IMPLEMENTAÇÃO DE UM ALGORITMO GENÉTICO UTILIZANDO O MODELO DE ILHAS

Angelo José Moreira Silva

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL

Aprovada por:

_____

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

_____

Dra. Myrian Christina de Aragão Costa, D.Sc.

_____

Prof. Mario Antonio Ribeiro Dantas, Ph.D.

_____

Prof. Beatriz de Souza Leite Pires de Lima. D.Sc.

RIO DE JANEIRO, RJ - BRASIL

AGOSTO DE 2005

MOREIRA SILVA, ANGELO JOSÉ

Implementação de um Algoritmo Genético utilizando o Modelo de Ilhas [Rio de Janeiro] 2005

XI, 73p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Civil, 2005)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. algoritmos genéticos, otimização, programação paralela

I. COPPE/UFRJ II. Título (série)

Aqueles que são iluminados nunca param de forjar a si mesmos. A realização de tais mestres não pode ser expressa em palavras ou por teorias. As mais perfeitas ações são o eco do modelo encontrado na natureza. (**Morihei Ueshiba - A Arte da Paz**)

Agradecimentos

Ao meu pai, esteja onde Deus o colocou, inspiração, caráter e conduta não somente palavras, mas ações em toda a sua existência.

As minhas filhas Priscilla e Gabrielle por me segurarem nas quedas dos piores momentos e por me abençoarem com a dádiva de poder ser pai.

A minha mãe, aos meus irmãos e a Claudia pelo carinho e força.

Ao professor Nelson pela orientação e pelo estímulo que foi indispensável à execução desta tese.

A professora Myrian não tenho palavras que possam descrever a importância que teve ao longo de toda esta etapa em minha vida. As discordâncias serviram para aumentar o respeito, a amizade e o carinho.

Aos amigos Valeriana, Leonardo e Paula pelo ajuda que não há como ser paga.

Ao CNPq, que viabilizou a realização deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

IMPLEMENTAÇÃO DE UM ALGORITMO GENÉTICO UTILIZANDO O MODELO DE ILHAS

Angelo José Moreira Silva

Agosto/2005

Orientadores: Nelson Francisco Favilla Ebecken
Myrian Christina de Aragão Costa

Programa: Engenharia Civil

Esta dissertação apresenta a implementação de um algoritmo genético paralelo utilizando o modelo de ilhas. No algoritmo de ilhas, populações aleatórias são geradas de maneira independente em cada uma das ilhas, que ficam restritas a processadores específicos. A obtenção de uma melhora global se dá com as evoluções independentes das ilhas e com a migração de indivíduos entre as mesmas a partir de critérios determinados. A troca de informações entre as ilhas se faz através da biblioteca de troca de mensagens MPI. Diferentes topologias lógicas estão sendo analisadas para a fase de de migração de indivíduos entre as ilhas, sendo utilizada especificamente a topologia em anel. Resultados experimentais foram gerados com a utilização de exemplos da literatura e analisados e comparados com outras implementaçãoes. O objetivo dessa dissertação é a disponibilização de uma ferramenta paralela de mineração de dados de alto de desempenho para, por exemplo, otimizar a arquitetura de uma rede neural artificial, gerando um algoritmo híbrido.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IMPLEMENTATION OF A GENETIC ALGORITHM USING THE MODEL OF ISLAND

Angelo José Moreira Silva

August/2005

Advisors: Nelson Francisco Favilla Ebecken
Myrian Christina de Aragão Costa

Department: Civil Engineering

This dissertation presents the parallel genetic algorithm implementation using the island model. In the island algorithm, random populations are generated of independent manner in each of the islands that remain restricted to specific processors. The optimal solution is obtained with the independent evolution of the islands and the migration of individuals among them, using predefined criterials. Communications among islands is performed using the MPI library. Different logical topologies are currently being analyzed for migration phase among islands, using specifically the ring topology. Some results were generated using examples from the literature and were compared with other implementations. The objective this dissertation is developing a parallel tool for high performance data mining, for example, this tool will be used to optimize the architecture of a artificial neural network, generating a hybrid algorithm.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As pesquisas sobre modelos computacionais inteligentes (Tanomaru 1995) têm, nos últimos anos, se caracterizado pela tendência em buscar inspiração na natureza, onde existe um sem-número de exemplos vivos de processos que podem ser ditos "inteligentes". Algoritmos Genéticos (AG) são as mais difundidas e estudadas técnicas de Computação Evolucionária, pela sua flexibilidade, relativa simplicidade de implementação, e eficácia em realizar busca global em ambientes adversos. Vista de forma global, a evolução natural implementa mecanismos adaptativos de otimização embora estejam longe de serem uma forma de busca aleatória. AGs tentam dirigir a busca para regiões do espaço onde é provável que os pontos ótimos estejam. AGs possuem um paralelismo implícito decorrente da avaliação independente de cada um dos candidatos à solução, ou seja, pode-se avaliar a viabilidade de um conjunto de parâmetros para solução de problemas de otimização.

Os problemas de otimização em sistemas de razoável complexidade não raro exigem grande capaciade de processamento devido à necessidade de muitas avaliações da função objetivo. Estas avaliações podem envolver longas simulações que, para muitos casos práticos, implica na utilização de um ou mais aplicativos de alto custo computacional e complexidade já desenvolvidos e disponíveis.

Apesar de muitos programas de simulação incluírem módulos para otimização, na prática as restrições e o cálculo da função objetivo podem envolver o uso de vários processos.

A crescente disponibilidade de redes de computadores com máquinas cada vez mais poderosas e comunicação cada vez mais rápida vem contribuindo para o aumento de aplicações paralelas em sistemas de memória distribuída. A capacidade de processamento hoje dispo-

nível permite que problemas de grande complexidade sejam resolvidos.

Dentre muitos métodos de otimização hoje disponíveis o uso de algoritmos genéticos é uma opção que oferece grande flexibilidade, pois permite exploração e explotação do espaço de busca aliado ao uso de variáveis contínuas ou discretas, e oferece resultados satisfatórios na maioria dos casos (Cantú-Paz & Goldberg 1999).

Devido à grande demanda computacional associada aos algoritmos genéticos uma estratégia bem aceita para acelerar a convergência é a de um algoritmo genético híbrido, por exemplo, algoritmos genéticos e redes neurais artificiais

Dentre os muitos usos possíveis das redes neurais artificiais pode-se destacar a sua habilidade para aproximar funções complexas. Esta habilidade em aproximar funções sugere o seu uso aliado aos algoritmos genéticos que pode oferecer algum ganho no que diz respeito a aproximar e eventualmente substituir o complexo cálculo da função objetivo e das restrições.

O cenário de um problema de otimização, com o uso de diversos programas para a avaliação das restrições e da função objetivo, com variáveis contínuas e discretas e um tempo de avaliação expressivo para cada alternativa no domínio de busca faz pensar em uma implementação paralela de um AG.

O desenvolvimento de aplicações paralelas é uma tarefa complexa, pois envolve, além de todos os problemas que existem nas aplicações seqüênciais, dificuldades inerentes ao paralelismo tais como criação e inicialização de processos, sincronismo na transmissão de dados entre outros (Ignácio & Filho 2002).

As abordagens convencionais lançam mão de bibliotecas em linguagens compiladas tipo Fortran, C ou C++ a partir das quais o programa de otimização é desenvolvido e para o ambiente paralelo pode-se usar uma biblioteca para troca de mensagens entre os processos.

Normalmente, para fazer um programa para esse tipo de processamento usa-se uma biblioteca específica para passagem de mensagens. Embora nada impeça que o programador use sockets diretamente, não há muito motivo para aumentar a complexidade do programa em função de detalhes de baixo nível. O uso de uma biblioteca permite que o programador só precise se preocupar com instruções como envie uma mensagem e não com detalhes mais específicos de como a mensagem será enviada.

Os dois grandes padrões de bibliotecas de passagem de mensagens são o Parallel Virtual Machine (PVM)(Geist, Beguelin & Dongarra 1994) e o Message Passing Interface (MPI)(Pacheco 1996) (*http://www.mpi-forum.org* 2005) (Gropp, Lusk & Skjellum 1994). O

PVM é o padrão mais antigo e o MPI é a novidade na área, embora ambos existam há anos. Enquanto PVM é realmente o nome de uma biblioteca, MPI é um padrão com várias implementações criadas principalmente por universidades e algumas suportadas comercialmente por empresas.

O PVM se baseia em duas primitivas básicas envie mensagem e receba mensagem, de fácil utilização, mas não tão poderoso quando comparado com o MPI. Este tem opções mais avançadas, como envio de mensagens **broadcast** (para todas as máquinas do cluster) e **multicast** (para um grupo específico de máquinas), assim como melhor controle sobre o tratamento que cada mensagem terá ao ser recebida por outro ponto do cluster (Geist, Kohl & Papadppoulos 1996).

Para o usuário, o que ele vai precisar configurar em um cluster PVM é algo como manter as máquinas o mais idênticas possível, para facilitar a manutenção, e estabelecer alguma forma de relação de confiança entre elas. Usar rhosts e rsh é a forma mais simples de conseguir isso. O usuário roda o gerenciador do PVM, adiciona máquinas ao cluster e depois simplesmente executa o programa feito para PVM. A configuração do MPI depende da implementação, e algumas delas chegam a instalar front-ends para compiladores C e Fortran, mas a forma geral de uso é semelhante.

A biblioteca MPI oferece, através do paradigma de passagem de mensagens, uma grande flexibilidade e portabilidade para programação distribuída. Programas que utilizam esta biblioteca podem rodar em máquinas com múltiplos processadores com memória compartilhada, clusters de computadores com memória distribuída, redes de workstation, grids ou combinações de todos esses, assim o MPI tem se tornado um padrão de comunicação para troca de mensagens e foi adotado nesta dissertação.

1.1 Objetivo

Existem diversas técnicas que detectam padrões em massas de dados, de forma automática, adicionando inteligência à análise dos dados e tornando-a independente do usuário. Uma das principais técnicas utilizadas em aplicações de mineração de dados é conhecida como *redes neurais artificiais*.

As redes neurais são programas computacionais que simulam o trabalho do sistema nervoso, de forma a resolver os problemas de forma eficiente e com muita tolerância aos ruídos

da base de dados. Entretanto existem algumas limitações relacionadas à utilização das redes neurais, principalmente no que se refere à definição da sua configuração ideal e à extração das regras incorporadas no seu modelo.

Uma alternativa para a resolução destas limitações é a utilização de algoritmos de otimização global, conhecidos como *algoritmos genéticos*. Estes algoritmos se baseiam nos mecanismos de seleção natural e genética, sendo adequados para problemas com grande espaço de busca.

Dentre as muitas técnicas escolhidas para o desenvolvimento encontra-se a otimização de rede neural artificial a partir de parâmetros escolhidos por um algoritmo genético paralelo para a obtenção de um melhor resultado em uma quantidade de tempo considerada satisfatória.

O objetivo desta dissertação é o desenvolvimento de uma ferramenta que permite a otimização via algoritmos genéticos, em ambiente paralelo, utilizando-se do modelo de ilhas. Esta abordagem permite a evolução de diferentes populações de forma independente e uma avaliação global para a obtenção do melhor resultado.

1.2 Otimização de Funções e Redes Neurais Artificiais

Nas mais diversas áreas científicas aparecem situações onde é necessária, ou desejável, a solução de problemas de otimização. Uma grande parte dos problemas científicos pode ser formulada como problemas de busca ou otimização. Basicamente, existe uma série de fatores influenciando o desempenho de um dado sistema. Tais fatores podem assumir um número limitado ou ilimitado de valores, e podem estar sujeitos a certas restrições. O objetivo é encontrar a melhor combinação de fatores, ou seja, a combinação de fatores que proporcione o melhor desempenho possível para o sistema em questão. O conjunto de todas as combinações possíveis para os fatores constitui o chamado *espaço de busca*. Não é difícil perceber que existe uma dualidade entre os conceitos de busca e otimização, de tal modo que todo problema de busca pode ser considerado um problema de otimização e vice-versa.

Um problema, por exemplo, de otimização poder ser definido como:

Minimizar	$f(x_1, x_2, \ldots, x_n)$	
dado que	$g_j(x_1, x_2, \ldots, x_n) \geq 0$	$j = 1, 2, \ldots, J$
	$h_k(x_1, x_2, \ldots, x_n) = 0$	$k = 1, 2, \ldots, K$
	$x_i^{inf} \leq x_i \leq x_i^{sup}$	
	$x_i \in X_i$	

Quadro 1 - Exemplo de Otimização

No quadro 1 tem-se que a função f é denominada *função objetivo*, as variáveis x_i são as variáveis de projeto, as desigualdades g_j e as igualdades h_k são denominadas restrições de comportamento e as restrições limitando as variáveis de projeto são as restrições laterais. Um problema de otimização pode não ter restrições.

O conjunto X_i é o domínio da variável de projeto *i* e pode ser qualquer. Também pode configurar restrições laterais para o problema. Normalmente o domínio de uma variável de projeto é um intervalo de número reias ou inteiros ou um conjunto discreto de valores. O produto cartesiano dos domínios das variáveis de projeto define o domínio da função objetivo. Este domínio pode ser então dividido em dois subconjuntos: a região viável onde todas as restrições são satisfeitas e a região inviável onde alguma restrição não é observada. Sempre que existir alguma região viável o problema tem solução.

A solução do problema de otimização, quando existe, leva a um valor ótimo da função associado a um ou mais pontos de ótimo.

Entre os principais métodos para solução de problemas de otimização há os métodos analíticos e os métodos numéricos (de Campos & Saito 2004).

Os métodos analíticos normalmente se baseiam no cálculo diferencial ou no cálculo variacional e permitem, em geral, somente a solução de problemas simples de otimização.

Os métodos numéricos para solução de problemas de otimização são essencialmente classificados em métodos de programação matemática e método probabilísticos. Os métodos de programação matemática são classificados em métodos de programação linear, programação não-linear e métodos baseados em teoria de aproximações como programação linear seqüêncial e programação quadrática seqüêncial. Entre os métodos probabilísticos temos os algoritmos genéticos, a programação evolucionária, as estratégias evolucionárias, o **Simulated Annealing** e a estratégia de colônias entre outros. A diferença essencial dos métodos de programação matemática para os métodos probabilísticos é que os últimos procuram encontrar

o ótimo global do problema de otimização tentando evitar os ótimos locais. Já os métodos de programação matemática podem fornecer um mínimo local. Os métodos probabilísticos, como o próprio nome sugere, utilizam um processo de busca randômica guiados por decisões probabilísticas para obter o ótimo global. Além disso, os métodos probabilísticos são também ferramentas poderosas para problemas com variáveis discretas.

1.3 Apresentação da Dissertação

A tese está organizada como se segue.

Este capítulo descreve alguns aspectos importantes sobre problemas de otimização de funções e redes neurais, além das soluções mais utilizadas.

O capítulo 2 contém uma revisão sobre algoritmos genéticos (codificação, inicialização, seleção e operadores genéticos). São descritos alguns dos processos de seleção mais encontrados na literatura e alguns tipos de operadores de crossover e mutação. Também são descritos os principais parâmetros de um algoritmo genético.

O capítulo 3 contém uma introdução sobre as características e tipos de paralelismo e um resumo sobre a biblioteca MPI. É apresentado uma classificação das técnicas de paralelização de algoritmos genéticos e uma descrição do modelo de ilhas e do operador de migração. Também é descrito o Algoritmo Genético Paralelo Epidêmico e são apresentadas algumas estratégias de migração em função deste.

O capítulo 4 mostra alguns trabalhados correlatos que foram pesquisados e descreve o ambiente experimental em que foram desenvolvidos os algoritmos. É mostrado a implementação de algoritmos genéticos seguindo umas das estratégias de migração descritas no capítulo 3. É explicado a estrutura geral do algoritmo com o operador de migração. São analisados e comparados os resultados obtidos com resultados conhecidos na literatura para otimização de funções.

O capítulo 5 contém uma introdução sobre redes neurais e os problemas encontradados para o seu treinamento, sendo apresentado uma solução utilizando algoritmos genéticos paralelos. Também é mostrado a análise do caso culminando com a otimização da quantidade de neurônios na camada intermediária de uma rede neural.

O capítulo 6 as conclusões do trabalho são expostas e futuros trabalhos são sugeridos.

Capítulo 2

Algoritmos Genéticos

2.1 Introdução

Algoritmos genéticos são algoritmos de otimização global, baseados nos mecanismos de seleção natural e da genética, que exploram informações históricas para encontrar pontos onde são esperados os melhores desempenhos. Isto é feito através de processos iterativos, onde cada iteração é chamada de geração.

Durante cada iteração, os princípios de seleção e reprodução são aplicados a uma população de candidatos. Através da seleção, se determina quais indivíduos conseguirão se reproduzir, gerando um número determinado de descendentes para a próxima geração, com uma probabilidade determinada pelo seu *índice de aptidão*. Em outras palavras, os indivíduos com maior adaptação relativa têm maiores chances de se reproduzir.

Nos algoritmos genéticos, uma população de possíveis soluções para o problema em questão evolui de acordo com operadores genéticos (probabilísticos) concebidos a partir de metáforas biológicas, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua.

Embora o algoritmo genético use um método heurístico e probabilistico para obter os novos elementos, ele não pode ser considerado uma simples busca aleatória, uma vez que explora inteligentemente as informações disponíveis de forma a buscar novos indivíduos ou soluções capazes de melhorar ainda mais um critério de desempenho.

Os algoritmos genéticos procuram privilegiar indivíduos com melhores aptidões, com isto tentam dirigir a busca para regiões do *espaço de busca* onde é provável que os pontos

ótimos estejam.

O fluxo geral de um algoritmo genético é ilustrado na figura 2.1.

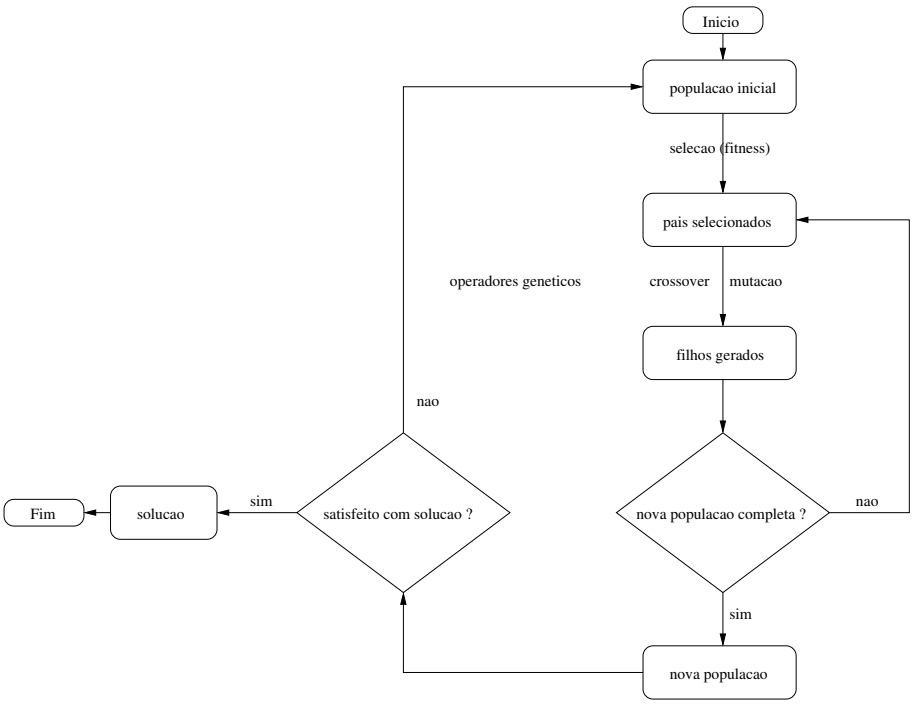

FIGURA 2.1: Fluxo Geral de um Algoritmo Genético

2.2 Codificação

O ponto de partida para a aplicação de algoritmos genéticos a um problema qualquer (busca ou otimização) é a representação do problema a ser analisado, de maneira que os algoritmos genéticos possam atuar adequadamente sobre ele. Naturalmente para cada representação deve haver operadores genéticos correspondentes.

