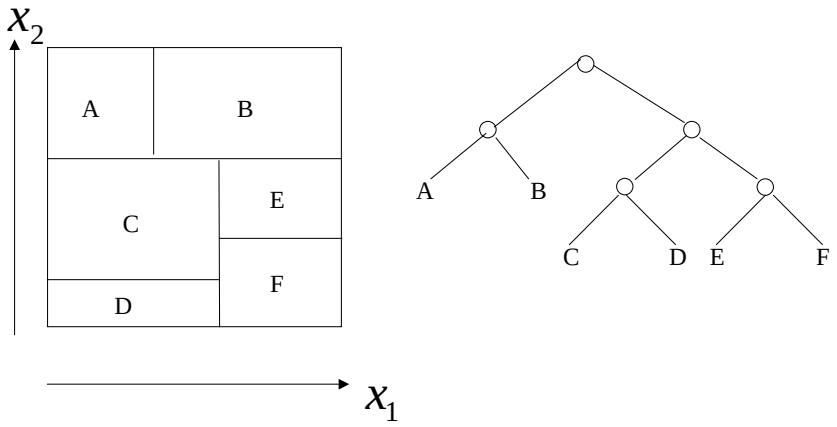

Figura 8. (a) Particionamento BSP. (b) árvore BSP correspondente

3.3.2. Modelo *Neuro-Fuzzy* Hierárquico BSP (NFHB)

O modelo NFHB (De Souza, 1999) é composto de uma ou várias células padrão chamadas células básicas *neuro-fuzzy* BSP. Estas células estão dispostas numa estrutura hierárquica de árvore binária. A célula de maior hierarquia gera a saída. As de menor hierarquia trabalham como conseqüentes das células de maior hierarquia. As células intermediárias e a de saída têm como conseqüentes as saídas das células de menor hierarquia.

3.3.2.1.
Célula Básica NFHB

Uma célula NFHB é um micro-sistema *neuro-fuzzy* que realiza um particionamento *fuzzy* binário em um determinado segmento do espaço. A célula NFHB gera uma saída precisa (*crisp*) após o processo de defuzzificação.

A Figura 9(a) ilustra o processo de defuzzificação da célula e o encadeamento dos conseqüentes. Nesta célula, a entrada x gera os antecedentes das duas regras *fuzzy* após serem computados os graus de pertinência $\rho(x)$ e $\mu(x)$ onde ρ é o conjunto nebuloso baixo e μ é o conjunto nebuloso alto.

A Figura 9(b) ilustra a representação desta célula de forma simplificada.

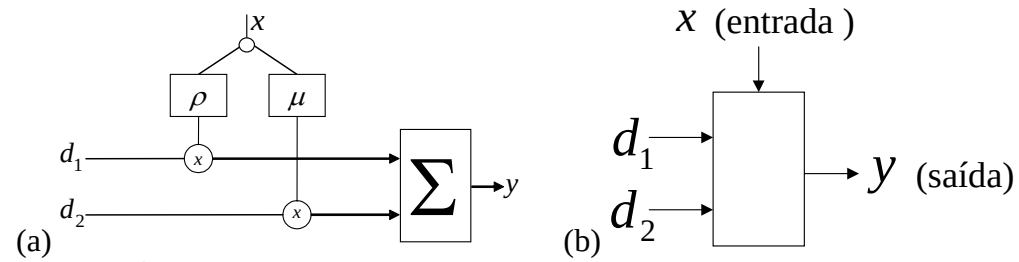

Figura 9. Célula *Neuro-Fuzzy* BSP (a) Interior e (b) Simplificada

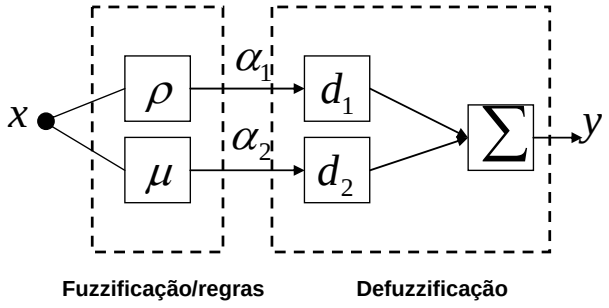

Figura 10. Representação da célula NF-BSP em formato de rede neuro-fuzzy

Nesta célula, x representa a entrada, ρ e μ são as funções de pertinência (complementares) que geram os antecedentes das duas regras. A interpretação lingüística do mapeamento implementado pela célula NFHB é dada pelo seguinte conjunto de regras:

Regra1 : Se $x \in \rho$ então $y = d_1$

Regra2 : Se $x \in \mu$ então $y = d_2$

3.3.2.2. Arquitetura NFHB

Um modelo NFHB pode ser descrito como um sistema composto por interligações de células NFHB. A Figura 11 ilustra um sistema NFHB juntamente com o particionamento resultante do espaço de entrada.

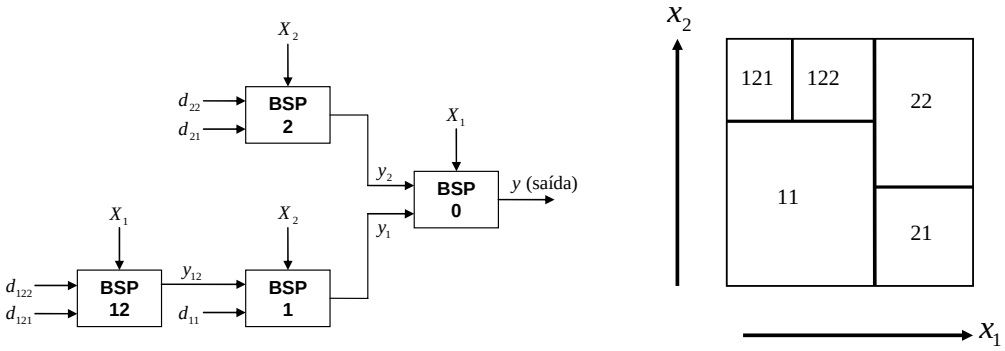

Figura 11. Exemplo de Sistema NFHB e o seu respectivo particionamento

Neste sistema, as partições iniciais 1 e 2 (célula 'BSP 0') foram subdivididas, portanto, os conseqüentes de suas regras são as saídas dos subsistemas 1 e 2, respectivamente. Estes, por sua vez têm, como conseqüentes, os valores d_{11}, y_{12}, d_{21} e d_{22}, respectivamente. Se estiver utilizando conseqüentes de Sugeno de ordem 1 (Jang, 1995) cada d_i corresponde a uma combinação linear das entradas como:

$$d_i = \sum_{k=0}^{n} w_k x_k \tag{17}$$

A saída do sistema da Figura 11 é dada pela equação (18) a seguir

$$y = \alpha_1\left(\alpha_{11} d_{11} + \alpha_{12}\left(\alpha_{121} d_{121} + \alpha_{122} d_{122}\right)\right) + \alpha_2\left(\alpha_{21} d_{21} + \alpha_{22} d_{22}\right) \tag{18}$$

onde $\alpha_{ij\ldots}$ são os níveis de disparo das regras de cada bipartição $ij\ldots$ respectiva, e os $d_{ij\ldots}$ são os conseqüentes (*singletons* ou combinações lineares) das regras existentes.

3.3.2.3. Algoritmo de Aprendizado

O aprendizado do sistema NFHB pode ser visto em dois estágios: aprendizado da estrutura do modelo que define as regras lingüísticas, onde é utilizado o método do gradiente descendente, e atualização dos pesos do

sistema *neuro-fuzzy*, que definem os formatos das funções de pertinência dos antecedentes e conseqüentes. Os parâmetros d_i e os parâmetros a e b são os pesos *fuzzy* do modelo.

É importante ressaltar que pode existir o crescimento indefinido da estrutura do modelo, e para lidar com isto foi criado o parâmetro de taxa de decomposição. Este parâmetro adimensional, age como um limitador no processo de decomposição. Mais detalhes do funcionamento do algoritmo de aprendizado para o modelo NFHB podem ser vistos em (De Souza, 1999).