Os algoritmos genéticos processam populações de indivíduos ou *cromossomos*. O cromossomo é uma estrutura de dados, geralmente vetores ou cadeia de valores binários, reais ou combinação de ambas, que representa uma possível solução do problema a ser otimizado. Em geral, o cromossomo representa o conjunto de parâmetros da função objetivo cuja resposta será otimizada (como por exemplo maximizada ou minimizada). O conjunto de todas as configurações que o cromossomo pode assumir forma o seu *espaço de busca*. Se o cromossomo representa n parâmetros de uma função, então o espaço de busca é um espaço com n dimensões. A maioria das representações são genotípicas. O *genótipo* é o conjunto de genes que define a constituição genética de um indivíduo e sobre estes genes é que serão

aplicados os operadores genéticos. Essas representações utilizam vetores de tamanho finito.

Tradicionalmente, o genótipo de um indivíduo é representado por um vetor binário, onde cada elemento de um vetor denota a ausência ou presença de uma determinada característica relevante para a construção de um indivíduo único. Os elementos podem ser combinados formando as características reais do indivíduo, ou seja o seu fenótipo. Teoricamente, essa representação é independente do problema, pois uma vez encontrada a representação em vetores binários, as operações padrões podem ser utilizadas, facilitando o seu emprego em diferentes classes de problemas (Spears, Jong, Back, Fogel & Garis 1993) (Corrêa 2000).

A representação binária é historicamente importante (Jong 1975) (Goldberg 1989), uma vez que foi utilizada nos trabalhos pioneiros de Holland (1962). Além disso, ela ainda é a representação mais utilizada, por ser de fácil utilização e manipulação, e simples de analisar teoricamente. Contudo, se um problema tem parâmetros contínuos e o usuário desejar trabalhar com maior precisão, provavelmente acabará utilizando longos cromossomos para representar soluções, necessitando de uma grande quantidade de memória. Outro aspecto a ser observado é a não-uniformidade dos operadores; por exemplo, se o valor real de um gene for codificado por um vetor binário, a mutação nos primeiros valores binários do gene afetará mais a aptidão do cromossomo que a mutação nos seus últimos valores.

A representação do cromossomo usando valores reais é mais naturalmente compreendida (Wright 1990) (Michalewicz 1992) pelo ser humano e requer menos memória que aquela usando uma cadeia de bits.

A representação em níveis de abstração mais altos tem sido investigada (de Oliveira 2001) e por serem mais *fenótipicas*, facilitariam seu uso em determinados ambientes. Nesse caso, precisariam ser criados os operadores específicos para utilizar essas representações.

A grande maioria dos algoritmos genéticos propostos na literatura usam uma população de número fixo de indivíduos, com cromossomos também de tamanho constante.

Tendo definido a reprentação cromossômica para o problema, gera-se um conjunto de possíveis soluções, chamadas de *soluções candidatas*. Um conjunto de soluções codificadas de acordo com a representação selecionada corresponde a uma população de indivíduos, que representa, ao longo dos ciclos de evolução, o estágio atual da solução do problema. Algoritmos genéticos são algoritmos iterativos, e a cada iteração a população é modificada. Cada iteração de um algoritmo genético é denominada uma geração, embora nem todos os indivíduos de uma população sejam necessariamente *filhos* de indivíduos da população da

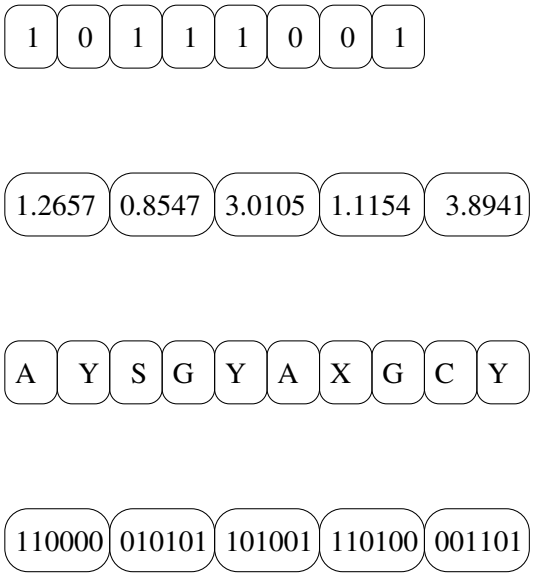

FIGURA 2.2: Representações de Cromossomos

iteração anterior.

Na população são calculados diversos valores estatísticos que servem de medida para avaliar se a busca está próxima do fim. Normalmente os parâmetros avaliados são o melhor indivíduo, a média dos objetivos atingidos, o desvio padrão e também a *diversidade*.

2.3 Inicialização

Como a representação do espaço de busca deve ser a mais sensível possível, a inicialização deste requer algumas ponderações que se representam por meio dos seguintes tipos de inicialização:

- **Inicialização Aleatória :** os indivíduos da população são gerados de forma aleatória
- **Inicialização Determinística :** os indivíduos da população são gerados segundo uma determinada heurística
- **Inicialização Aleatória com Nicho :** os indivíduos da população são gerados de forma que possam ser divididos em espécies, isto é, indivíduos com características semelhantes

Na maior parte das aplicações, a população inicial de n indivíduos é gerada aleatóriamente ou através de algum processo heurístico.

Como no caso biológico, não há evolução sem variedade. Ou seja, a teoria da seleção natural ou *lei do mais forte* necessita de que indivíduos tenham diferentes graus de adaptação

ao ambiente em que vivem. De acordo, é importante que a população inicial cubra a maior área possível do espaço de busca.

A população inicial pode ser obtida através da geração de indivíduos, obedecendo condições de contorno previamente estabelecidas pelo usuário. O usuário estabelece estas condições, tendo em vista o seu conhecimento prévio do problema a ser otimizado. Quanto mais restritas forem as condições de contorno, mais rápida será a convergência, isso porque os valores gerados aleatoriamente estarão mais próximos da solução desejada (Lacerda & Carvalho 1999).

O número de indivíduos que comporá a população, ainda é motivo de estudos, mas existem várias heurísticas, ou seja, depende muito da experiência do usuário e do seu conhecimento prévio sobre a função a ser otimizada (Soares 1997). É claro que, quanto maior o número de elementos na população, maior é a probabilidade de convergência, tendo em vista que a probabilidade da solução desejada ser constadada entre os elementos da população aumenta. Em contrapartida, o tempo de processamento também aumenta. Já no caso da população inicial ser muito pequena, ela terá o problema da perda de *diversidade* (R.K.Ursem 2002), isto é, o espaço de busca seria muito pequeno para ser avaliado. Desta forma, a solução obtida poderia não estar dentro do ótimo global. Conseqüentemente, a convergência seria prematura.

A população inicial não precisa, necessariamente, ser gerada aleatóriamente, tendo em vista que o objetivo é gerar uma população dentro de certo intervalo onde se acredita estar a resposta. Pode-se obter a população inicial também através de um escalonamento do número de indivíduos, pelo intervalo especificado, isto é: se a população é de 50 indivíduos e o intervalo inicial é de 0 a 10, os indivíduos da população inicial deverão estar distribuídos uniformemente neste intervalo.

2.4 Avaliação

Algoritmos genéticos necessitam de informação do valor de uma função objetivo para cada membro da população, que deve ser um valor não-negativo. Nos casos mais simples, usa-se justamente o valor da função que se quer otimizar. A função objetivo dá, para cada indivíduo, uma medida de quão bem adaptado ao ambiente ele está, ou seja, suas chances de sobreviver no ambiente e reproduzir-se, passando parte do seu material genético a gerações posteriores.

A avaliação de cada indivíduo resulta num valor denominado *aptidão* (**fitness**).

A validação é o processo de expor cada elemento da população a função objetivo e, ao final, ordená-los de acordo com a aptidão desta função.

Na convergência, analisa-se o desempenho da população em relação ao objetivo. Isto pode ser feito através de vários fatores, tais como: valores máximo, mínimo e médio da função aptidão. Também, é relativamente comum utilizar-se o desvio padrão dos valores da função de aptidão, como forma de análise da convergência (Goldberg 1989).

Como o algoritmo genético é regido por população, se na população inicial existir um elemento que seja a resposta exata do problema, o algoritmo ainda assim não finalizará o processo de busca da solução, pois isto é um tipo de avaliação. A finalização ou convergência só ocorrerá quando a aptidão média da população estiver suficientemente estável, ou seja, quando houver pouca variação da aptidão média da população atual em relação a anterior. Isto indica que a população se adaptou ao meio, isto é, os elementos da população levam a função ao valor otimizado (desejado) (Lacerda & Carvalho 1999).

Utiliza-se memorizar o indivíduo mais apto, independentemente deste fazer, ou não, parte da população atual. Assim, ao final, este será o resultado esperado (Tanomaru 1995).

Contudo na utilização de algoritmos genéticos pode ocorrer uma rápida convergência para uma solução sub-ótima, porém não o esperado ótimo global. Este problema é denominado *convergência prematura*, podendo ocorrer devido a população reduzida ou a má distribuição da população inicial, em torno do ponto sub-ótimo. Ou seja, um indivíduo próximo de um ótimo local, possui um valor de aptidão superior aos demais indivíduos da população.

Conseqüentemente, o processo de seleção fará com que este indivíduo tenha grande chance de dominar a próxima geração e, assim sucessivamente, se não aparecerem outros indivíduos com melhores valores de aptidão (Tanomaru 1995).

Conforme pode ser visto, a convergência prematura pode ocorrer devido a uma má distribuição dos indivíduos no espaço de busca. Esta má distribuição, também recebe a denominação de *perda de diversidade* (Tanomaru 1995)(Goldberg 1989). Segundo Júlio Tanomaru (Tanomaru 1995), o conceito de diversidade indica o grau em que as diversas regiões estão representadas no epaço de busca. Este problema pode ser amenizado através da escolha criteriosa do número de indivíduos na população, melhora da distribuição dos indivíduos da população inicial no espaço de busca e, também, impedindo a perda de diversidade nas primeiras gerações.

2.5 Seleção

O princípio básico do funcionamento dos algoritmos genéticos é que um critério de seleção vai fazer com que, depois de muitas gerações, o conjunto inicial de indivíduos gere indivíduos mais aptos.

O mecanismo de seleção em algoritmos genéticos emula os processos de reprodução assexuada e seleção natural.

O algoritmo genético começa com uma população inicial de *n* indivíduos. Quando não existe nenhum conhecimento prévio sobre a região do espaço de busca onde se encontra a solução do problema, os indivíduos são gerados aleatoriamente. Se houver um conhecimento *a priori* sobre a região em que está localizada a solução, ou seja, forem conhecidas soluções aceitáveis que podem estar próximas à solução ótima, os indivíduos iniciais podem ser definidos de forma determinística.

Para que o processo de seleção privilegie os indivíduos mais aptos, a cada indivíduo da população é atribuído um valor dado por uma função f denominada *função de aptidão*. Esta função recebe como entrada os valores do gene do cromossomo (*indivíduo*) e fornece como resultado a sua aptidão. A aptidão pode ser vista como uma nota na qual se avalia a solução codificada por um indivíduo. Esta aptidão é baseada no valor da função objetivo, que é específica para cada problema.

Para alguns métodos de seleção, é desejável que o valor de aptidão de cada indivíduo seja menor que 1 e que a soma de todos os valores de aptidão de cada indivíduo seja igual a 1. Para isso, para cada indivíduo é calculada a *aptidão relativa* (f_{rel}). A aptidão relativa para um dado indivíduo é obtida dividindo o valor de sua aptidão pela soma dos valores de aptidão de todos os indivíduos da população.

Em geral, gera-se uma população inicial de *n* indivíduos extraídos com probabilidade proporcional à aptidão relativa de cada indivíduo na população, ou seja, a probabilidade de seleção de um cromossomo ou indivíduo *x* é dada por

$$f(x_i)_{rel} = \frac{f(x_i)}{\sum_{j=1}^{n} f(x_j)} \tag{2.1}$$

onde $f(x_i)$ é a função de aptidão.

Usando a probabilidade acima, seleciona-se n indivíduos. Neste processo, indivíduos com baixa aptidão terão alta probabilidade de desaparecerem da população, ou seja, serem extintos, ao passo que indivíduos com alta aptidão terão grande chance de sobreviverem.

Uma função objetivo (ou função de avaliação) é geralmente uma expressão matemática que mede o quanto uma solução está próxima ou distante da solução desejada (satisfaz o objetivo do problema). Muitas vezes ela inclui restrições que devem ser satisfeitas pela solução. Alguns problemas de otimização procuram maximizar o valor da função objetivo, isto é, encontrar soluções que produzam o maior valor possível para a função objetivo; por exemplo definir o número máximo de caixas que podem ser colocadas dentro de um depósito. Outros problemas procuram minimizar o valor da função objetivo; por exemplo, encontrar a solução mais barata. Existem ainda funções que procuram satisfazer mais de um objetivo. Essas funções são encontradas em problemas de otimização multiobjetivo.

Associada uma nota ou aptidão a cada indivíduo da população, o processo de seleção escolhe então um subconjunto de indivíduos da população atual, gerando uma população intermediária. Vários métodos de seleção têm sido propostos. A maioria deles procura favorecer indivíduos com maiores valores de aptidão, embora não exclusivamente, a fim de manter a diversidade da população. Alguns métodos de seleção são:

- roleta
- torneio
- amostragem estocástica
- classificação

2.5.1 Roleta

O método da roleta é o método de seleção mais simples e também o mais utilizado. Os indivíduos de uma geração são selecionados para a próxima geração utilizando uma roleta, semelhante à roleta utilizada em jogos de azar.

Neste método, cada indivíduo da população é representado na roleta conforme seu valor de aptidão. Desta forma, os indivíduos com elevada aptidão receberão um intervalo maior na roleta, enquanto aqueles que tem mais baixa aptidão receberão menor intervalo na roleta. Após a distribuição na roleta, são gerados aleatóriamente valores no intervalo entre 0 e o

total do somatório da aptidão de todos os indivíduos da população. É gerado um determinado número de vezes, dependendo do tamanho da população. O indivíduo que possuir em seu intervalo o valor gerado, será selecionado. Os indivíduos selecionados são inseridos na população intermediária.

Na figura 2.3 é mostrado o esquema da roleta. Na tabela 2.1 é mostrado um exemplo. Na primeira coluna estão os indivíduos da população, na segunda está o valor de aptidão correspondente a cada indivíduo e na terceira coluna está a percentagem correspondente a cada um deles.

Indivíduo	**Aptidão**	**Roleta**	**Aptidão Relativa**	**Porcentagem**
I_1	22	22	0.27160493	27%
I_2	25	47	0.30864197	31%
I_3	17	64	0.20987654	21%
I_4	7	71	0.08641975	9%
I_5	10	81	0.12345679	12%

TABELA 2.1: Exemplo de seleção pelo método da roleta

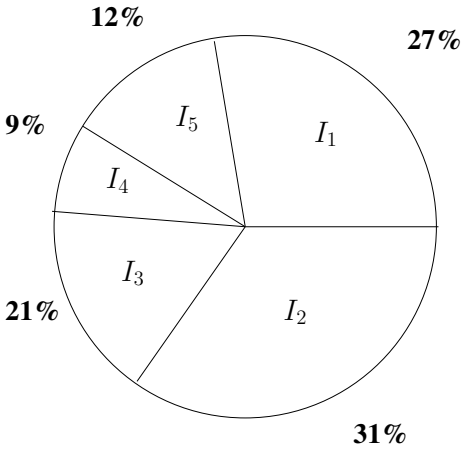

FIGURA 2.3: Esquema da Roleta

De forma simplificada, o método da roleta é realizado em três passos:

1. [Soma] Calcular a soma dos valores da aptidão de todos os indivíduos - soma **S**.

2. [Seleção] Gerar um número aleatório dentro do intervalo $(0, S)$ - **r**.

3. [Loop] Começar a somar os valores de aptidão dos indivíduos até atingir ou ultrapassar o valor **r**. Retornar o último indivíduo utilizado na soma.

2.5.2 Torneio

Neste método n indivíduos da população são escolhidos aleatoriamente, com a mesma probabilidade. O indivíduo com maior aptidão dentre estes n indivíduos é selecionado para a população intermediária. O processo se repete até que a população intermediária seja preenchida. Geralmente, o valor utilizado para n é 3. Na figura 2.4 é apresentada a utilização de seleção por torneio para $n = 3$.

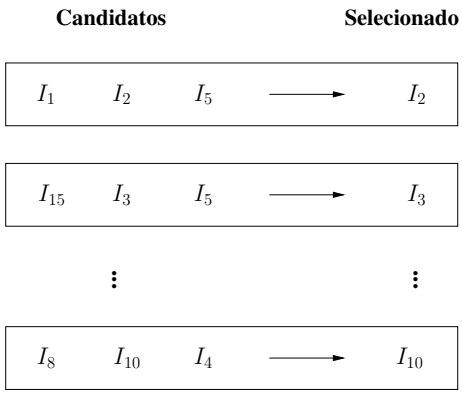

FIGURA 2.4: Esquema do Torneio

2.5.3 Amostragem Estocástica

Este método é uma variação do Método da Roleta em que, em vez de uma única agulha, n agulhas igualmente espaçadas são utilizadas, onde n é o número de indivíduos a serem selecionados. Assim, em vez de n vezes, a roleta é girada uma única vez.

2.5.4 Classificação

Este método primeiramente classifica a população, depois cada indivíduo recebe um valor de acordo com esta classificação. O pior terá valor 1, o segundo pior terá valor 2 e assim sucessivamente. O melhor terá valor n igual ao número de indivíduos da população. Após a classificação todos os indivíduos têm chances de serem selecionados.

2.6 Operadores Genéticos

Um algoritmo de otimização global deve ser capaz de explorar pontos inteiramente novos dentro do espaço de busca, bem como intensificar a busca em determinadas regiões consideradas promissoras. Esse mecanismo de diversificação e intensificação (**exploration** e **explotation**) é obtido nos algorimos genéticos pela correta aplicação dos operadores genéticos.

Goldberg formalizou as principais características dos operadores clássicos de *cruzamento* (**crossover**), *mutação* e *inversão*, principalmente no que tange a codificação binária (Goldberg 1989).

Há consenso, pelo menos, que o cruzamento utiliza a informação contida em dois ou mais indivíduos (pais), para gerar um ou mais novos indivíduos (filhos). Esse processo tende a não acrescentar novas informações à população, por explorar apenas a região próxima aos indivíduos pais.

A mutação, por sua vez, pode ser entendida tanto como um *diversificador* ou como um *intensificador* de busca. Em algumas abordagens, como estratégias de evolução, a mutação é a única responsável pela evolução e o que determina se o movimento é de exploração ou intensificação são parâmetros adaptáveis ao longo das gerações (de Oliveira 2001) (Tanomaru 1995).

A mutação diversifica quando introduz uma informação inteiramente nova no indivíduo e, conseqüentemente à população. Por outro lado, quando apenas aplica um ruído à solução contida no indivíduo, a mutação é um mecanismo *intensificador* de busca na vizinhança dessa solução.

A seguir são apresentados alguns operadores de cruzamento e mutação encontrados na literatura e uma divisão categórica dos mesmos encontrada em (de Oliveira 2001)

2.6.1 Cruzamento

O processo de recombinação é um processo sexuado - ou seja, envolve mais de um inivíduo - que emula o fenômeno de **crossover**, a troca de fragmentos entre pares de cromossomos. Na forma mais simples, trata-se de um processo aleatório que ocorre com probabilidade fixa p_{rec} que deve ser especificada pelo usuário.

1. Convencionais
    (a) Uniforme
    (b) N pontos
2. Aritméticos
    (a) Média aritmética
    (b) Média geométrica
    (c) Michalewicz
    (d) Blend - α
    (e) Unimodal normal distribution
    (f) Simulated binary
    (g) Simplex
3. Heurísticos (ou direcionais)

2.6.1.1 Blend - α

No **blend crossover**, que também é chamado de BLX-α são gerados dois filhos a partir de dois pais. Inicialmente é gerado um número β aleatório com distribuição uniforme no intervalo $[-\alpha\,,\,1+\alpha]$. Os filhos são gerados a partir das equações abaixo. O valor de α usual é 0.5. Este tipo de cruzamento foi utilizado na dissertação.

$$gene_i^{filho1} = gene_i^{pai1} + \beta * (gene_i^{pai2} - gene_i^{pai1}) \tag{2.2}$$

$$gene_i^{filho2} = gene_i^{pai2} + \beta * (gene_i^{pai1} - gene_i^{pai2}) \tag{2.3}$$

2.6.2 Mutação

O processo de mutação em algoritmos genéticos é equivalente à busca aleatória. Basicamente, seleciona-se uma posição num cromossomo e muda-se o valor do gene correspondente aleatoriamente para um outro possível. O processo é geralmente controlado por um parâmetro fixo p_{mut} que indica a probabilidade de um gene sofrer mutação.

1. Binária
2. Aleatória (ou uniforme)
3. Não-uniforme (Michalewicz)
4. Gaussiana
5. Modal discreta
6. Modal contínua
7. Mühlenbein
8. Creep

2.6.2.1 Creep

A mutação **Creep** (*escorregamento*) consiste em somar ao valor de um gene um valor gerado a partir de uma distribuição normal com média zero e desvio padrão baixo. Uma alternativa é multiplicar-se o valor do gene por um número aleatório próximo a 1.

Este operador é aplicado no caso da representação real e não gera grande pertubação nas populações. Isto permite que seja usado com taxas de mutação mais elevadas.

O efeito da mutação creep auxilia na busca local (*explotação*) pois parte da idéia de que se um cromossomo está perto de uma valor ótimo uma pequena alteração pode levá-lo ao ótimo (de Mendonça 2004).

Este tipo de mutação foi utilizado na dissertação.

2.7 Parâmetros Genéticos

O desempenho de um Algoritmo Genético é fortemente influenciado pela definição dos parâmetros a serem utilizados, portanto é importante , analisar de que maneira alguns parâmetros influem no comportamento dos algoritmos genéticos, para que se possa estabelecê-los conforme as necessidades do problema e dos recursos disponíveis (Cantú-Paz & Goldberg 1999) (Tanomaru 1995).

2.7.1 Tamanho da População

O tamanho da população afeta o desempenho global e a eficiência dos algoritmos genéticos. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.

2.7.2 Taxa de Cruzamento

Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidamente que a capacidade da seleção em criar melhores estruturas. Com um valor baixo, o algoritmo pode se tornar lento ou estagnar.

2.7.3 Taxa de Mutação

Uma baixa taxa de mutação previne que a busca fique estagnada em regiões do espaço de busca. Além disso, possibilita que qualquer ponto do espaço de busca seja atingido. Com uma taxa muito alta a busca se torna essencialmente aleatória.

2.7.4 Taxa de Substituição

Controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

2.7.5 Condição de Parada

Como se está tratando de problemas de otimização o ideal seria que o algoritmo terminasse assim que o ponto ótimo fosse descoberto. Já no caso de funções multimodais, um ponto

ótimo poder ser o suficiente, mas pode haver situações onde todos ou o maior número possível de pontos ótimos sejam desejados. Um problema prático é que, na maioria dos casos de interesse, não se pode afirmar com certeza se um dado ponto ótimo corresponde a um ótimo global. Como conseqüência, normalmente usa-se o critério do *número máximo de gerações* ou um tempo limite de processamento para parar um algoritmo genético. Outro critério plausível é parar o algoritmo usando a idéia de *estagnação*, ou seja, quando não se observa melhoria da população depois de várias *gerações consecutivas*, isto é, quando a aptidão média ou do melhor indivíduo não melhora mais ou quando as aptidões dos indivíduos de uma população se tornarem muito parecidas. Ao conhecer a resposta máxima da função objetivo, é possível utilizar este valor como critério de parada.

Capítulo 3

Algoritmos Genéticos Paralelos

3.1 Introdução sobre Paralelismo

Paralelismo é uma estratégia utilizada em computação para obter-se, mais rapidamente, resultados de tarefas grandes e complexas. Segundo esta estratégia, uma tarefa grande pode ser dividida em várias tarefas pequenas, que serão distribuídas entre vários processadores e executadas simultaneamente. Os processadores se comunicam entre si para que haja sincronização entre as tarefas em execução.

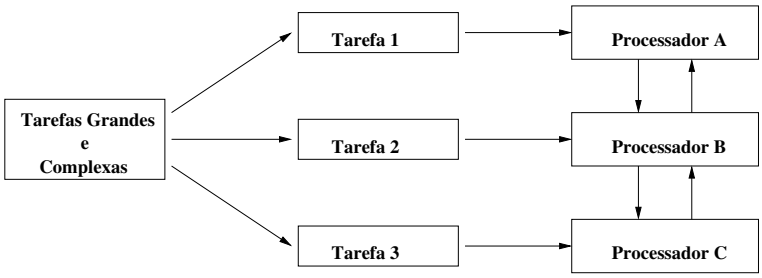

FIGURA 3.1: Comunicação entre processadores

Os principais objetivos do paralelismo são:

- Aumentar o desempenho (reduzindo o tempo) no processamento
- Resolver grandes desafios computacionais
- Fazer uso de um sistema distribuído para resolução de tarefas
- Obter ganhos de performance

Existem complexidades pertinentes ao próprio paralelismo. Os resultados (desempenho) podem não corresponder ao esforço (programação) empregados. O programador é diretamente responsável (em ferramentas não automáticas) pelo paralelismo.

3.1.1 Tipos de paralelismo

Dentre as várias formas de classificar o paralelismo, levamos em consideração o objeto paralelizado, como segue abaixo:

3.1.1.1 Paralelismo de Dados

O processador executa as mesmas instruções sobre dados diferentes. É aplicado, por exemplo, em programas que utilizam matrizes imensas e para cálculos de elementos finitos.

Exemplos:

- Resolução de sistemas de equações
- Multiplicação de matrizes
- Integração numérica

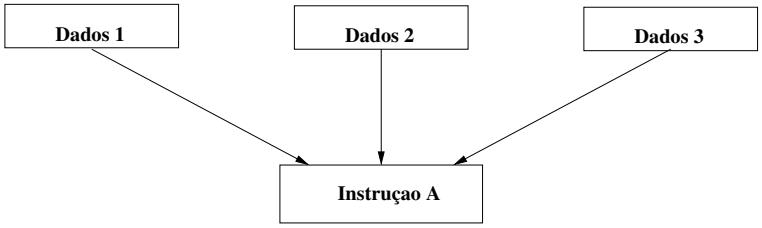

FIGURA 3.2: Processador executando as mesmas instruções sobre dados diferentes

3.1.1.2 Paralelismo Funcional

O processador executa instruções diferentes que podem ou não operar sobre o mesmo conjunto de dados. É aplicado em programas dinâmicos e modulados onde cada tarefa será um programa diferente.

Exemplos

- Paradigma Produtor-Consumidor
- Simulação

- Rotinas específicas para tratamento de dados (imagens)

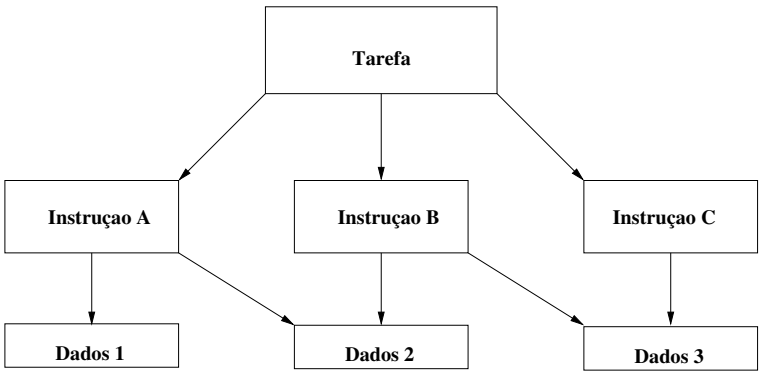

FIGURA 3.3: Comunicação entre processadores

3.1.2 Ambiente Paralelo

Um ambiente paralelo possui vários processadores interligados em rede sobre uma plataforma para manipulação de processos paralelos. A plataforma deve possuir um sistema operacional que esteja habilitado a executar este tipo de processamento além de uma linguagem de programação que atenda as exigências de um modelo de programação paralela.

Tipos de ambientes:

- **Message Passing** : é o método de comunicação baseada no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. O programador é responsável pela sincronização das tarefas. Exemplos :
  - PVM - Parallel Virtual Machine (Geist et al. 1994)
  - MPI - Message Passing Interface (Ignácio & Filho 2002)
  - MPL - Message Passing Library (Soch & Tvrdik 1998)
- **Data Parallel** : é a técnica de paralelismo de dados, normalmente automática ou semi-automática, ou seja, é o método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação. Exemplos :
  - HPF - High Performance Fortran (Chapman & Mehrotra 1998) (Sarma, Zacharia & Miles 1998)

3.1.3 Obstáculos no paralelismo

As tarefas executadas em paralelo, num determinado instante, aguardam a finalização mútua para coordenar os resultados ou trocar dados e reiniciar novas tarefas em paralelo. É necessário que haja a coordenação dos processos e da comunicação entre eles para evitar que a comunicação seja maior que o processamento e que consequentemente haja uma queda no desempenho dos processos em execução.

Existem poucos compiladores e ferramentas prontas para paralelização automática que resolvam definitivamente o problema do paralelismo.

Há um considerável tempo gasto do programador em analisar o código fonte para paralelizar e recodificar. Basicamente é necessário rever o programa sequencial, avaliar como será particionado, quais os pontos de sincronização, quais as partições que poderão ser executadas em paralelo e qual a forma de comunicação que será empregada entre as tarefas paralelas (Grama, Gupta, Karypis & Kumar 2003).

É necessário que haja uma análise do algoritmo a ser paralelizado para que seja possível a paralelização nos dados ou nas funções do mesmo, levando sempre em consideração a quantidade de comunicação em relação ao processamento (Jordan & Alaghband 2003). No processamento paralelo, o tempo de CPU quase sempre aumenta, no entanto pode-se reduzir o tempo total de processamento.

A programação paralela utiliza multiprocessadores com arquiteturas de comunicação entre os processadores baseados em *memória compartilhada* (**shared memory**) e *memória distribuída* (**distributed memory**). Os programas adaptados a arquitetura de comunicação entre os processadores baseado em memória compartilhada não podem ser executados em uma máquina com memória distribuída, pois não utilizam os mecanismos de troca de mensagens, impossibilitando assim a portabilidade. No caso inverso, memória distribuída para memória compartilhada, seria possível a portabilidade, porém o mecanismo de troca de mensagem seria ignorado, pois todos os dados já poderiam ser acessados pelos processadores através da memória compartilhada, isto apenas traria um fluxo desnecessário de comunicação.

Os processos são distribuídos e executados em vários processadores simultaneamente, entretanto não existe uma forma eficiente de acompanhar passo-a-passo a alteração das variáveis durante a execução do processamento das diversas tarefas paralelas.

3.2 Message Passing Interface

O MPI é uma biblioteca com funções para troca de mensagens, responsável pela comunicação e sincronização de processos. Dessa forma, os processos de um programa paralelo podem ser escritos em uma linguagem de programação seqüencial, tal como C ou Fortran.

O principal objetivo do MPI é disponibilizar uma interface que seja utilizada no desenvolvimento de programas que utilizem troca de mensagens. Além de garantir a portabilidade dos programas paralelos, essa interface deve ser implementada eficientemente nos diversos tipos de máquinas paralelas existentes.

O MPI (Message-Passing Interface) é uma padronização do paradigma da troca de mensagens, que foi sugerida por um grupo de trabalho formado por representantes da indústria, órgãos governamentais e universidades. O trabalho se iniciou em 1992 e em novembro de 1993 o primeiro *draft* do padrão MPI-1 foi apresentado. Em 1995 foi disponibilizado o MPI-1.1 com correções e algumas extensões ao padrão MPI-1 com aproximadamente 150 funções. Atualmente o padrão é o MPI-2 que inclui uma série de funcionalidades novas como suporte a processos dinâmicos e aproximadamente 250 funções (Ignácio & Filho 2002).

As vantagens principais de estabelecer um padrão para troca de mensagens entre processos são a portabilidade e a facilidade de utilização. O padrão também permite que os fabricantes de hardware e software trabalhem sobre uma base clara possibilitando que as implementações do padrão nas diversas plataformas sejam mais eficientes para o uso em máquinas paralelas de memória compartilhada ou distribuída.

Mais detalhes sobre a sintaxe das chamadas de rotina e demais detalhes da biblioteca podem ser encontradas em (Pacheco 1996) (*http://www.mpi-forum.org* 2005) (Gropp et al. 1994)

3.3 Paralelismo nos Algoritmos Genéticos

Os algoritmos genéticos são habitualmente considerados algoritmos implicitamente paralelos, sendo esta característica um dos seus pontos fortes. Atualmente existe uma procura cada vez maior por algoritmos que, além de resolver os problemas de forma aceitável, também os resolvam de um modo que seja paralelizável, pois cada vez mais a tendência está voltada para a utilização de sistemas paralelos. Os algoritmos genéticos possuem uma estrutura com-

putacional altamente paralelizável. Assim, se analisarmos a estrutura de algoritmo genético chegamos às seguintes conclusões:

- Cada indivíduo da população tem uma qualidade Q, que pode ser avaliada independentemente de qualquer outro fator
- Cada operador e operação genética é independente pelo que podem ser aplicado em qualquer ordem, seqüencial ou não, a qualquer elemento da população.

Observando mais uma vez a natureza, chegamos à conclusão que nela todos os processos são paralelos, ou seja, os processos seqüenciais é que não são naturais. Basta imaginarmos que neste momento, em qualquer parte do mundo, existem crianças nascendo, lutas pela sobrevivência, pessoas morrendo, e muitas outras atividades, sem que exista um ponto comum de controle. Podemos interferir diretamente ou não no que se passa, mas não existe ninguém com um conhecimento global do estado atual do mundo e acima de tudo ninguém consegue interferir ou controlar tudo o que acontece.

3.3.1 Modelos de Algoritmos Genéticos Paralelos

Uma das áreas de maior investigação nos algoritmos genéticos tem sido sobre o modo como podem ser efetivamente paralelizados, tendo surgido desse esforço um grande conjunto de possíveis implementações. O que todas estas diferentes implementações têm demonstrado é que o que é importante não é apenas o algoritmo em si, mas sim o conjunto do algoritmo, os problemas para que vai ser utilizado, a sua parametrização e o equipamento que vai servir de base à execução do algoritmo genético.

Embora estes já fossem fatores essenciais nos algoritmos seqüenciais, o que se veio a notar é que surgiram muitos algoritmos que eram bons em determinadas situações, piores em outras e com pequenas variações de parâmetros e/ou nos problemas tudo se poderia inverter. Foram ainda acrescentados mais fatores a um problema que já não era simples. Assim, poderíamos ter um excelente algoritmo para executar num conjunto de transputers, que se fosse executado numa rede de workstations já não seria tão bom, ou vice-versa, mas dependendo também da parametrização e do problema.

A idéia básica atrás da maioria dos programas paralelos é dividir uma tarefa em partes e solucionar as partes simultaneamente usando múltiplos processadores. Esta abordagem de

dividir-e-conquistar pode ser aplicada em algoritmos genéticos de muitos modos diferentes, e a literatura contém muitos exemplos de implementações paralelas (Adamidis 1994) (Gordon & Whitley 1993) (Goodman, Lin & Punch 1994). Alguns métodos de paralelização usam uma única população, enquanto outros dividem a população em várias subpopulações relativamente isoladas. Alguns métodos podem explorar arquitetura de computadores maciçamente paralelos, enquanto outros são melhores adaptados para multicomputadores com menos e elementos de processamento, mas mais potentes (Cantú-Paz 1997).

De acordo com (Cantú-Paz & Goldberg 1999) (Adamidis 1994) (Gordon & Whitley 1993) (Goodman et al. 1994), existem três tipos (modelos) principais de Algoritmos Genéticos Paralelos:

Granularidade Grossa : Neste modelo várias subpopulações isoladas evoluem em paralelo e periodicamente trocam informações através da *migração* dos seus melhores indivíduos para subpopulações vizinhas. Os indivíduos competem somente na respectiva subpopulação.

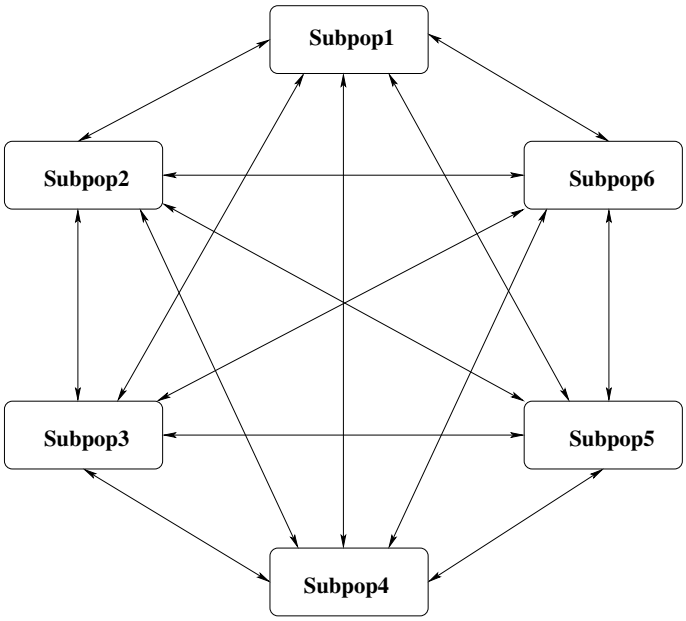

FIGURA 3.4: Esquema de um algoritmo genético paralelo de granularidade grossa. Cada processo é um simples algoritmo genético seqüêncial, e há (não frequentemente) comunicação entre as subpopulações

Granularidade Fina : Também conhecido como *modelo de vizinhança* (**neighborhood model**), onde uma única população evolui e cada indivíduo é colocado em uma célula de uma grade planar. Os processos de *seleção* e *cruzamento* são aplicados somente entre indivíduos vizinhos na grade (de acordo com a topologia definida).