O modelo *neuro fuzzy* hierárquico com particionamento binário (NFHB) vem sendo aplicado em diferentes tipos de problemas: classificação de padrões, extração de regras, aproximação de funções, previsão entre outros. Para estas aplicações foram pesquisados e implementados outros modelos que utilizam o NFHB. Estas implementações podem ser revisadas na seguinte literatura (De Souza, 2002; Gonçalves, 2005; Bezerra, 2003; Contreras, 2003; Figueiredo 2004)

3.4. Algoritmos Genéticos (AG)

Essencialmente, Algoritmos Genéticos são métodos de busca e otimização (Melanie, 1994; Koza, 1992; Goldberg, 1989; Michalewicz, 1996), que têm sua inspiração nos conceitos da teoria de seleção natural das espécies.

Os sistemas desenvolvidos a partir deste princípio são utilizados para procurar soluções de problemas complexos ou que possuem espaço de busca muito grande, o que os tornam problemas de difícil modelagem e solução quando se aplicam métodos de otimização clássicos.

Estes algoritmos são baseados nos processos genéticos de organismos biológicos para procurar soluções ótimas ou sub-ótimas. Para tanto, procede-se da seguinte maneira: codifica-se cada possível solução do problema em uma estrutura chamada "cromossomo". Estes cromossomos representam indivíduos, que são evoluídos ao longo de várias gerações, de forma similar aos seres vivos, de acordo com os princípios de seleção natural e sobrevivência dos mais aptos. Emulando estes processos, os Algoritmos Genéticos são capazes de "evoluir" soluções de problemas do mundo real.

O processo de evolução começa com a criação aleatória dos indivíduos que formarão a população inicial. A partir de um processo de seleção baseado

na aptidão de cada indivíduo, são escolhidos indivíduos para a fase de reprodução que cria novas soluções, utilizando-se, para isto, um conjunto de operadores genéticos (basicamente cruzamentos e mutações) e calculando as aptidões dos indivíduos resultantes. As aptidões dos novos indivíduos determinam a probabilidade de serem selecionados e evoluídos nas próximas gerações.

O pseudo código da Figura 12 mostra o procedimento básico de um algoritmo genético (Lawrence, 1991).

```
Inicio
  t←1
  inicializar população P(t)
  avaliar população P(t)
  enquanto (não condição_de_fim) faça
    t←t+1
    selecionar população P(t) a partir de P(t-1)
    aplicar operadores genéticos
    avaliar população P(t)
  fim enquanto
fim
```

Figura 12. Procedimento básico do algoritmo genético

Como critério de parada pode-se fixar o número de gerações, o número de total de indivíduos criados, ou alcançar uma solução satisfatória. Outras condições para a parada incluem o tempo de processamento e o grau de similaridade entre os elementos numa população (critério de convergência).

As seções seguintes apresentam em mais detalhes cada um dos componentes de um algoritmo genético.

3.4.1. Representação

A representação é uma questão fundamental na modelagem de um AG para a solução de um problema. Ela define a estrutura do cromossomo, com os respectivos genes que o compõem, de maneira que seja capaz de descrever uma solução e varrer todo o espaço de busca relevante do problema.

A solução de um problema pode ser representada por um conjunto de parâmetros (genes), concatenados para formar uma cadeia de valores (cromossomo); a este processo chama-se codificação. As soluções (cromossomos) são codificadas através de uma seqüência formada por caracteres de um sistema alfabético. Originalmente, utilizou-se o alfabeto binário

(0, 1), porém, novos modelos de algoritmos genéticos codificam as soluções de forma mais abrangente como por exemplo com números reais (Michalewicz, 1996).

A decodificação do cromossomo consiste na recuperação da solução real do problema a partir da informação existente no cromossomo. Uma vez construída esta solução pode ser enviada para ser avaliada pelo problema.

3.4.2. Avaliação

A avaliação é a ligação entre o *framework* do AG e o problema a ser solucionado. A avaliação é realizada através de uma função objetivo que depende do problema. Dado um cromossomo, a função de avaliação retorna um valor numérico, que representa a utilidade do indivíduo representado em solucionar o problema em questão. O objetivo do Algoritmo Genético inteiro é encontrar o máximo ou mínimo global desta função, recaindo no problema de otimização clássico.