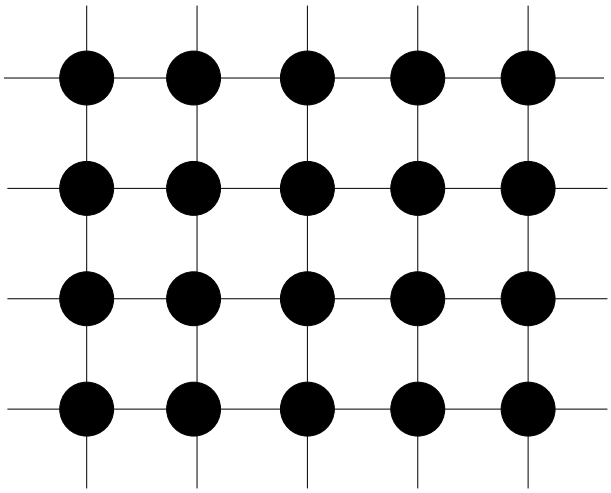

FIGURA 3.5: esquema de um algoritmo genético paralelo de granularidade fina. Esta classe de algoritmo genéticos tem uma população distribuída espacialmente.

Paralelização Global : São essencialmente versões paralelas de algoritmos genéticos seqüênciais. Eles operam sobre uma população global e são normalmente síncronos. Este modelo é adequado à arquiteturas paralelas com memória compartilhada. Nesta classe de algoritmos genéticos paralelos a população permanece global, e os operadores genéticos são aplicados a toda população, em paralelo. Um processador principal (*mestre*) mantém a população do algoritmo genético e envia partes desta população aos processadores secundários (*escravos*) para reprodução, mutação ou avaliação. Para ser eficiente, este método geralmente requer uma rede altamente conectada devido ao grande volume de comunicação necessária à transferência dos indivíduos. Este processo também exige que o processador mestre gerencie toda a população. Para grandes populações, algoritmos genéticos como este podem necessitar de uma grande quantidade de memória, considerada inviável (presumindo-se que toda população será armazenada na memória). A caracterpistica principal desta classe de algoritmo genético é que ela se comporta como um algoritmo genético seqüencial.

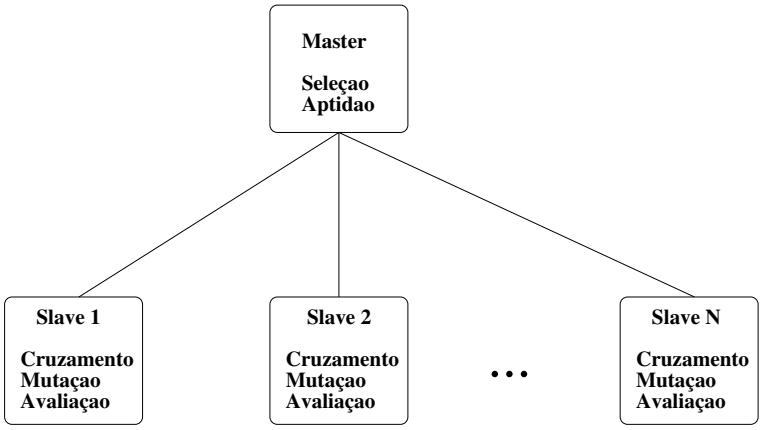

FIGURA 3.6: Esquema de um algoritmo genético paralelo global (**master-slave**). O processo *mestre* armazena a população, executa operações de um algoritmo genético, e distribui indivíduos para os processos *escravos*. Os processos *escravos* somente avaliam a aptidão dos indivíduos.

3.3.1.1 Modelo Ilha

A observação de ambientes naturais isolados, como ilhas, têm demonstrado que algumas das espécies que surgem são especialmente adaptadas às particularidades dos seus ambientes, em contraste com populações equivalentes mas em ambientes mais abertos, em que todos os membros têm acesso a todos os outros membros.

Estas observações levaram ao surgimento de novas arquiteturas e operadores que melhoravam os algoritmos genéticos paralelos para além do ganho puro de performance, que em conjunto com as considerações relativas à possibilidade de várias populações concorrentes poderem gerar uma melhor solução do que a conseguida com apenas uma população, levaram à construção de um modelo que aproveita ambas as vantagens. Aproveita os ganhos de fiabilidade na obtenção de soluções com a melhor qualidade das soluções.

Este modelo é o chamado *Modelo Ilha*, devido à fonte inspiradora do conceito. Assim, o algoritmo seqüencial fica dividido numa série de ilhas (populações), que comunicam entre si as melhores soluções. No resto do tempo, estão trabalhando para melhorar as soluções internas da ilha. Este modelo, entretanto, mantém a estrutura seqüencial do algoritmo genético, apenas sendo adicionados os operadores para fazer a *migração* dos elementos entre as difrentes ilhas.

No modelo ilha existem diversos processos trabalhando sobre subpopulações que evoluem em paralelo. As populações iniciais são geradas de modo idêntico à sua versão seqüencial.

O modelo de paralelização ilha introduz um *operador de migração*, além das três etapas básicas (*seleção*, *cruzamento* e *mutação*). O operador migração é usado para enviar indivíduos de uma subpopulação para outra.

Devido aos elevados custos de comunicação neste tipo de sistema, a freqüência de migração não deve ser muito alta. desta forma, no modelo ilha o operador de migração é executado apenas quando existir a necessidade de uma renovação de uma subpopulação (fase de diversificação de uma população).

O critério de terminação local dos processos é baseado em uma condição global que envolve todos os processos que compõem o algoritmo genético paralelo; impedindo com isso, que um processador fique ocioso enquanto os demais permaneçam executando.

Os processos executados por cada processador e a comunicação entre eles pode ser melhor entendido através da ilustração da figura 3.7, onde cada par de processos m_i e q_i compartilham um mesmo processador, que é comutado entre os dois para a execução.

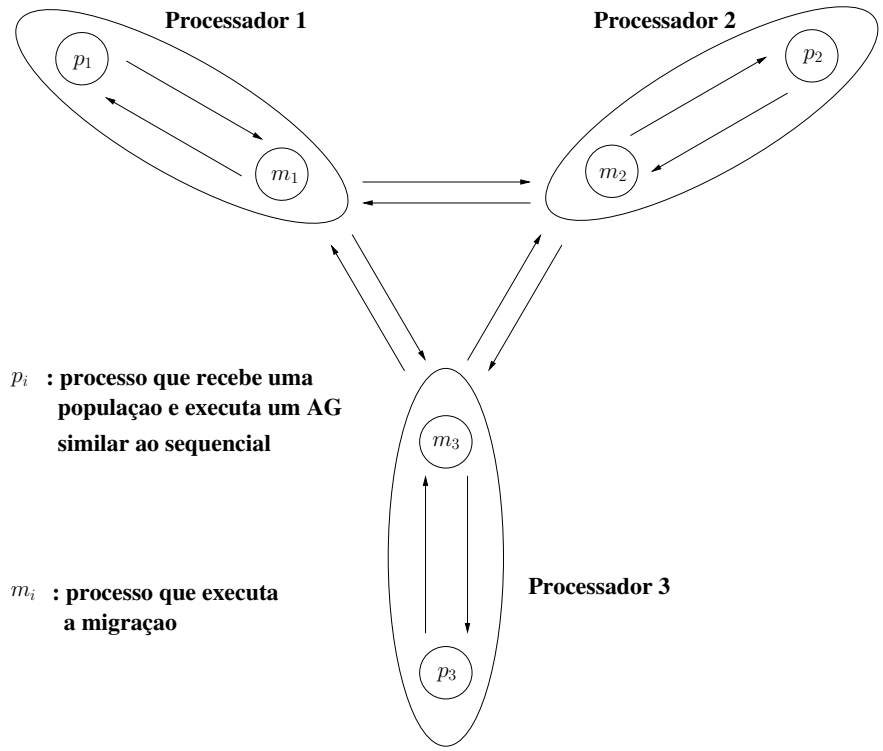

FIGURA 3.7: Comunicação entre processos no modelo ilha para 3 processadores

Inicialmente cada processo m_i recebe uma população e executa um algoritmo genético similar ao algoritmo genético seqüencial. O processo q_i tem a função de realizar a migração e controlar a terminação do algoritmo genético paralelo.

A estratégia adotada no algoritmo consiste em associar a cada processo m_i (módulo

principal), que compõe o algoritmo genético paralelo do modelo ilha, um processo q_i, tal que 1 < i < n (onde n é o número de processadores), se comunicam apenas por troca de mensagens e possuem a característica descrita abaixo.

O processo q_i é ativado pelo processo m_i correspondente nos seguintes casos:

1. Quando, por um período de *w* iterações consecutivas, não existir renovação num percentual mínimo exigido da população, disparando neste caso o processo de migração de indivíduos, onde *w* é um parâmetro de entrada.

2. Quando o processo m_i está habilitado a terminar, dando início ao processo de terminação do algoritmo genético.

No primeiro caso, conforme figura 3.8, q_i envia pedidos a todos os outros processos do tipo q para que os mesmos lhe enviem suas melhores soluções correspondentes. Baseando-se nas soluções recebidas em resposta ao seu pedido, q_i repassa as soluções recebidas ao m_i associado, efetuando-se deste modo a migração de indivíduos.

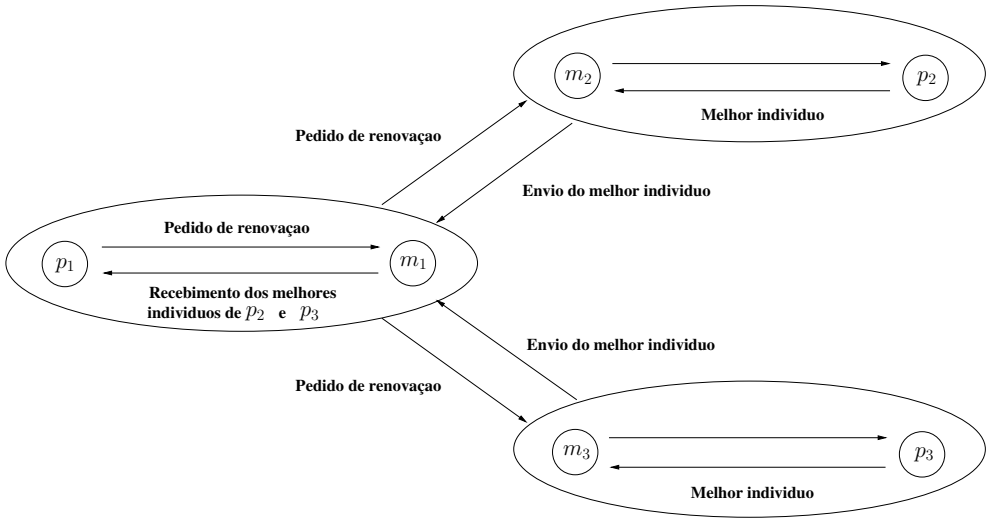

FIGURA 3.8: Migraçao de indivíduos pelo processo m_1 no caso de 3 processadores

No segundo caso, quando o processo m_i está habilitado a terminar, ou seja, depois de realizado um número *k* de diversificações (de acordo com o critério de parada do modelo seqüencial), este ativa o processo q_i que mantém um vetor de *n* elementos representando o estado do sistema (*vetor de estado*). Cada posição *i* do vetor de estado pode assumir o valor verdadeiro ou falso indicando que o processo m_i está habilitado a terminar ou não. Ao ser ativado, o processo q_i atualiza a posição *i* do vetor de estado com verdadeiro e inicia um **broadcast** a fim de que os demais vetores representem o estado atual do sistema. Um processo m_i, ainda que habilitado a terminar, continuará sua execução até que todos os outros

processos também possam terminar. Isto impedirá que um processador fique ocioso enquanto os demais permaneçam ativos, e ainda possibilita a melhora de sua solução ótima enquanto o sistema não termina.

Um processo m_i, a cada geração do algoritmo genético, informa a q_i a sua melhor solução gerada até o momento.

Caso a finalização seja detectada, isto é, todos os elementos do vetor de estado sejam verdadeiros, m_i é informado a fim de que possa terminar sua execução.

A estratégia descrita acima permite que o processo responsável pela execução do processo genético não se ocupe com a comunicação com os demais processos, necessária para controle da migração de indivíduos, e com a finalização do algoritmo genético paralelo, simplificando o seu projeto e implementação.

Muitos pesquisadores em Algoritmos Genéticos acreditam que um Algoritmo Genético Paralelo, com suas múltiplas subpopulações distribuídas e regras locais e iterações, é um modelo mais realístico para a evolução de espécies na natureza que uma única grande população (Cantú-Paz 1998) (Tanomaru 1995) (Cantú-Paz 1997) (Cantú-Paz 1995) (Adamidis 1994) (Levine 1994) (Schwehm 1996).

Em (Sambatti, de Campos Velho, Leonardo D. Chiwiacowsky & Preto 2005) é apresentado um algoritmo genético paralelo chamado *Algoritmo Genético Paralelo Epidêmico* que adota a estrutura de subpopulações (*granularidade grossa*). Neste trabalho foram implementadas algumas estratégias de migração com base na chamada *Estratégia Epidêmica* (Medeiros 2003), que por sua vez utiliza um operador genético chamado *Operador Genético Epidêmico*. Esta estratégia ativa este operador quando um número específico de gerações é alcançada sem melhoria no(s) melhor(es) indivíduo(s). Então todos os indivíduos são afetados por uma a *peste* ou *praga*, e somente aqueles que tem a melhor aptidão (isto é, 2% com a melhor aptidão na população) sobrevivem. Os indivíduos restantes morrem e são substituídos por novos indivíduos com nova variabilidade, tal como imigrantes chegando, a fim de evoluir a população. Dois parâmetros precisam ser escolhidos: um determina quando a estratégia será ativada, isto é, o número de gerações sem melhoria da melhor aptidão individual, enquanto o outro parâmetro determina a quantidade de indivíduos que sobreviverão a *peste* ou *praga*.

No *Algoritmo Genético Paralelo Epidêmico* as estratégias de migração implementadas foram a **island model** e **stepping-stone model**

Island model : indivíduos de melhor aptidão podem migrar para todos os outros processos.

Island-1 : denota o envio da melhor solução de cada processo para um processo *mestre* que seleciona e difunde o melhor dos melhores para todos os outros processos

Island-2 : denota múltiplas difusões no qual cada processo envia sua melhor solução para todos os outros processos.

Stepping-stone model : um anel lógico de processadores é definido e a comunicação ocorre em etapas enquanto cada processador envia seu melhor indivíduo para o(s) vizinho(s). Após um número finito de etapas, todos os processadores tem a melhor solução global.

Stepping-stone1 (SS1): denota o envio da melhor solução de cada processo para o vizinho da esquerda e da direita

Stepping-stone2 (SS2): denota o envio da melhor solução de cada processo para um vizinho.

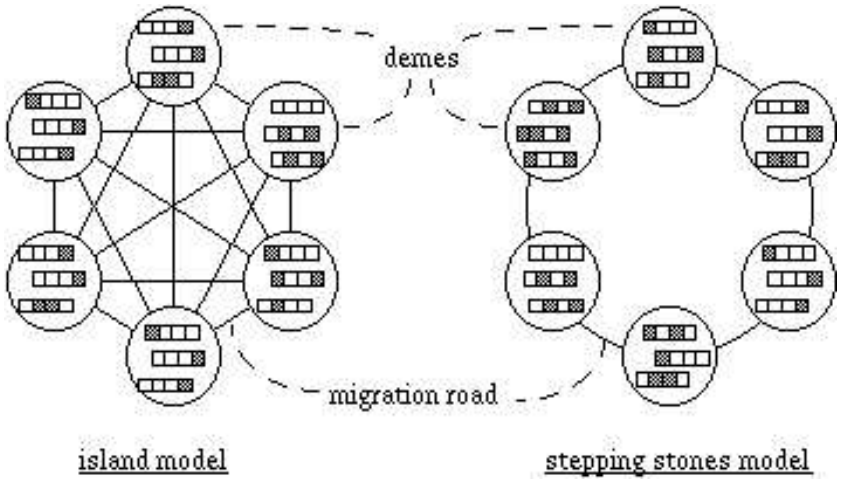

FIGURA 3.9: Modelos de Granularidade Grossa

Tecnicamente existem três importantes características em um algoritmo genético paralelo de granularidade grossa : a *topologia* que define as conexões entre as subpopulações, a *taxa de migração* que controla quantos indivíduos migram e o *intervalo de migração* que indica com que freqüência a migração ocorre (da Silva & Simoni 2001).

Capítulo 4

Proposta de Ferramenta no Modelo de Ilhas

4.1 Introdução

Este capítulo trata da implemetação de três protótipos de software que permitem a solução de problemas de otimização via Algoritmos Genéticos em ambiente paralelo. Um deles foi implementado por um algoritmo hibrido que utiliza redes neurais juntamente com os algoritmos genéticos. Os softwares foram desenvolvidos na linguagem Fortran com uma única rotina em C. Para permitir a troca de mensagens entre processos foi utilizada a biblioteca MPI.

Durante a fase inicial de implementação do Algoritmo Genético, de agora em diante chamado AGP, procurou-se chegar a uma implementação que fosse a mais prática e simples possível atendendo as características e restrições do do algoritmo.

4.2 Trabalhos Correlatos

(Cantú-Paz 1998) Este trabalho apresenta um estudo de algoritmos genéticos paralelos com múltiplas populações. Este estudo torna explícita a relação entre a probabilidade de obter uma solução desejada com o tamanho da ilha, a taxa de migração e o grau de conectividade do grafo.

(Sambatti et al. 2005) Um algoritmo genético paralelo é empregado para resolver problemas inversos. O PGA é codificado considerando os modelos **island model** e **stepping-stone model**. O código paralelo é gerado usando chamadas com MPI.

(Zanchettin & Ludermir 2005) Este trabalho apresenta uma técnica que integra as heurísticas **tabu search**, **simulated annealing**, algoritmos genéticos e **backpropagation**. Esta abordagem obteve promissores resultados na otimização simultânea da arquitetura e dos pesos de uma rede neural artificial.

(Loureiro, Margoto, Varejão & Queiroga 2005) Este trabalho propõe uma automatização do processo de busca de parâmetros para técnicas de classificação. Este processo automático foi utilizado em uma aplicação particular no domínio de fraudes.

(Brasileiro, de Oliveira Galvão & Brasileiro 2005) Este trabalho propõe um algoritmo genético para solucionar o problema do controle em tempo real de redes de escoamento complexas. Os resultados mostram que o AG encontra soluções mais econômicas que os procedimentos ad hoc de operação da rede, com uma redução média considerável.

(Basgalupp, Machado, de Souza & da Silva 2005) O trabalho trata da utilização de algoritmos genéticos para melhorar a definição da arquitetura de uma rede neural MLP numa aplicação prática de análise de séries temporais.

(Brasil, de Azevedo & Barreto 1998) Este trabalho propõe a utilização de um algoritmo genético para determinar o número de neurônios da camada intermediária de uma rede neural.

(Pacheco, Vellasco, Lopes & Passos 1998) Este trabalho faz um estudo da representação do cromossoma para otimizar as regras de associação em base de dados.

(Michalewicz, Logan & Swaminathan 1998) Este trabalho faz um estudo sobre operadores genéticos que operem sobre cromossomos com representação real.

(Cantú-Paz & Goldberg 1999) Este trabalho produz um guia para a escolha racional dos parâmetros de um algoritmo genético paralelo com múltiplas populações.

4.3 Ambiente Experimental

Nesta seção apresentamos alguns aspectos relacionados ao ambiente experimental, importantes para nossos experimentos empíricos.

Os programas tradicionais para o cálculo numérico têm sido escritos em Fortran ou C

com todas as vantagens e limitações destas linguagens. A opção pela linguagem Fortran se deve ao fato desta manipular vetores e matrizes com grande praticidade.

O Fortran (**Formula Translation**) é uma linguagem de alto nível, desenvolvida entre 1954 e 1958 por John Backus e colaboradores. Como o prórpio nome diz, ele permite uma tradução quase direta de fórmulas, por meio de uma simbologia de variáveis e operadores algébricos, sendo assim por excelência uma linguagem voltada para problemas que possam ser formulados matematicamente, em particular nos campos da física, da engenharia, da estatística e da própria matemática. Apesar de permitir a elaboração de códigos extremamente sofisticados para a resolução de problemas de grande complexidade, o amplo sucesso do Fortran nos meios acadêmicos e científicos deve-se ao uso de uma terminologia simples - OPEN, READ, STOP - aliada à codificação de cima para baixo (**top-down approach**), linha por linha, aproximando-se bastante do procedimento manual para a resolução desses problemas. Assim, a idéia é expressar de maneira simples o problema a ser resolvido.

Fortran em sua versão mais recente recebeu novas bibliotecas e funções que melhoraram ainda mais o seu desempenho no tratamento de vetores e matrizes.

Existem compiladores do Fortran para Linux e Windows, sendo que o utilizado nesta tese foi inicialmente o **ifort 7.1** da INTEL para Linux, mas este apresentou problemas em sua rotina de geração de números aleatórios, pois não conseguia inicializar a semente para que a cada nova geração do algoritmo genético uma população de indivíduos fosse diferente da geração anterior. Este problema foi constatado na fase inicial quando estava se implementando um algoritmo genético seqüêncial e este estava sendo testado no cluster Infoserver-Itautec Mercury localizado no NACAD-COPPE/UFRJ com as seguintes características:

Quantidade de nós	16 nós dual processados Intel Pentium III de 1GHz
Memória por nó	512MB RAM e cache de 256KB por CPU
Memória total	8.0 Gbytes (distribuída)
Rede	dedicada com tecnologia Fast-Ethernet (100 MBits/s)
Sistema operacional	Linux distribuição RedHat 7.3
Compiladores	Fortran-77, Fortran-90 e C/C++

TABELA 4.1: Cluster Infoserver Mercury - característica geral

O Cluster Infoserver Mercury possui além dos 16 nós, denominados node1 a node16, 1 estação de administração, denominada adm e 1 estação de acesso, denominada acc1.

Cada um dos nós, incluindo as estações adm e acc1, é uma estação de trabalho completa, com duas CPUs, memória RAM, disco local e suas próprias interfaces de rede.

Os 16 nós são interligados por uma rede fast ethernet, dedicado exclusivamente para a execução de programas paralelos.

Na tabela 4.2 são mostradas as principais características de cada um dos nós

...	**node1 a node16**	**adm**	**acc1**
Processor type	Pentium III	Pentium III	Xeon
Clock frequency	1 Ghz	1 Ghz	1 Ghz
Cache size	256 KB	256 KB	256 KB
RAM memory	512 MB	512 MB	512 MB
Disk storage	18.0 GB	160 GB	18.0 GB

TABELA 4.2: Cluster Infoserver Mercury - característica por nó

Foi constatado que o problema estava no compilador, pois o algoritmo foi executado também no software Visual Studio (Fortran da Compaq) e este conseguia inicializar a semente a cada nova geração. Este problema foi solucionado com uma nova versão do compilador da INTEL, o **ifort 8.1** que foi baixado e testado em um PC e conseguiu inicializar a semente.

O NACAD adquiriu um SGI ALTIX 350 da Silicon Graphic que utiliza este compilador e deste modo as implementações foram executadas nesta máquina, sem mais problemas com a geração de números aleatórios.

As características do ALTIX são :

Quantidade de CPUs	14 CPUs Intel Itanium2: palavra de 64 bits
Memória	28 Gbytes RAM (compartilhada - NUMA)
Armazenamento em disco	360 Gbytes
Sistema operacional	RedHat Enterprise Linux + SGI ProPack
Compiladores	Intel e GNU (Fortran-90 e C/C++) com suporte OpenMP e MPI

TABELA 4.3: SGI ALTIX 350 - característica geral

O SGI Altix-350 é um sistema de memória compartilhada (tecnologia NUMA) e como o sistema operacional é nativo de 64-bits (RedHat Enterprise Linux + SGI ProPack) isso implica na necessidade de recompilar códigos nativos de 32-bits. Possui um Pico Teórico de Performance de 6 GFlop/s por CPU.

4.4 Algoritmo Baseado no Modelo de Ilhas

Partindo da abordagem de *Granularidade Grossa* para Algoritmos Genéticos Paralelos, foram implementados dois algoritmos em que cada um utiliza um dos processos de migração denominados **SS1** e **SS2**.

Foi escolhida esta abordagem em virtude da estratégia de migração **island model** ter um elevado custo computacional devido a maciça comunicação entre as ilhas.

Na estratégia **SS1** e **SS2** ocorre comunicação somente entre duas ou no máximo três ilhas quando ocorrer uma migração, com isto podemos obter um rendimento melhor no que diz respeito ao tempo de comunicação gasto entre as ilhas. Duas ou três ilhas que estejam se comunicando após terem terminado esta fase aguardam somente comunicação com no máximo duas ilhas. Isto já favorece um baixo custo de comunicação e um sincronismo bem melhor.

As estratégias **SS1** e **SS2** foram implementadas com a utilização do *vetor de estados*, sendo que foi feita a opção para que uma população, ao final de sua evolução, pare e aguarde até que as outras populações terminem suas evoluções, o que não ocorre conforme visto na seção 3.3.1.1. Esta opção foi escolhida para se poder avaliar somente a evolução das populações durante uma quantidade de gerações determinada (*critério de parada*). As estratégias **SS1** e **SS2** foram implementadas com as características de um algoritmo genético seqüêncial simples, onde as únicas exceções seriam os operadores de cruzamento e mutação para valores reais e o operador de migração que é natural na abordagem de algoritmos genéticos paralelos de granularidade grossa. Com isso pode-se avaliar a capacidade do operador de migração aumentar a *diversidade genética* nas populações que estagnaram.

Para se conseguir resultados satisfatórios, deve haver diversidade genética suficiente no âmbito da população, durante as gerações, de tal modo a permitir que outras regiões, além da definida pelo indivíduo de melhor desempenho (*aptidão*), sejam representadas na população.

O aumento da diversidade genética, diminui a possibilidade do algoritmo genético ficar preso numa região de ótimo local. Se não houver nenhum mecanismo que restaure essa diversidade, a convergência pode ficar comprometida (Soares 1997) e o operador de migração é um dos mecanismo que pode restaurar essa diversidade. Um outro mecanismo seria a variação dinâmica das taxas de cruzamento e mutação, no decorrer das gerações, o que não foi adotado nesta dissertação.

A possibilidade de se avaliar populações que melhoraram suas aptidões após já terem alcançado a quantidade de gerações determinada, será objeto de pesquisas em trabalhos futuros.

Na figura 4.1 é exibido o fluxograma do modelo de ilhas para uma ilha.

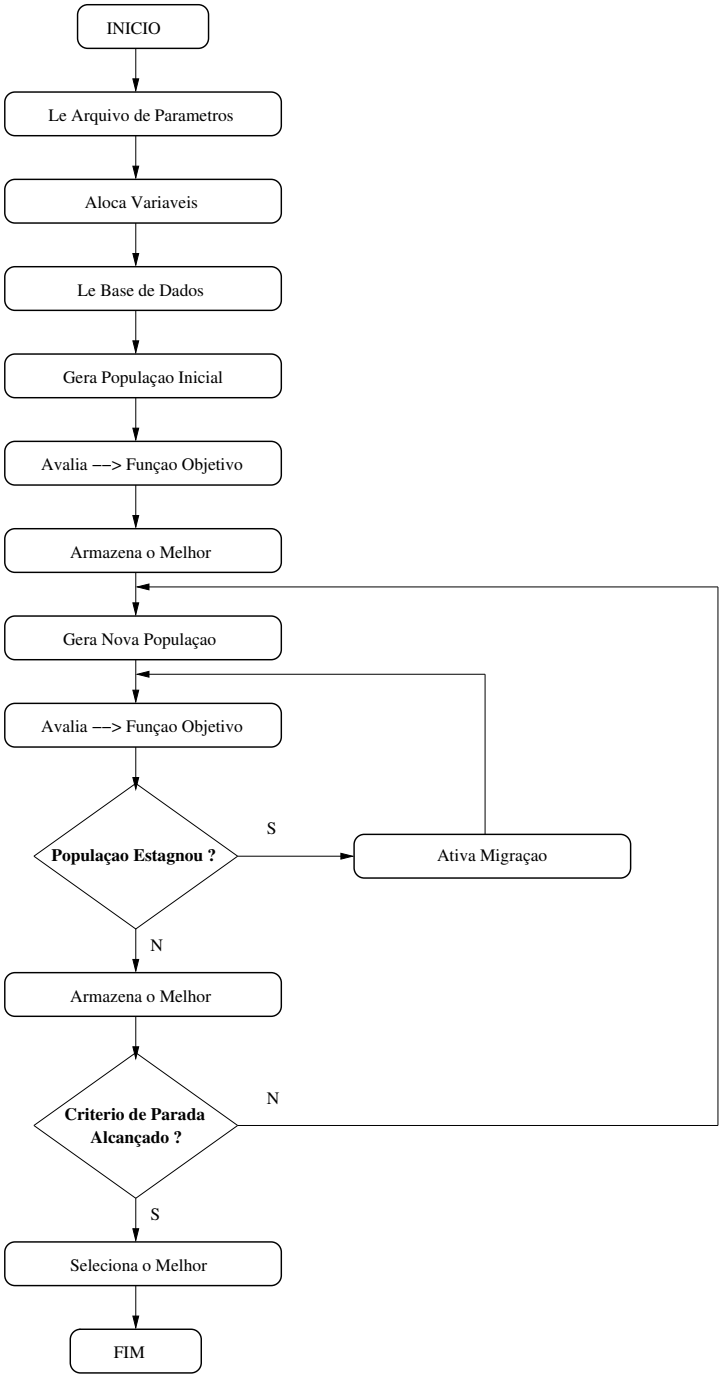

FIGURA 4.1: Fluxograma do modelo de ilhas para otimização de uma RNA

O algoritmo inicia com a leitura do arquivo que contém os parâmetros necessários para a configuração do algoritmo genético, contendo o nome da base de dados e a quantidade de registros e atributos, além de outros parâmetros listados a seguir:

- tamanho da população
- tamanho do cromossomo
- taxa de crossover
- taxa de mutação
- quantidade de gerações
- quantidade de indivíduos que vão migrar
- quantidade de indivíduos que indicam que a geração estagnou
- quantidade de gerações que indicam que a população estagnou
- nome da base de dados
- quantidade de registros
- quantidade de atributos

Com a leitura deste arquivo através da rotina *readfile()* são carregadas as variáveis da tabela 4.4

Variável	Parâmetro
popsize	tamanho da população
cromosize	tamanho do cromossoma
generationsize	quantidade de gerações
crossoverate	taxa de crossover
mutationrate	taxa de mutação
migrationrate	quantidade de indivíduos que irão migrar
tx_ind	quantidade de indivíduos estagnados que indicam que a população na geração estagnou
tx_ger	quantidade de gerações estagnadas que disparam o processo de migração
basename	nome da base de dados
m	quantidade de registros
n	quantidade de atributos

TABELA 4.4: Variáveis e parâmetros

Após estas variáveis serem carregadas ocorre a chamada da rotina *alocar()* que irá alocar dinamicamente as variáveis necessárias para armazenar a população e a base de dados. Esta alocação dinâmica permite uma maior flexibilidade do algoritmo para qualquer base de dados.

Variável	Característica
popinicial(popsize,cromosize)	armazena a população inicial
newpop(popsize,cromosize)	armazena a população gerada a cada nova geração
aptidaoatual(popsize)	armazena o finess da geração *i*
novaaptida(popsize)	armazena o finess da geração *i+1*
mejor(popsize)	armazena o melhor fitness entre todas as populações em cada geração
dados(m,n)	armazena a base de dados

TABELA 4.5: Variáveis alocadas dinamicamente

A partir do momento em que uma população é gerada e todos os seus membros são avaliados, o algoritmo genético inicia a emulação do ciclo de vida. A cada passo da iteração

descendentes são gerados. Os pais são combinados para reproduzirem e gerarem descendentes. Com esta idéia temos que antes da primeira iteração existe uma população inicial que é gerada aleatoriamente. Quando se iniciar a primeira iteração uma nova população será gerada a partir da população inicial. Ao fim da primeira iteração e início da próxima, esta nova população irá gerar uma outra nova população. Com isso temos que ao longo de todo processo de evolução (da primeira iteração até a última) serão consideradas somente duas populações: a nova, da iteração atual, e a antiga, da iteração anterior.

Um indivíduo (*cromossoma*) da população inicial, nova ou anterior possui o tamanho $n + 3$, onde n é a quantidade de atributos da base de dados e o valor 3 indica 3 parâmetros que também são utilizados pela rede neural. Estes parâmetros são: taxa de aprendizado, momentum e quantidade de neurônios na camada intermediária. A implementação adota somente uma camada intermediária para a rede neural como está definido no algoritmo da rede (Costa 1999). As n posições restantes podem assumir os valores um ou zero dependendo se o atributo da base de dados for utilizado ou não. Com isto teremos um vetor de tamanho $n + 3$ com as três primeiras posições como valor real e as *n* posições restantes com valores também reais, porém manipuladas como valores binários.

Com a população inicial gerada calcula-se a aptidão da cada inidivíduo utilizando-se a rotina *funcobjetivo()* que neste caso é o erro *RMS* do treinamento da rede neural correspondente por uma quantidade específica de épocas.

A função objetivo é uma chamada a rotina externa *treinarede()* que é a rede neural utilizada em (Costa 1999). O código fonte original da rede neural é um programa feito em C, mas quando a rede neural foi utilizada no algoritmo genético paralelo, este programa foi transformada em uma subrotina (**function**) que foi referenciada como uma rotina externa ao Fortran.

Antes havia se pensado em reescrever todo o programa da rede neural em Fortran, mas pesquisas feitas na bibliografia e manuais sobre Fortran indicaram o uso de rotinas externas no Fortran em linguagem C ou C++.

A função objetivo utilizada é definida como :

funcobjetivo(entradas,saídas,q_ entradas,q_ saídas,q_ reg,q_ indiv, indiv,q_ epocas,fitness)

Na tabela 4.6 são exibidos e descritos os parâmetros da função objetivo.

Parâmetro	Característica
entradas	matriz de entradas
saídas	matriz de entradas
q_ entradas	quantidade de entradas
q_ saídas	quantidade de saídas
q_ reg	quantidade de registros
q_ indiv	quantidade de indivíduos a serem treinados
indiv	matriz de indivíduos
q_ épocas	quantidade de épocas
fitness	vetor de erros

TABELA 4.6: Parãmetros da função objetivo

A aptidão de um indivíduo simplesmente é o erro **RMS** resultado do treinamento de uma rede por *x* épocas, com isso temos que uma população de tamanho *p* indica *p* possíveis configurações para a rede neural a ser otimizada.

Obtem-se o menor (ou maior) valor de aptidão e o correspondente indivíduo com o uso de duas funções do Fortran

Função	Característica
MAXVAL(aptidao)	retorna o maior valor no vetor aptidao
MINVAL(aptidao)	retorna o menor valor no vetor aptidao
MAXLOC(aptidao)	retorna a posição do maior valor no vetor aptidao
MINLOC(aptidao)	retorna a posição do menor valor no vetor aptidao

TABELA 4.7: Funções do Fortran

Ainda nesta etapa é gerado um arquivo $procn\_ger.txt$ que armazena a maior (ou menor) aptidão e o correspondente indivíduo na geração.

Um indivíduo de uma população possui outras características importantes, definidas por parâmetros limitados de acordo com o especificado abaixo:

- O parâmetro que fornece a quantidade de neurônios na camada intermediária é sempre maior ou igual a dois e menor ou igual a três vezes a quantidade de atributos da base de dados que treina a rede neural
- O momentum tem que pertencer ao intervalo [0.1,0.9]
- A taxa de aprendizado tem que pertencer ao intervalo $[10^{-9}, 0.1]$

Todos os indivíduos da população e as respectivas aptidões são armazenados e é selecionado o melhor indivíduo.

Os procedimentos descritos até aqui são iguais em todas as ilhas, ou seja, cada ilha tem sua própia população inicial (que é diferente das outras), calcula a aptidão, armazena a melhor aptidão e o indivíduo a qual está associado em um arquivo respectivo a sua ilha. Quando o fluxo do programa retorna a rotina principal todos as ilhas enviam seu melhor indivíduo e a respectiva aptidão para um processo que seleciona o melhor entre todos, denominado processo *mestre*. Esta operação é efetuada através de uma rotina específica do **MPI**:

$$mpi\_reduce()$$

que faz com que todos os processos executem uma operação aritmética, *minval(valor mínimo)* onde o resultado parcial de cada processo é combinado e retornado para um processo mestre.

O processo mestre irá armazenar o melhor par *indivíduo-aptidão* em um arquivo chamado $mastern\_ger.txt$. Neste ponto inicia-se a contagem das gerações. Assim uma geração é iniciada e uma nova população é gerada em cada ilha com a rotina *novapop()* e esta por sua vez ativa três outras rotinas: *selecao()*, *crossover()* e *mutacao()*.

A população anterior é utilizada para gerar a nova população como já dito acima. Para esta fase usa-se como processo de seleção o método da roleta, já descrito na seção 2.5.1. Este método seleciona os melhores pais de acordo com sua aptidão e aplica sobre estes os operadores de *cruzamento* e *mutação*, através das rotinas *crossover()* e *mutacao()*. Os indivíduos da população são escolhidos dois a dois para que possa ser aplicada a operação de cruzamento e depois a mutação em cada um dos filhos gerados (nova população).

Tanto na fase de cruzamento (**blend**) e mutação (**creep**) quando estes são aplicados sobre o par de pais escolhidos e filhos gerados, as restrições já descritas que um indivíduo deve respeitar são testadas. Isto ocorre a cada nova populaçao gerada pelo algoritmo. Após a verificação desses requisitos é calculado a aptidão de cada indivíduo e estes são armazenados (indivíduo-aptidão) e comparados através da rotina *compara()* com a aptidão da população anterior. Esta comparação serve para saber se a população evoluiu ou não.

A implementação considera que um indivíduo estagnou de uma geração para outra quando seu valor de aptidão permanece o mesmo. Um indivíduo não evolui quando esta estagnação

é consecutiva ao longo de *p* gerações. O valor *p* é calculado sobre o total de gerações estipuladas pelo algoritmo. Este valor é o parâmetro tx_ger

Uma população está estagnada de uma geração para outra quando um percentual representativo desta, isto é, y indivíduos estão estagnados. Este percentual representativo é o parâmetro tx_ind. Assim temos que uma população não evoluiu quando a quantidade *p* de gerações fica estagnada.