A partir do valor de avaliação é possível obter um outro valor chamado a aptidão como função do valor de avaliação. Normalmente o valor de aptidão é obtido a partir de ajustes do valor de avaliação, com a finalidade de facilitar o processo de evolução.

3.4.3. Seleção e Reprodução

A seleção refere-se ao processo de escolha e cópia de um determinado cromossomo para a população seguinte de acordo com sua aptidão. Isto significa que os cromossomos mais aptos (valor de aptidão maior) têm maior probabilidade de contribuir para a formação de um ou mais indivíduos da população seguinte. Existem basicamente os seguintes métodos: troca de toda a população; troca de toda a população com elitismo, onde todos os cromossomos são substituídos sendo o cromossomo mais apto da população corrente copiado para população seguinte; e troca parcial da população (*steady state*), onde os M melhores indivíduos da população corrente são copiados para a população seguinte.

3.4.4.
Operadores Genéticos

Os operadores mais conhecidos nos AG's são os de Reprodução, Cruzamento (*crossover*) e Mutação, descritos a seguir.

3.4.4.1.
Cruzamento

É um operador baseado na troca de partes dos cromossomos (pais), formando-se duas novas soluções (filhos). Este processo pode ser observado no exemplo a seguir (Figura 13), onde a solução está codificada com alfabeto binário.

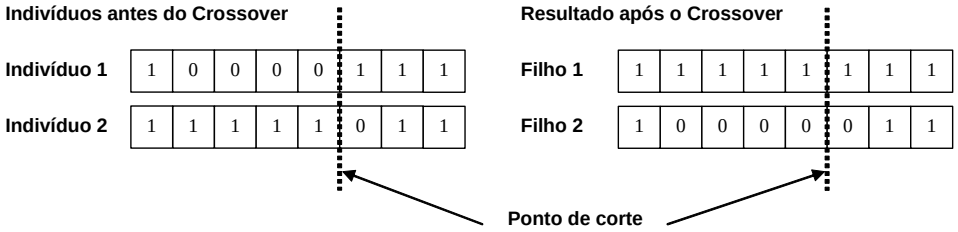

Figura 13. Cruzamento de um ponto.

O ponto onde ocorre o corte para a realização do cruzamento é escolhido aleatoriamente; no exemplo da Figura 13 utilizou-se um único ponto, mas podem ser realizados cortes em mais de um ponto, caracterizando o cruzamento multiponto (Goldberg, 1989; Michalewicz, 1996; Holland, 1992). Para realizar o cruzamento, primeiro é necessária a escolha, por sorteio, dos cromossomos genitores. Em seguida ocorre a realização ou não do cruzamento segundo um parâmetro, denominado taxa de cruzamento. Deste modo, de acordo com a taxa de cruzamento, pode ocorrer que os cromossomos genitores sejam repassados sem alteração para a geração seguinte, criando descendentes idênticos a eles.

A idéia do operador de cruzamento é tirar vantagem (*exploit*) do material genético existente na população.

3.4.4.2.
Mutação

É a troca aleatória do valor contido nos genes de um cromossomo por outro valor válido do alfabeto ou pertencente ao domínio do gene. No caso de alfabeto binário troca-se de 0 para 1 e vice-versa. Da mesma forma que para o cruzamento, utiliza-se uma taxa de mutação que, para cada bit da seqüência de caracteres, sorteia-se se ocorrerá ou não a mutação; no caso de ocorrência, o bit será trocado por outro valor válido como mostra a Figura 14, a seguir.

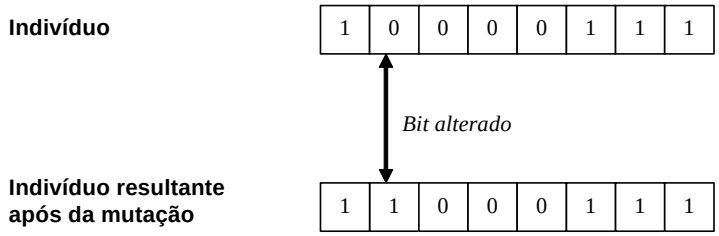

Figura 14. Mutação binária.

A mutação garante a diversidade das características dos indivíduos da população e permite que sejam introduzidas informações que não estavam presentes em nenhum dos indivíduos. Além disto, proporciona uma busca aleatória (*exploration*) no AG, oferecendo oportunidade para que mais pontos do espaço de busca sejam avaliados.

Existem variações do operador mutação que mantêm a característica de introdução de novos indivíduos na população: mutação uniforme, mutação local, mutação de fronteira. Estas variações podem ser vistas com maior detalhe em (Michalewicz, 1996)

3.4.5.
Parâmetros da Evolução

A seguir mostram-se os parâmetros que mais influenciam no desempenho do AG. Estes parâmetros normalmente apresentam um ponto de equilíbrio (*trade-off*) que varia de acordo com o problema sendo abordado.

Tamanho da População Afeta o desempenho global e a eficiência dos AG's. Uma população muito pequena oferece uma pequena cobertura do espaço de busca, causando uma queda no desempenho. Uma população suficientemente

grande fornece uma melhor cobertura do domínio do problema e previne a convergência prematura para soluções locais. Entretanto, com uma grande população tornam-se necessários recursos computacionais maiores, ou um tempo maior de processamento do problema. Logo, deve-se buscar um ponto de equilíbrio no que diz respeito ao tamanho escolhido para a população.

Taxa de Cruzamento É a probabilidade de um indivíduo ser recombinado com outro. Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Entretanto, isto pode gerar um efeito indesejável pois a maior parte da população será substituída, podendo ocorrer perda de estruturas de alta aptidão e convergência para uma população com indivíduos extremamente parecidos, indivíduos estes de solução boa ou não. Com um valor baixo, o algoritmo pode tornar-se muito lento para oferecer uma resposta aceitável.

Taxa de Mutação É a probabilidade do conteúdo de um gene do cromossomo ser alterado. A taxa de mutação previne que uma dada população fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Porém deve-se evitar uma taxa de mutação muito alta uma vez que pode tornar a busca essencialmente aleatória, prejudicando fortemente a convergência para uma solução ótima.

Taxas de Operadores Adaptativas Os operadores genéticos podem resultar mais úteis no começo ou no final do processo de otimização. Deste modo, as taxas dos operadores deve ser adaptativas, permitindo variar seus valores durante as gerações. No início, a taxa de cruzamento deve ser maior, de forma a aproveitar melhor o material genético inicial; no final da rodada, quando aparecem os efeitos da convergência, deve-se dar ênfase à busca de novo material aumentando a taxa de mutação (Davis, 1989, 1991).

Intervalo de Geração Controla a porcentagem da população que será substituída durante a próxima geração (substituição total, substituição com elitismo, substituição dos piores indivíduos da população atual, substituição parcial da população sem duplicatas). Esse número de indivíduos substituídos também é conhecido como GAP.

Número de gerações Representa o número total de ciclos de evolução de um Algoritmo Genético, sendo este um dos critérios de parada do AG. Um número de gerações muito pequeno causa uma queda no desempenho, pois não se consegue cobrir todo o espaço de busca. Um grande valor requer um tempo maior de processamento, mas fornece uma melhor cobertura do domínio do problema evitando a convergência para soluções locais.

3.4.6. Avaliação de um algoritmo genético

Uma das formas de avaliar o bom desempenho de um algoritmo genético é através das curvas de evolução: curva *off-line* e curva *online.*

A curva *off-line* é obtida a partir da média das melhores soluções de todas as gerações, isto é, para uma dada geração, o valor *off-line* é calculado pela média atirmética de todos os melhores elementos obtidos até o momento, mesmo se forem valores repetidos. Esta curva mostra a qualidade do algoritmo em encontrar soluções desde o início, isto é, a curva *off-line* premia a qualidade das soluções sem importar o número de gerações.