Na comparação caso a quantidade de indivíduos estagnados seja igual ao parâmetro tx_ind um contador é incrementado e este depois é comparado ao parâmetro tx_ger e se for igual é disparado o procedimento de *migração* (**SS1**ou **SS2**) através da rotina *migracao()*. Quando isto acontece as ilhas ligadas, pela topologia de anel, à ilha que não evoluiu, interrompem suas evoluções e lhe enviam uma quantidade de seus melhores indivíduos (*taxa de migração*).

Enquanto a migração está ocorrendo com um ou mais ilhas, as ilhas que não participam da migração continuam seu processo de evolução.

Após o término da migração as ilhas envolvidas neste procedimento retornam a sua evolução.

Novamente ocorre a chamada da rotina *mpi _ reduce* onde as melhores aptidões são enviadas ao processo mestre e é armazenada a melhor entre estas. O critério de parada (quantidade máxima de gerações) é vericado e caso seja satisfeito o melhor indivíduo e correpondente aptidão são armazenados.

Ao final de uma geração cada ilha armazena os seus melhores indivíduos e respectivos fitness e quando a última ilha executa esta ação o algoritmo genético irá armazenar o melhor indivíduo e correpondente fitness entre todas as ilhas naquela geração.

Como cada ilha pode evoluir de maneira distinta uma das outras (estagnou / não estagnou) utiliza-se a rotina de **MPI**

$$mpi\_barrier()$$

após o critério de parada ter sido alcançado, a fim de sincronizar todos os processos para que o algoritmo termine sua execução.

4.4.1 Testes

Para mostrar o funcionamento dos algoritmos SS1 e SS2 serão analisados problemas de minimização conhecidos na literatura : Tanomaru, De Jong, etc...

4.4.1.1 Função De Jong1

$$f_1(x) = \sum_{i=1}^{3} (x_i)^2 \ ; \ -5.12 \leq x_i \leq 5.12 \tag{4.1}$$

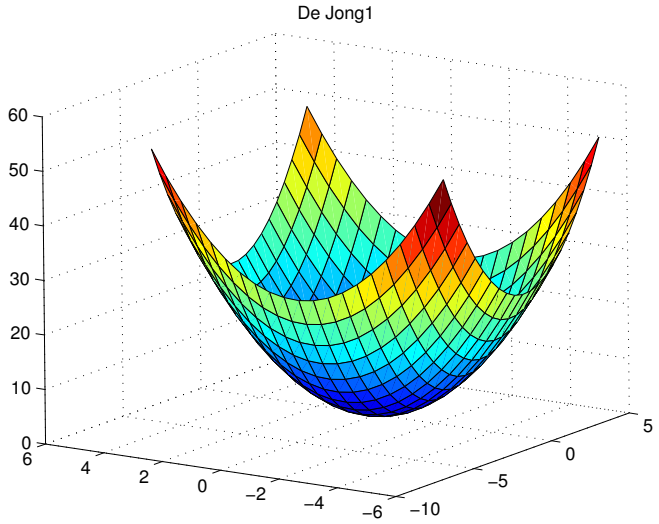

FIGURA 4.2: Função De Jong1

Esta função possui mínimo igual a 0 e ponto de mínimo em (0,0,0). É uma função em que o seu mínimo é calculado com pouco esforço computacional.

Na tabela 4.8 é comparado os resultados obtidos em (de Mendonça 2004) com as implementações **SS1** e **SS2**.

O trabalho apresentado em (de Mendonça 2004) é o desenvolvimento de um sistema que permite a otimização via algoritmos genéticos em ambientes paralelos ou não, fornecendo também o suporte a criação, treinamento e uso de redes neurais. Este sistema se propõe a servir como ferramenta para a otimização de problemas práticos de engenharia, que pode envolver simulação através do uso de diversos aplicativos para o cálculo das restrições e da função objetivo. Entre os testes realizados neste sistema está a otimização das funções de De Jong os quais serão comparados aos realizados nesta dissertação.

Em (de Mendonça 2004) a otimização das funções de De Jong foi obtida utilizando-se uma rede neural que calculava os valores ótimos e cujos parâmetros eram otimizados por um

algoritmo genético. Nesta dissertação foi usado somente o algoritmo genético paralelo de granularidade grossa.

Na otimização das funções de De Jong, **SS1** e **SS2** utilizaram populações maiores que em (de Mendonça 2004). O tipo de seleção adotado foi o método da roleta. Os operadores de cruzamento e mutação e o tamanho e tipo do cromossoma foram os mesmos que em (de Mendonça 2004). Cada implementação foi executada 6 vezes para cada função e a quantidade de ilhas adotadas foi 10.

Características	**(de Mendonça 2004)**	**SS1**	**SS2**
Geração	34	2	2
Mínimo	3.409e-09	0	0
Ponto de mínimo	(-4.8e-05 , 3.3e-05 , 4.0e-06)	(0,0,0)	(0,0,0)
Qde. de gerações	200	200	200
Tamanho da população	45	1000	1000
Tamanho do cromossomo	3	3	3
Tipo de seleção	torneio com dois indivíduos	roleta	roleta
Tipo de cruzamento	blend	blend	blend
Taxa de cruzamento	0.95	0.8	0.8
Tipo de mutação	creep	creep	creep
Taxa de mutação	0.20	0.20	0.20
Taxa de migração		0.2	0.2
Taxa de gerações		0.2	0.2
Ilhas		10	10

TABELA 4.8: Resultados obtidos em (de Mendonça, C. 2004) e SS1 e SS2 para a função de De Jong 1

SS1 e **SS2** conseguiram alcançar o mínimo já na 2ª geração e as variações observadas no grau de diversidade em ambos mostrou que as populações estavam bem próximas de um ótimo já na 8ª geração. A função não apresentou grandes dificuldades para o cálculo do mínimo. Não foram observadas estagnações em nenhuma das população para esta função.

4.4.1.2 Função De Jong2

$$f_2(x) = 100 * (x_1^2 - x_2^2)^2 + (1 - x_1)^2 \ ; \ -2.048 \leq x_i \leq 2.048 \tag{4.2}$$

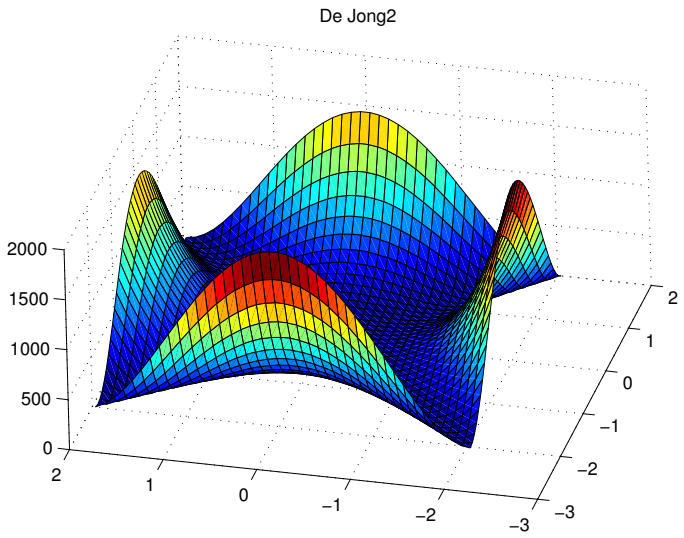

FIGURA 4.3: Função De Jong2

Esta função possui mínimo igual a 0 e ponto de mínimo em (1,1). Apesar de possuir valores simples tanto para o mínimo quanto para o ponto de mínimo esta tambem apresenta uma quantidade maior de mínimos locais. Neste aspecto os aperadores de cruzamento e mutação servem de auxilio para que se consiga escapar da convergência prematura, isto é, mínimos locais.

Na tabela 4.9 são comparados os resultados obtidos em (de Mendonça 2004) com as implementações **SS1** e **SS2**

A função apresenta mais mínimos locais próximos do mínimo global que a função anterior. Este fato fez com **SS1** e **SS2** tivessem quase o mesmo desempenho. Entre os valores encontrados, somente o calculado em **SS2** foi melhor que em (de Mendonça 2004). O fato de **SS1** encontrar um valor ainda maior que em (de Mendonça 2004) torna claro que a migração é um fator determinante para que se possa melhorar o resultado, pois em **SS2** as estagnações foram maiores, mas em virtude deste receber mais indivíduos na migração isto possibilitou a melhora na procura da solução. Os operadores de cruzamento (**blend**) e mutação (**creep**) por tratarem valores reais, representaram um bom diferencial para que se pudesse evitar a convergência prematura.

Características	(de Mendonça 2004)	SS1	SS2
Geração	47	10	10
Mínimo	1.228e-06	1.311e-06	1.179e-06
Ponto de mínimo	x_1 = 0.998895 x_2 = 0.997783	x_1 = 0.987835 x_2 = 0.985565	x_1 = 0.999274 x_2 = 0.999355
Qde. de gerações	1000	200	200
Tamanho da população	30	1000	1000
Tamanho do cromossomo	2	2	2
Tipo de seleção	torneio com dois indivíduos	roleta	roleta
Tipo de cruzamento	blend	blend	blend
Taxa de cruzamento	0.95	0.8	0.8
Tipo de mutação	creep	creep	creep
Taxa de mutação	0.20	0.20	0.20
Taxa de migração		0.2	0.2
Taxa de gerações		0.2	0.2
Ilhas		10	10

TABELA 4.9: Resultados obtidos em (de Mendonça, C. 2004) e SS1 e SS2 para a função de De Jong 2

4.4.1.3 Função De Jong3

$$f_3(x) = \sum_{i=1}^{5} inteiro(x_i) \ ; \ -5.12 \leq x_i \leq 5.12 \tag{4.3}$$

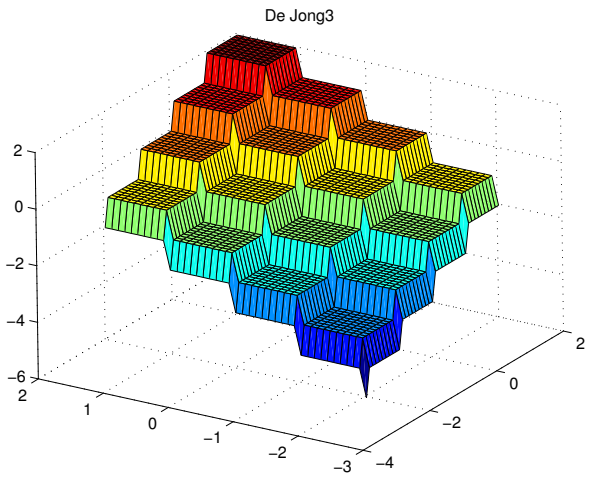

FIGURA 4.4: Função De Jong3 em 3 dimensões

Esta função é como uma escada em 5 dimensões. Há um patamar para os pontos $(x_i)i \in 1, 2, 3, 4, 5, x_i \in [-5.12, -5]$ onde o valor da função atinge o mínimo -25. A figura 4.4 representa a função para os pontos $(x_i)i \in 1, 2, 3$. A função apresenta *platôs* que indicam a

possibilidade de convergência prematura ou estagnação.

Na tabela 4.10 são comparados os resultados obtidos em (de Mendonça 2004) com as implementações **SS1** e **SS2**

Características	**(de Mendonça 2004)**	**SS1**	**SS2**
Geração	35	21	17
Mínimo	-25	-25	-25
Ponto de mínimo	x_1 = -5.119999 x_2 = -5.118405 x_3 = -5.065301 x_4 = -5.063194 x_5 = -5.032257	x_1 = -5.120149 x_2 = -5.105311 x_3 = -5.051357 x_4 = -5.065117 x_5 = -5.029987	x_1 = -5.119588 x_2 = -5.118405 x_3 = -5.058691 x_4 = -5.066411 x_5 = -5.032857
Qde. de gerações	50	200	200
Tamanho da população	75	1000	1000
Tamanho do cromossomo	5	5	5
Tipo de seleção	torneio com dois indivíduos	roleta	roleta
Tipo de cruzamento	blend	blend	blend
Taxa de cruzamento	0.95	0.8	0.8
Tipo de mutação	creep	creep	creep
Taxa de mutação	0.20	0.20	0.20
Taxa de migração		0.2	0.2
Taxa de gerações		0.2	0.2
Ilhas		10	10

TABELA 4.10: Resultados obtidos em (de Mendonça, C. 2004) e SS1 e SS2 para a função de De Jong 3

A função apresentou dificuldades para o cálculo do mínimo em virtude dos mínimos locais que se encontraram. **SS1** e **SS2** conseguiram o ponto de mínimo em gerações muito próximas, indicando que os operadores genéticos foram mais representativos para a população que o operador de migração. Como **SS2** recebe mais indivíduos por migração é esperado que este consiga melhorar a qualidade de sua população e conseqüentemente caminhar com mais precisão e direção na busca pela solução, isto é, pode chegar mais rápido a uma boa solução.

4.4.1.4 Tanomaru

$$f(x) = \cos(20x) - \frac{|x|}{2} + \frac{x^3}{4} \; ; \; -2 \leq x \leq 2 \tag{4.4}$$

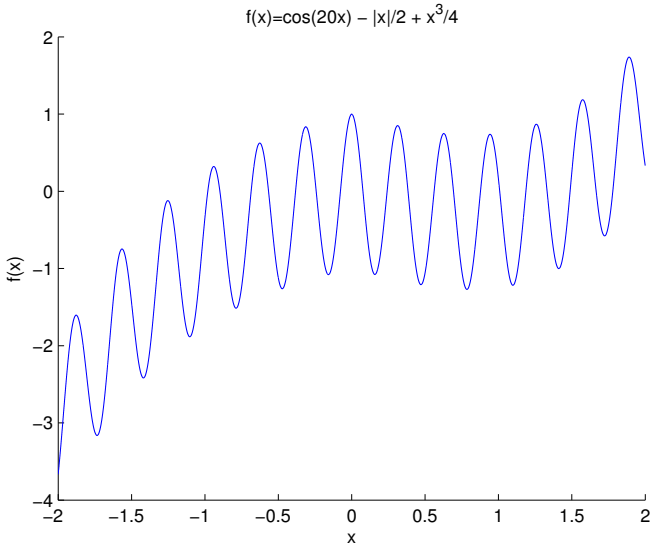

FIGURA 4.5: Função de Tanomaru

Esta função é não-linear com 12 máximos locais e um máximo global no intervalo de interesse. Possui máximo global aproximado à 1.737752 e ponto de máximo aproximado 1.88929. Por se tratar de uma função multimodal, é evidente que puros métodos de gradiente não devem ser capazes de encontrar o ótimo global na maior parte das tentativas.

Para esta função, **SS1** e **SS2** utilizaram os mesmos operadores genéticos e método de seleção utilizados em (Tanomaru 1995). A população inicialmente foi codificada como um vetor real e depois foi alterada para a codificação em binário para que se avaliasse o desempenho de **SS1** e **SS2** com a mesma configuração em (Tanomaru 1995).

Na tabela 4.11 são comparados os resultados obtidos em (Tanomaru 1995) com as implementações **SS1** e **SS2**

SS1 e **SS2** tiveram desempenho melhor que o algoritmo genético seqüêncial em (Tanomaru 1995). Conseguiram o ponto de máximo em gerações muito próximas e bem menores que em (Tanomaru 1995). Com a representação em binário, os operadores genéticos tiveram o mesmo desempenho e este fato mostrou que **SS1** e **SS2** tiveram um melhor desempenho em virtude do operador de migração.

Características	(Tanomaru 1995)	SS1	SS2
Geração	48	10	7
Máximo	1.88929	1.88966	1.88966
Ponto de máximo	1.73752	1.73765	1.73765
Qde. de gerações	50	200	200
Tamanho da população	50	1000	1000
Tamanho do cromossomo	16	16	16
Tipo de seleção	roleta	roleta	roleta
Tipo de cruzamento	1 ponto	1 ponto	1 ponto
Taxa de cruzamento	0.8	0.8	0.8
Tipo de mutação	aleatória	aleatória	aleatória
Taxa de mutação	0.01	0.01	0.01
Taxa de migração		0.2	0.2
Taxa de gerações		0.2	0.2
Ilhas		10	10

TABELA 4.11: Resultados obtidos em (Tanomaru 1995), SS1 e SS2 para a função $f(x) = \cos(20x) - \frac{|x|}{2} + \frac{x^3}{4}$

Capítulo 5

Estudo de Caso

Neste capítulo serão introduzidos conceitos sobre redes neurais e os problemas encontrados para o seu treinamento. Será apresentado o caso estudado e serão analisados os resultados obtidos para a validação dos algoritmos implementados e também para verificar a eficácia da combinação entre o algoritmo genético paralelo de granularidade grossa (**stepping-stone**) e as redes neurais.

5.1 Redes Neurais Artificiais

Existem diversas técnicas que detectam padrões em massas de dados, de forma automática, adicionando inteligência à análise dos dados e tornando-a independente do usuário. Uma das principais técnicas utilizadas em aplicações de mineração de dados é conhecida como *redes neurais artificiais*.

As redes neurais são programas computacionais que simulam o trabalho do sistema nervoso, de forma a resolver os problemas de forma eficiente e com muita tolerância aos ruídos da base de dados. Entretanto existem algumas limitações relacionadas à utilização das redes neurais, principalmente no que se refere à definição da sua configuração ideal e à extração das regras incorporadas no seu modelo (Haykin 2001).

Uma alternativa para a resolução destas limitações é a utilização de *algoritmos genéticos* (Omer 1995).

5.1.1 Introdução

A utilização de redes de neurônios ou neurais não é nova. Em 1943, McCulloch e Pitts já se interessavam pelo problema de representar funções lógicas, como o *E* ou o *OU* lógico, através de um conjunto elementar de unidades de decisão. Eles definiram desta forma o *neurônio formal* a partir dos resultados da neurologia e propuseram arquiteturas para a realização de funções lógicas.

Nessa época, sabia-se implementar uma função booleana com a ajuda de redes de neurônios, mas não se dispunha infelizmente de um algoritmo capaz de aprender uma função booleana qualquer a partir de exemplos.

Os modelos de redes neurais artificiais, realizam a manipulação de informação através da iteração de um grande número de unidades básicas de processamento, chamadas *neurônios artificiais*.

Os neurônios em uma rede neural estão organizados em camadas que se interligam, como apresentado na figura 5.1. As camadas inferiores são compostas de neurônios que se ligam com os neurônios da camada superior.