A curva *on-line* é obtida a partir da média aritmética das avaliações de todos os indivíduos realizadas até o momento. Esta curva permite verificar a rápida obtenção de boas soluções e também permite visualizar o grau de convergência dos indivíduos da população.

3.4.7. Algoritmos Genéticos Distribuídos

O uso de computação paralela e processos distibuídos é uma tecnologia que vem crescendo nestes últimos anos, dada a crescente demanda por recursos computacionais a baixo custo, aproveitamento da capacidade ociosa em ambientes colaborativos de pesquisa e desenvolvimento e pela evolução dos sistemas de rede para sistemas de alto desempenho. O objetivo principal é atingir melhor desempenho partindo do princípio de se juntar processadores, memória e uma rede a fim de em conjunto trabalharem na solução de um dado problema.

Nesta seção são apresentados alguns conceitos sobre Algoritmos Genéticos Distribuídos que vem a ser os próprios algoritmos genéticos inseridos em uma arquitetura de computação paralela.

3.4.7.1.
Modelos de Algoritmos Genéticos Distribuídos

Um algoritmo genético, ao ser distribuído em um ambiente de computação paralela, torna-se conveniente pelos dois seguintes motivos:

- Ganho de tempo de processamento pela distribuição do esforço computacional entre os processadores do ambiente paralelo;
- Vantagens dadas pelo próprio ambiente paralelo pois surgem analogias com a evolução natural existente em populações que estão distribuídas espacialmente.

Um primeiro tipo de algoritmo genético distribuído usa os processadores para executar problemas independentes. Isto resulta bastante simples em virtude de não haver comunicação necessária entre os diferentes processadores; por isso é chamado de *algoritmo paralelo trivial*. Contudo, este método tão elementar resulta muito útil na execução de várias versões do mesmo problema com diferentes populações iniciais, permitindo gerar estatísticas de desempenho. Dada a natureza estocástica do AG, a obtenção de estatísticas é muito relevante.

Outra forma de aproveitar o ambiente paralelo pode ser executando várias versões do mesmo problema com valores diferentes nos parâmetros do AG (taxas de cruzamento e mutação, tamanho da população, número de gerações). Resulta evidente que, nenhuma destas metodologias acrescenta novidades na natureza do AG, mas o tempo de execução pode ser reduzido de forma considerável.

Existem outros modelos de algoritmos genéticos distribuídos, nos quais partes específicas da estrutura do algoritmo genético como a população, a avaliação ou a reprodução, são distribuídas dentro do ambiente paralelo.

3.4.7.1.1.
Algoritmos Genéticos Distribuídos Globais

Refere-se aos algoritmos genéticos cujo módulo de avaliação é a parte executada dentro do ambiente paralelo. Ainda não surgem grandes mudanças na arquitetura do algoritmo evolucionário, dado que, na maioria dos casos, a avaliação de um indivíduo da população independe das avaliações dos outros

indivíduos da mesma população. Contudo, surgem vantagens no sentido de melhor aproveitamento do poder computacional paralelo, dado que, na maioria dos problemas reais, o maior percentual de tempo do algoritmo genético é empregado no cálculo da avaliação. Neste caso, é condição necessária para um melhor aproveitamento do poder paralelo que os tempos de comunicação entre os computadores sejam bem menores do que o tempo levado no cálculo da avaliação.

A opção mais imediata é a avaliação simultânea dos indivíduos da população empregando os diferentes processadores da arquitetura paralela. Na arquitetura *master-slave* (Cantú-Paz, 1997), um processador *master* executa o ciclo principal do algoritmo genético e, na hora de avaliar uma população de p indivíduos, envia os p indivíduos (cromossomas) para os n processadores *slave* disponíveis de forma a serem avaliados. Uma vez obtidos os valores de avaliação, o processador *master* continua com o ciclo do algoritmo genético aplicando os processos de seleção e evolução para obter a próxima geração de indivíduos.