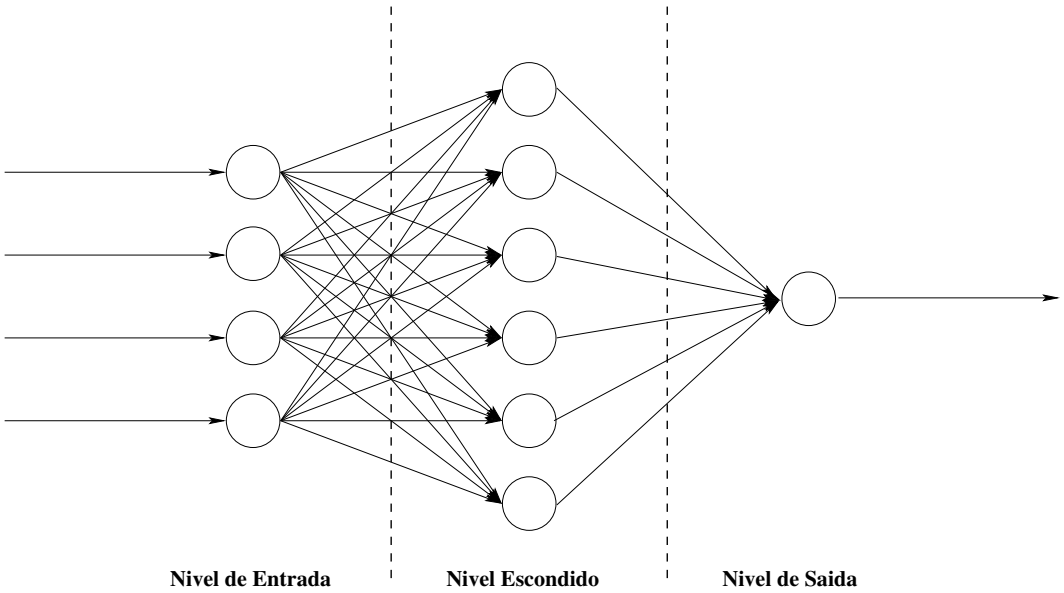

FIGURA 5.1: Arquitetura de uma RNA

A rede neural apresentada na figura 5.1 possui três camadas : a camada de entrada, a camada intermediária (ou camada escondida) e a camada de saída.

Uma rede neural pode apresentar mais de uma camada intermediária e a quantidade de conexões entre os neurônios também pode variar.

A representação de um neurônio artificial ou unidade de processamento *j* está apresentada na figura 5.2.

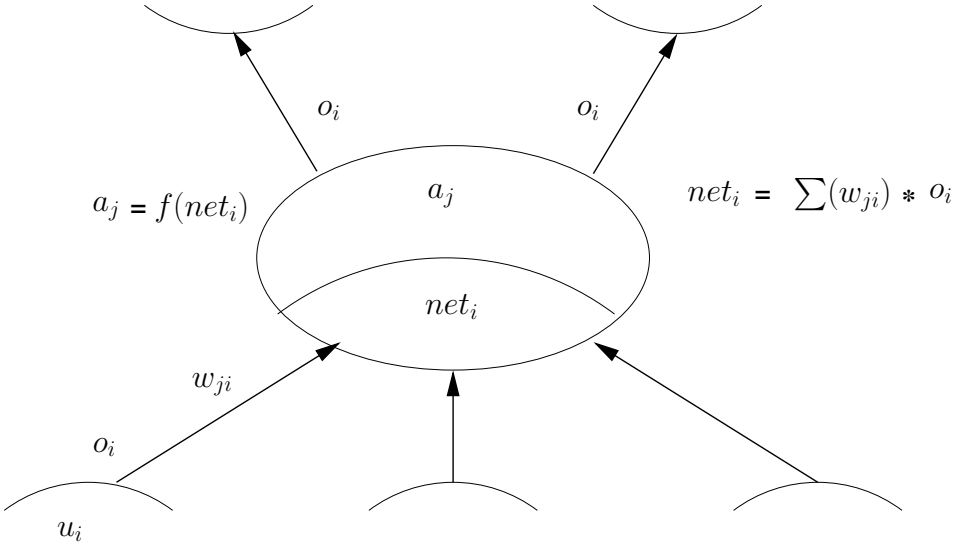

FIGURA 5.2: Neurônio artificial

os componentes da unidade *j* são:

- a_j é o valor de ativação da unidade *j*
- o_i é a saída da unidade *i* (que é uma entrada da unidade *j*)
- o_j é o valor de saída da unidade *j*
- w_{ji} é o peso da conexão entre a unidade *j* e a unidade *i*

Cada unidade da rede recebe sinais das unidades anteriores. As unidades estão interligadas através de conexões que possuem um peso. O peso define o efeito que a saída de uma unidade exerce sobre a entrada da unidade seguinte. A combinação das entradas, pela soma ponderada das mesmas, gera uma entrada total que serve de base para a modificação do estado interno da unidade, o seu valor de ativação. A saída de uma unidade também é uma função do seu valor de ativação, dada por:

$$o_j = f(a_j) \tag{5.1}$$

Cada vez que uma unidade recebe entradas um novo valor de ativação é gerado, assim como um novo valor de saída.

O grande diferencial do emprego de redes neurais artificiais na solução de problemas é sua capacidade de aprender (Abdi 1994). O aprendizado ou treinamento de uma rede consiste no ajuste dos parâmetros internos da rede, de forma que esta produza o resultado esperado, dada a apresentação de um *conjunto de padrões específicos*. Os padrões de treinamento da rede contém as informações que se desja que uma rede aprenda. Os parâmetros a ajustar são os pesos das conexões que interligam os neurônios. Os diversos modelos de redes neurais artificiais se caracterizam pela utilização de diferentes técnicas de treinamento.

Outros dois conjuntos de dados são: *conjunto de validação* e *conjunto de teste*. O conjunto de validação não é utilizado na atualização dos pesos da rede, e sim como mais um parâmetro para avaliação do rumo do treinamento. O conjunto de teste é um conjunto que deve conter somente padrões que não existam no conjunto de treinamento, sendo utilizado para avaliar a capaciade da rede responder a estímulos nunca recebidos. Independente de quais sejam os conjuntos, estes devem ser os mais representativos possíveis do universo de dados que a rede receberá ao longo da sua fase de produção.

A rede neural utilizada nesta dissertação utilizou o algoritmo de aprendizagem **back-propagation**. Este algoritmo é um procedimento iterativo que utiliza pares de valores de entrada e saída como padrões para o treinamento. Um padrão é apresentado na entrada da rede, através de seus valores de entrada, e é calculada uma saída da rede. A saída calculada é comparada com a saída desejada do padrão e, havendo diferença, os pesos das conexões entre as unidades são modificados para que esta diferença seja minimizada. A apresentação desses pares é repetida até que as diferenças atinjam um valor mínimo pré-estabelecido ou até que a quantidade de épocas percorridas, isto é, a quantidade de vezes que um conjunto de treinamento é apresentado à rede, atinja o valor determinado.

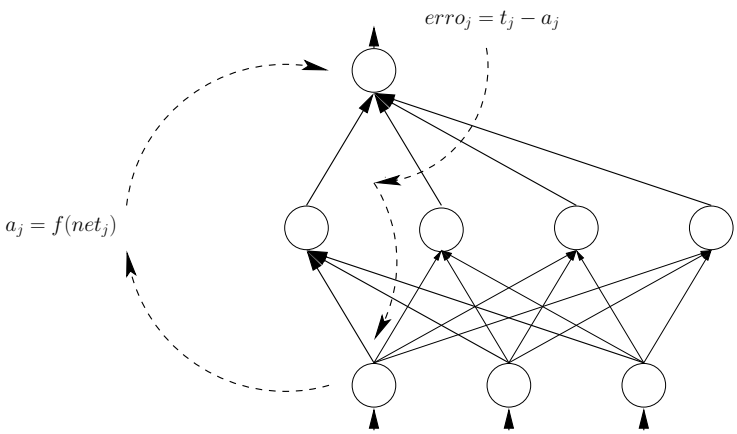

FIGURA 5.3: Algoritmo back-propagation

Neste algoritmo a função de saída das unidades é a função identidade:

$$o_j = a_j \tag{5.2}$$

A técnica utilizada para a minimização do erro calcula o *gradiente descendente* da função de erro nas unidades de saída, tendo sido denominada *regra do delta generalizada*.

5.1.2 Problemas das Redes Neurais

Para que um rede neural artificial possa ser utilizada e responder de maneira eficiente ao problema ao qual foi empregada existe a necessidade de saber se esta realmente esta apta a resolver este problema e mesmo esta fase sendo consolidada ainda resta o problema de ajustar a rede, ou melhor os seus parâmetros para que se possa obter um bom desempenho desta.

Aplicações práticas de redes neurais envolvem etapas altamente experimentais, com cansativos e ineficientes processos de tentativa e erro.

Primeiro deve-se especificar a arquitetura da rede, o número de camadas, neurônios e o padrão de conectividade entre os neurônios. Depois é necessário especificar os parâmetros de aprendizagem, os pesos inicias, e então efetuar o treino. Este processo é lento, depende de sorte e procedimentos heurísticos, e deve ser repetido até que o desejado grau de desempenho seja atingido. Há vários problemas óbvios com tal procedimento. Para melhorá-lo, algoritmos genéticos têm sido propostos com três objetivos distintos:

- Determinação dos pesos de uma rede de configuração conhecida
- Determinação da configuração para resolver um dado problema
- Determinação da configuração e pesos

Esta dissertação aborda a utilização de um algoritmo genético paralelo para otimizar a arquitetura de uma rede neural (quantidade de neurônios na camada intermediária) simultaneamente com a busca dos melhores atributos de uma base de dados.

5.1.3 Escolha de parâmetros da RNA

Alguns dos parâmetros mais relevantes na especificação de uma rede neural são:

1. Quantidade de neurônios na camada escondida: Neurônios que se encontram no nível escondido têm uma função fundamental no treinamento de redes neurais com aprendizado através de **back-propagation**. A dimensão do nível escondido, é crucial para a determinação da capacidade de aprendizado e generalização da rede. Uma rede de pequena dimensão pode não ser capaz de aprender um conjunto de padrões maior e o treinamento se tornar indefinido. Porém com o aumento da complexidade da rede, o processo de treinamento torna-se mais lento;

2. Taxa de aprendizado: A Regra de Delta Generalizada determina que a atualização do peso da conexão de uma unidade seja função do acréscimo de um $delta$ (que é calculado pela derivada parcial do erro medido na saída da rede) com uma constante de proporcionalidade negativa. Esta constante é a taxa de aprendizado e implementa um gradiente conjugado na função. A taxa de aprendizado determina o tamanho do passo do gradiente descendente e está diretamente relacionado com o tempo que o algoritmo vai levar para atingir um mínimo global do espaço de busca da solução;

3. *Momentum*: Para que se possa aumentar o desempenho do cálculo pelo método do gradiente descendente (que precisa que as diferenças aplicadas em cada passo sejam infinitesimais) adiciona-se um termo denominado *momentum*, que leva em conta as contribuições dos passos anteriores para o ajuste dos pesos. Com isto, a taxa de aprendizado pode ser maior e o método converge mais rapidamente.

Estes 3 parâmetros são de fundamental importância para a obtenção de uma solução global em um tempo considerado aceitável.

Um outro parâmetro de vital importância para o treinamento de uma rede neural é a *quantidade de atributos* a serem considerados e a seleção destes atributos. A seleção de atributos para o uso em tarefas de mineração de dados vem sendo estudada e avaliada em relação ao desempenho dos algoritmos e a obtenção de um melhor resultado, seja de classificação ou de predição. A etapa de seleção de atributos pode ser considerado um pré-processamento ou estar completamente acoplado ao algoritmo de mineração. Podem ser utilizados modelos estatísticos ou heurísticas para a determinação do melhor conjunto de dados a ser utilizado no treinamento. Um dos métodos estatísticos mais utilizados para a determinação deste conjunto é a análise de componente principal que na verdade reduz a dimensionalidade do

problema. Este método consegue eliminar atributos que são linearmente dependentes, que atrapalham o treinamento de uma rede neural.

Outros parâmetros também importantes são : *quantidade de camadas ocultas*, *quantidade de épocas* e o *tipo de conexão*.

Neste dissertação pretende-se utilizar algoritmos genéticos para a seleção de atributos e consequente redução da dimensionalidade do problema, e avaliação do aumento do desempenho e redução do custo computacional do treinamento de uma rede neural.

5.2 Estudo de Caso

O algoritmo implementado foi empregado para otimizar os parâmetros de uma rede neural (Costa 1999), como a quantidade de neurônios na camada intermediária, e simultaneamente escolher os atributos da base de dados utilizada. A rede neural utiliza o algoritmo de aprendizagem **back-propagation**, que é um procedimento iterativo que utiliza pares de valores de entrada e saída como padrões de treinamento. A rede é totalmente conectada e possui somente uma camada intermediária. Para mais detalhes sobre a implementação da rede podem ser observados em (Costa 1999).

O parâmetro utilizado para avaliar o desempenho da rede foi o erro **RMS** e este é a aptidão pelo qual um indivíduo da população foi avaliado, um indivíduo é uma possível configuração para a rede neural a ser otimizada com a seguinte estrutura

FIGURA 5.4: Representação de um indivíduo

onde,

- A : Taxa de aprendizado
- B : Momentum
- C : Quantidade de neurônios na camada oculta
- X_1, X_I, X_N : Atributos da base de dados

Na seção 5.1.3 os parâmetros da rede neural foram apresentados e discutidas as características de cada um.

A população inicial com 20 indivíduos foi gerada aleatoriamente e o total inicial de 100 gerações foi o período utilizado para a evolução. Cada indivíduo, como já mencionado, representa uma possível configuração para a rede neural, neste caso uma ilha possui 20 configurações a serem testadas. O algoritmo **SS1** foi avaliado com 4, 6 e 8 ilhas e o **SS2** com 4, 6, 8 e 10 ilhas.

Na tabela 5.1 são exibidos os valores para os parâmetros dos modelos **SS1** e **SS2**.

Parâmetros	**SS1**	**SS2**
População	20	20
Cromossomo	7	7
Tipo de Cruzamento	Blend	Blend
Tipo de mutação	Creep	Creep
Taxa de indiv.	30%	20%
Taxa de gerações	40%	30%

TABELA 5.1: Parâmetros utlizados em SS1 e SS2

A escolha dos operadores genéticos seguiu a análise e os resultados em (de Oliveira 2001).

A base de dados utilizada foi a *base íris* que possui 150 registros, 4 atributos e 3 classes. A rede utilizou 4 entradas, 1 saída e 1 camada intermediária.

Em uma primeira fase foram executados os algoritmos com os seguintes valores fixos para a rede neural:

- taxa de aprendizado : 10^{-3}
- momentum : 0.3

Os outros parâmetros, que são os atributos da base de dados, foram variados.

Testou-se a rede em 20, 40 e 60 épocas.

Nas tabelas 5.2 e 5.3 são mostrados alguns resultados obtidos com as configurações já mencionadas anteriormente.

Para este caso em que os valores de *taxa de aprendizado* e *momentum* foram fixados o algoritmo foi executados 6 vezes para cada implementação. Foi observado que em boa parte das execuções todos os resultados sempre estavam com 3 dos 4 atributos da base e se verificou uma maior incidência do primeiro atributo. Em relação aos demais atributos a incidência foi bem dispersa tanto no modelo **SS1** quanto **SS2**.

Épocas	**SS1** (4 ilhas) (Atributos - Neurônios)	**SS1** (6 ilhas) (Atributos - Neurônios)	**SS1** (8 ilhas) (Atributos - Neurônios)
20	RMS = 0.313593 1,3,4 - 3	RMS = 0.290031 2,3,4 - 3	RMS = 0.113274 1,2,4 - 3
40	RMS = 0.308514 2,3,4 - 5	RMS = 0.286173 1,3,4 - 4	RMS = 0.111587 1,2,3,4 - 3
60	RMS = 0.300375 1,2,3,4 - 5	RMS = 0.280135 2,3,4 - 4	RMS = 0.110199 1,3,4 - 3

TABELA 5.2: RMS do melhor indivíduo, atributos utilizados e quantidade de neurônios na camada intermediária obtidos em **SS1**

Épocas	**SS2** (4 ilhas) (Atributos - Neurônios)	**SS2** (6 ilhas) (Atributos - Neurônios)	**SS2** (8 ilhas) (Atributos - Neurônios)
20	RMS = 0.315154 1,2,3,4 - 3	RMS = 0.281231 2,3,4 - 3	RMS = 0.114215 2,3,4 - 3
40	RMS = 0.303816 1,2,4 - 4	RMS = 0.276173 1,3,4 - 4	RMS = 0.111033 1,2,3,4 - 3
60	RMS = 0.301237 1,3,4 - 3	RMS = 0.270135 2,3,4 - 4	RMS = 0.110123 2,3,4 - 5

TABELA 5.3: RMS do melhor indivíduo, atributos utilizados e quantidade de neurônios na camada intermediária obtidos em **SS2**

No modelo **SS1** a variação do erro foi dentros das expectativas, pois este modelo a migração ocorre sempre com um vizinho, isto proporcionou um grau de diversidade um pouco menor comparado a **SS2**. Tendo somente a variação da quantidade de neurônios na camada intermediária isto deixou liberdade para a variação dos atributos o que resultou em uma melhor observação destes. Isto serviu para que se pudesse extrair mais informações da base com respeito a importância de certos atributos.

A quantidade de neurônios na camada intermediária apresentou sempre os melhores resultados com 3 neurônios o que não pode ser usado como justificativa ou princípio para que o número de neurônios na camada intermediária seja igual a quantidade de atributos, pois a base em questão, comparada a outras, não apresenta um grau muito forte de complexidade ou classificação de seus registros. Esta característica da base íris também explica o por quê do modelo **SS2** apresentar um percentual de desempenho um pouco melhor.

Nas figuras 5.5 e 5.6 são mostrados as evoluções, em gerações, dos erros em **SS1** e **SS2** com 8 ilhas cada, com taxa de aprendizado igual a 10^{-3}, momentum igual 0.3 e número de

épocas igual a 20. **SS1** e **SS2** foram executados 6 vezes cada e os valores são os menores erros obitidos em cada geração na melhor execução.

Neste exemplo tentou-se somente avaliar a migração, mas em virtude das *oscilações* que apareceram a longo das evoluções, ficou evidente que os operadores de cruzamento e mutação tiveram importante desempenho para contornar (ou tratar) o aparecimento de indivíduos com material genético que retornasse uma aptidão não ótima.

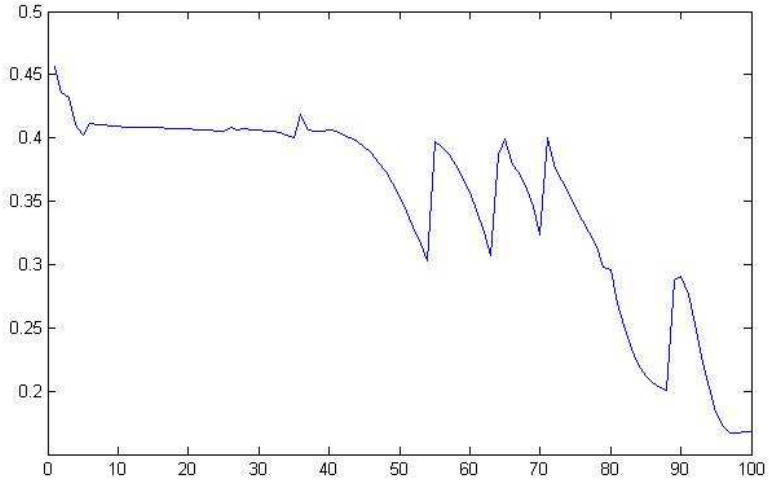

FIGURA 5.5: **SS1** com 8 ilhas : RMS x Geração

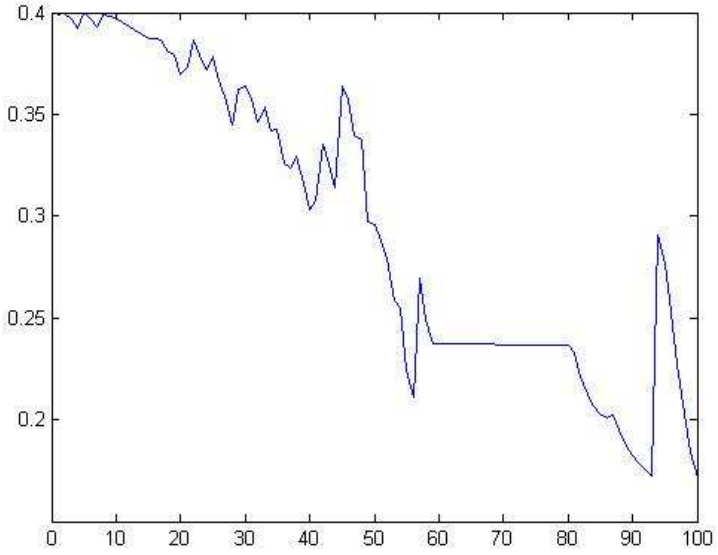

FIGURA 5.6: **SS2** com 8 ilhas : RMS x Geração

Observa-se que em ambos os gráficos as curvas são descendentes o que indica que com mais gerações pode se obter um indivíduo que seja a melhor configuração da rede, deve-se ressaltar que com 80 épocas a rede já está bem treinada para esta base de dados.