A situação mais freqüente é que o número de processadores *slaves* seja menor que o número de indivíduos da população; isto leva à questão de equidade na distribuição das avaliações dos indivíduos nos n processadores. Isto deve levar em consideração o poder computacional de cada processador no momento da distribuição dos indivíduos. A seguir mostra-se graficamente a arquitetura *master-slave.*

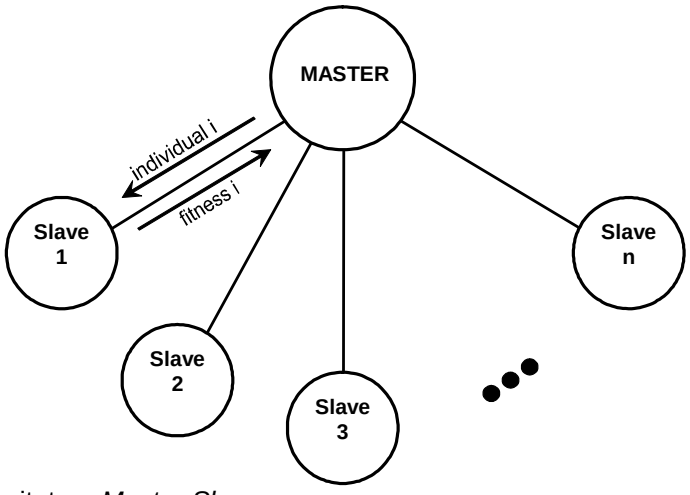

Figura 15. Arquitetura *Master-Slave*

O pseudo algoritmo que funciona para este tipo de distribuição é apresentado a seguir.

```
inicializa a primeira população
para todos os indivíduos executar em paralelo:
  avaliação
fim do executar em paralelo
enquanto não cumprir a condição de finalização
  seleção
  cruzamento
  mutação
  para todos os indivíduos executar em paralelo
    Avaliação
  fim do executar em paralelo
  elitismo
fim do enquanto
```

Figura 16. Pseudo-código: Algoritmo Genético Global

3.4.7.1.2.
Algoritmos Genéticos Distribuídos em Ilhas

Neste modelo, a distribuição do algoritmo genético é feita ao nível de indivíduos ou população. Este tipo de distribuição está inspirada no fato que as populações na natureza possuem uma disposição espacial. Existem grupos de indivíduos ou sub-populações que não são independentes entre si, pois mantêm um leve acoplamento entre sub-populações vizinhas. Cada uma destas entidades é denominada *demes.*

O acoplamento entre estas sub-populações manifesta-se como uma lenta migração ou difusão de elementos de uma para outra. Dentre os vários modelos propostos para esta estrutura existem dois que cabem ressaltar.

Modelos em Ilhas

O modelo em ilhas (Cohoon, 1987) caracteriza-se por ter sub-populações separadas fisicamente com tamanho relativamente grande. A cada certo intervalo de tempo, estas sub-populações trocam informações, permitindo que alguns indivíduos migrem de uma população para outra segundo um padrão estabelecido. O propósito disto é manter a diversidade populacional em quaisquer sub-populações durante o processo de convergência. É esperado que as diferentes sub-populações estejam explorando diferentes partes do espaço de busca. Dentro de cada sub-população executa-se um algoritmo genético clássico (não paralelo) durante os tempos intra-migrativos. Se a migração ocorre entre vizinhos próximos o modelo se chama *stepping stone.*

A seguir o pseudo–código do modelo.

```
Inicializar P subpopulações de N indivíduos
Número_gerações = 1
enquanto não cumprir a condição de finalização
  para cada subpopulação executar em paralelo
    avaliação
    seleção pela aptidão
    se numero_gerações mod freqüência = 0  então
      enviar K < N melhores indivíduos para uma subpopulação
      vizinha
      receber K indivíduos da subpopulação vizinha.
      substituir K indivíduos na subpopulação
    fim (se - então)
    cruzamento
    mutação
  fim do executar em paralelo
  número_gerações += 1
fim do enquanto
para todos os indivíduos executar em paralelo:
  avaliação
fim do executar em paralelo
```

Figura 17. Pseudo-código do Modelo AG em Ilhas

No pseudo-código apresentado na Figura 17, `freqüência` é o numero de gerações a se obter antes de trocar indivíduos entre sub-populacões. Para a migração existem várias políticas de substituição; uma das mais usadas é substituir os K piores indivíduos de uma subpopulação pelos K elementos vindos na migração. Nota-se que existem novos parâmetros no algoritmo como: tamanho da subpopulação, freqüência de troca, numero K de indivíduos migrantes, topologia de migração e outros.

Na Figura 18 a seguir mostra-se um exemplo com 5 populações quase-isoladas.

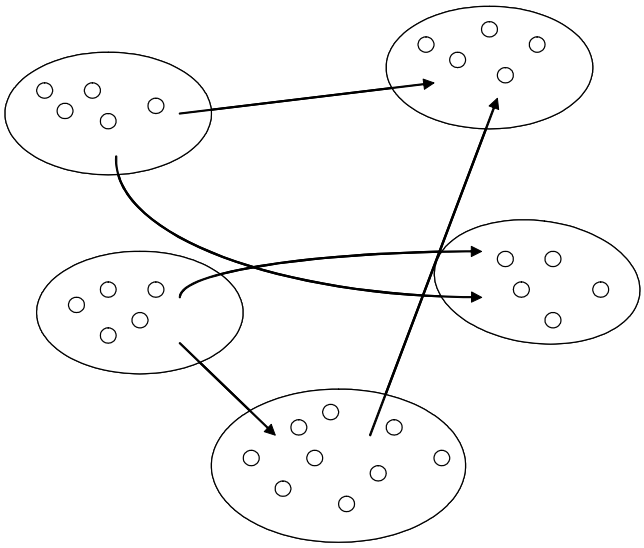

Figura 18. Arquitetura de Populações quase-isoladas

3.4.7.1.3.
Algoritmos Genéticos Distribuídos Celulares

Conhecidos como modelos de malha ou granulado fino (Manderick, 1989). Neste modelo os indivíduos são colocados sobre uma forma toroidal como uma malha de uma ou duas dimensões, com um indivíduo para cada posição da malha. Este modelo é chamado celular pela semelhança com os autômatos celulares com regras de transição estocásticas (Tomassini, 1993).

Neste modelo, a avaliação é feita de forma simultânea para todos os indivíduos, enquanto que a seleção e reprodução são feitas localmente dentro de uma pequena vizinhança. Ao passar o tempo, os *clusters* quase-isolados, com indivíduos homogêneos, emergem através do *grid* como resultado de uma lenta difusão individual. Este fenômeno é chamado de isolação pela distância, devido ao fato que a probabilidade de dois indivíduos interagirem é inversamente proporcional à distância entre eles. Segue, na Figura 19 o pseudo código do modelo *grid*.

```
para cada célula i executar em paralelo
  gerar um indivíduo randômico i
fim do executar em paralelo
enquanto não cumprir a condição de finalização
  para cada célula i executar em paralelo
    avaliação indivíduo i
    seleção um indivíduo vizinho j
    criar os filhos de i e j
    substituir um dos filhos em i
    mutação de i com probabilidade pmut
  fim do executar em paralelo
  número_gerações += 1
fin do enquanto
```

Figura 19. Pseudo-código para o modelo AGs celulares.

Geralmente, a vizinhança está formada de 4 ou 8 vizinhos próximos de um dado ponto da malha, no caso de *1-D* é um pequeno número de celas fora do centro que são levadas em consideração. A seleção do indivíduo na vizinhança para se “cruzar” com o indivíduo central pode ser feita de diferentes formas: seleção por sorteio, que é a mais utilizada, uma vez que se encaixa muito bem com a natureza espacial do sistema. Assim, com um sorteio local extraem-se k indivíduos da população com probabilidade uniforme sem re-inserção; em seguida dentre esses k indivíduos é sorteado o indivíduo a se cruzar com o

indivíduo central: utilizando as aptidões normalizadas dos k indivíduos como probabilidades deles.

Desta forma, o paralelismo computacional é bem aproveitado. O modelo pode ficar ainda mais dinâmico se forem acrescentadas às interações entre vizinhos, movimentações amplas de alguns indivíduos seguindo modelos de passeio aleatório.