A ocorrência de *picos* com consequentes *declives* em ambos os gráficos mostra que os operadores genéticos de cruzamento (**blend**) e mutação (**creep**) conseguiram evitar que a busca se perde-se, pois a população que apresentava indivíduos com características genéticas que podiam comprometer a evolução, após a ação destes operadores em cada geração, conseguia retornar ao caminho em busca do ótimo. Os operadores genéticos para variáveis reais conseguiram um desempenho dentro das expectativas já comprovadas em (de Mendonça 2004).

Outro fator a ser observado em ambos os gráficos, é a ocorrência de *platôs* ou indícios de estagnação. Para este problema a migração conseguiu promover a diversidade genética o que propiciou o retorno a busca do ótimo.

A possibilidade da quantidade de épocas ser outro parâmetro a ser otimizado pelo algoritmo genético não ser torna necessário, pois uma análise da base de dados a ser tratada já pode ser suficiente para indicar quantas épocas serão necessárias para que se possa ter um bom treinamento.

O **RMS** observado nas tabelas anteriores mostram que **SS1** e **SS2** com taxa de aprendizado e momentum fixo já encontraram bons resultados. Agora segue-se **SS1** e **SS2** com estes parâmetros aleatórios.

Na figura 5.7 é exibido o desempenho de **SS1** em 60 gerações, 50 épocas, 4, 6 e 8 ilhas (foi escolhida a melhor rodada entre 3 rodadas executadas para cada quantidade de ilhas).

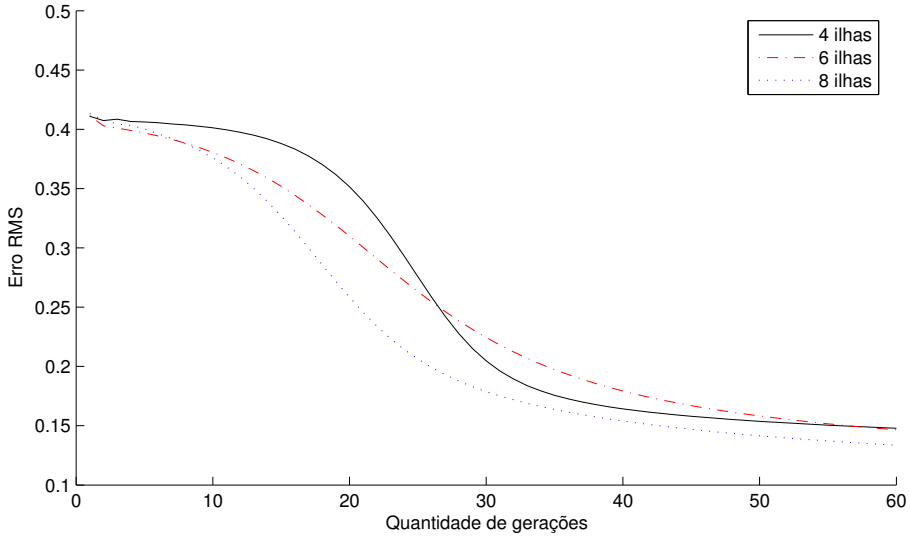

FIGURA 5.7: **SS1** com 60 gerações e 50 épocas

Na figura 5.8 é exibido o desempenho de **SS2** em 60 gerações, 100 épocas, 4, 6 e 8 ilhas

(foi escolhida a melhor rodada entre 3 rodadas executadas para cada quantidade de ilhas).

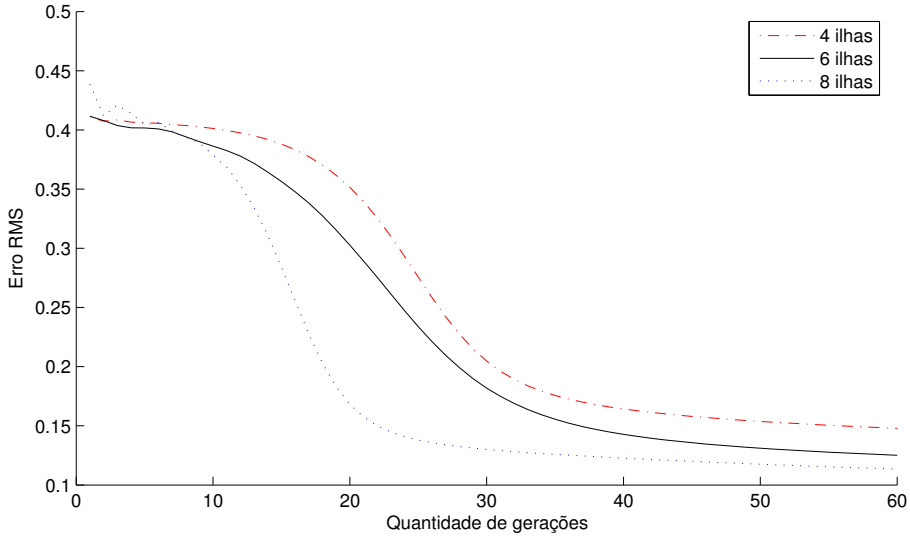

FIGURA 5.8: **SS2** com 60 gerações e 100 épocas

A tabela 5.4 contém o melhor erro **RMS** por ilhas e o respectivo indivíduo (parâmetros da rede) (Figuras 5.7 e 5.8).

Modelo	**4 ilhas**	**6 ilhas**	**8 ilhas**
SS2	RMS = 0.147895 Tx Aprend = 0.0195 Momentum = 0.5279 Atributos = 1,2,3,4 Qde neurônios = 3	RMS = 0.125116 Tx Aprend.= 0.0283 Momentum = 0.3812 Atributos = 1,3,4 Qde neurônios = 4	RMS = 0.113424 Tx Aprend.= 0.0517 Momentum = 0.7115 Atributos = 2,3,4 Qde neurônios = 4
SS1	RMS = 0.152431 Tx Aprend = 0.0513 Momentum = 0.8335 Atributos = 1,3,4 Qde neurônios = 3	RMS = 0.152355 Tx Aprend.= 0.0211 Momentum = 0.3255 Atributos = 1,2,4 Qde neurônios = 4	RMS = 0.147875 Tx Aprend.= 0.0219 Momentum = 0.5391 Atributos = 1,2,3,4 Qde neurônios = 3

TABELA 5.4: RMS por ilhas obtidos em **SS1** e **SS2**

Na figura 5.9 é exibido o desempenho de **SS1** em 60 gerações, 150 épocas, 4, 6, 8 e 10 ilhas.

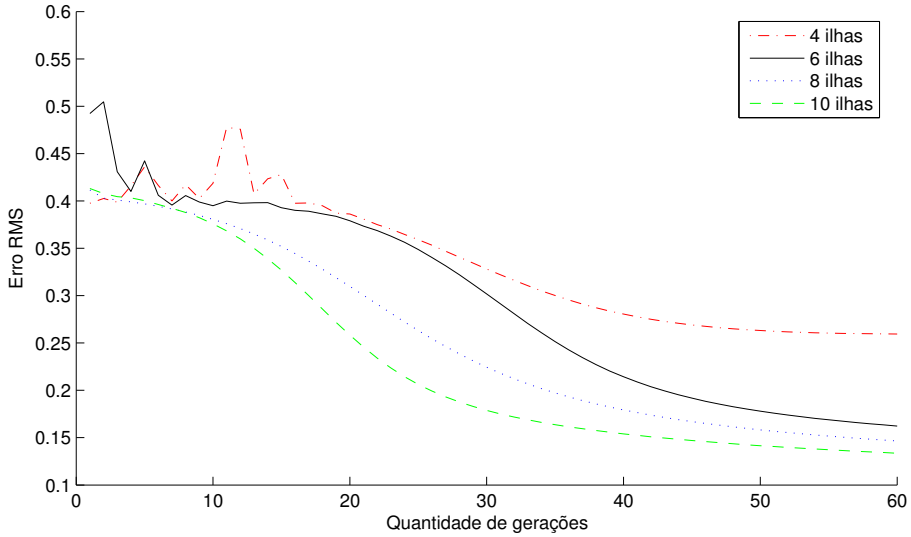

FIGURA 5.9: **SS1** com 60 gerações e 150 épocas

Na figura 5.10 é exibido o desempenho de **SS2** em 60 gerações, 150 épocas, 4, 6, 8 e 10 ilhas.

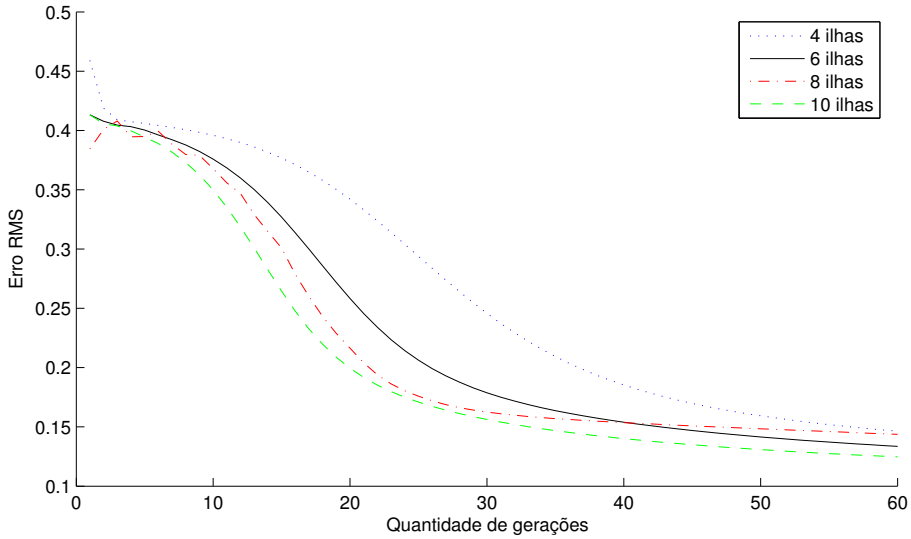

FIGURA 5.10: **SS2** com 60 gerações e 150 épocas

A tabela 5.5 comtém o melhor erro **RMS** por ilhas e o respectivo indivíduo (parâmetros da rede) (Figuras 5.9 e 5.10).

Pelos resultados que são exibidos nas tabelas anteriores o modelo **SS2** teve um desempenho dentro das expectativas, já que possibilita um maior grau de diversidade nas populações das ilhas.

O aumento da quantidade de épocas e redução da quantidade de gerações seguiu a idéia de que quanto mais épocas uma rede for treinada melhor esta pode aprender.

Modelo	4 ilhas	6 ilhas	8 ilhas	10 ilhas
SS2	RMS = 0.146367 T A = 0.0257 Mom = 0.4479 Atr = 2,3,4 N = 3	RMS = 0.143749 T A = 0.0474 Mon = 0.7655 Atr = 1,3,4 N = 3	RMS = 0.133582 T A = 0.0283 Mom = 0.5731 Atr = 2,3,4 N = 3	RMS = 0.124704 T A = 0.0387 Mom = 0.7954 Atr = 1,3,4 N = 4
SS1	RMS = 0.259407 T A = 0.0233 Mom = 0.6349 Atr = 1,2,3,4 N = 4	RMS = 0.163391 T A = 0.0339 Mom = 0.5873 Atr = 1,2,4 N = 3	RMS = 0.146513 T A = 0.0457 Mom = 0.5335 Atr = 1,2,3,4 N = 4	RMS = 0.133582 T A = 0.0641 Mon = 0.8112 Atr = 1,3,4 N = 5

TABELA 5.5: RMS por ilhas obtidos **SS1** e **SS2**

Com esta idéia, procurou-se deixar que SS1 (ou SS2) simplesmente evoluissem sobre populações que teoricamente estariam bem aptas a serem soluções, já que o indivíduo a ser avaliado representa uma configuração de rede que foi treinada durante certas épocas.

Quando se aumenta a quantidade de ilhas os indivíduos ficam com o fitness melhor pois o erro **RMS** diminui e mais ilhas permitem uma busca mais eficiente pelo espaço de soluções.

Na tabela 5.6 são exibidos os parâmetros que foram utilizados em **SS1** e **SS2**

Parâmetros	Valor
Qde. gerações	60
População	20
Cromossomo	7
Tipo de Crossover	Blend ($\alpha = 0.4$)
Taxa de crossover	0.7
Tipo de mutação	Creep
Taxa de mutação	0.01
Taxa de indiv.	30% em **SS1** 20% em **SS1**
Taxa de gerações	40% em **SS1** 30% em **SS1**

TABELA 5.6: Parâmetros utilizados em **SS1** e **SS1**

Capítulo 6

Conclusões e trabalhos futuros

Nesta dissertação foi apresentada uma implementação de AG utilizando o modelo ilha para otimizar a arquitetura (quantidade de neurônios na camada intermediária) de uma rede neural simultaneamente com a busca dos melhores atributos da base de dados.

O algoritmo genético paralelo não treinou a rede neural e sim forneceu um indicativo que um certo indivíduo com uma configuração específica pode levar o treinamento da rede ao sucesso.

Foram estudados modelos de AG paralelo e topologias para pesquisa de um modelo que obtivesse um melhor desempenho de acordo com o problema apresentado. A obtenção do melhor global ocorreu com as evoluções independentes das ilhas e com a migração de indivíduos entre elas a partir de critérios pré-estabelecidos. A troca de informações entre as ilhas se fez através da biblioteca MPI. Diferentes topologias lógicas foram analisadas para a fase de de migração de indivíduos entre as ilhas.

Com o uso de ilhas consegui-se uma melhor distribuição do domínio do problema em questão e consequentemente ocorreram análises e avaliações com total independência entre os mesmos, sendo que a migração teve por finalidade garantir a diversidade destas ilhas no que diz respeito a subpopulação de cada um. Consegui-se com este trabalho definir caminhos e perspectivas para o operador de migração tais como a escolha da topologia de migração a ser adotada.

O caso analisado (seção 5.2) sugere a viabilidade da otimização da arquitetura (rede neural) e seleção dos atributos (base de dados) com erros pequenos pelo modelo **stepping-stone**.

A utilização de operadores genéticos diferentes em cada ilha foi cogitada inicialmente, mas optou-se por realizar estes testes mais adiante após a validação.

O algoritmo será utilizado no projeto ClusterMiner, além de que serão implementados novos operadores de crossover, mutação e seleção. Nesta nova etapa pretende-se desenvolver métodos e pesquisas que possibilitem encontrar novas heurísticas sobre o operador de migração do modelo stepping-stone.

Como sugestões para futuras implementações do sistema pode-se citar entre outras:

- implementação dos modelos **island model 1** e **island model 2**;
- implementação de um objeto genérico para os operadores de migração;
- uma interface gráfica para a visualização do progresso das soluções para a evolução do treinamento das redes neurais;
- utilização de biblioteca de MPI orientadas à objeto.

Referências

Abdi, H. (1994). A neural network primer, *Journal of Biological Systems* **Vol 2**: 247–283.

Adamidis, P. (1994). Review of parallel genetic algorithms bibliography, *Technical report*, Department of Electrical and Computer Engineering - Aristotle University of Thessaloniki.

Basgalupp, M. P., Machado, M. L. M., de Souza, J. A. & da Silva, J. B. (2005). Otimização na definição da arquitetura da rede neural mlp para análise de séries temporais : Uma proposta utilizando algoritmos genéticos, *Technical report*, PUC-RS, UFSC, UFPel.

Brasil, L. M., de Azevedo, F. M. & Barreto, J. M. (1998). Uma arquitetura híbrida para sistemas especialistas, *Technical report*, Departamento de Informática e Estatística - Universidade Federal de Santa Catarina.

Brasileiro, E. V., de Oliveira Galvão, C. & Brasileiro, F. V. (2005). Otimização em tempo real de redes de escoamento de petróleo usando algoritmos genéticos, *Technical report*, Departamento de Sistemas e Computação - Departamento de Engenharia Civil - Universidade Federal de Campina Grande.

Cantú-Paz, E. (1995). A summary of research on parallel genetic algorithms, *Technical report*, Computer Science Department and Illinois Genetic Algorithms Laboratory - University of Illinois at Urbana-Champaign.

Cantú-Paz, E. (1997). A survey of parallel genetic algorithms, *Technical report*, Computer Science Department and Illinois Genetic Algorithms Laboratory - University of Illinois at Urbana-Champaign.

Cantú-Paz, E. (1998). Topologies, migration rates and multi-population parallel genetic algorithms, *Technical report*, Department of Computer Science and Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Cantú-Paz, E. & Goldberg, D. E. (1999). Efficient parallel genetic algorithms : Theory and practice, *Technical report*, University of Illinois at Urbana-Champaign.

Chapman, B. & Mehrotra, P. (1998). Openmp and hpf : Integrating two paradigms, *Europar 98*, pp. 65–658.

Corrêa, E. (2000). *Algoritmos Genéticos e Busca Tabu Aplicados ao Problema das P-Medianas*. Dissertação de M.Sc., Universidade Federal do Paraná.

Costa, M. (1999). *Data Mining em Computadores de Alto Desempenho Utilizando-se Redes Neurais*. D.Sc. thesis, COPPE-UFRJ.

da Silva, J. D. S. & Simoni, P. O. (2001). On a new migration scheme for the island model parallel genetic algorithm, *V Simpósio Brasileiro de Automação Inteligente*.

de Campos, M. M. & Saito, K. (2004). *Sistemas Inteligentes em Controle e Automação de Processos*, Editora Ciência Moderna.

de Mendonça, C. E. L. R. (2004). *Um Sistema Computacional Para Otimização Através de Algoritmos Genéticos e Redes Neurais*. D.Sc. thesis, COPPE-UFRJ.

de Oliveira, A. C. M. (2001). Algoritmos evolutivos para problemas de otimização numérica com variáveis reais, Monografia apresentada para o exame de qualificação do Curso de Computação Aplicada - Instituto Nacional de Pesquisas Espaciais.

Geist, A., Beguelin, A. & Dongarra, J. (1994). *PVM : Parallel Virtual Machine*, MIT Press.

Geist, G., Kohl, J. & Papadppoulos, P. (1996). Pvm and mpi : a comnparison of features, http://www.csm.ornl.gov/pvm/PVMvsMPI.ps.

Goldberg, D. (1989). *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company.

Goodman, E. D., Lin, S. & Punch, W. F. (1994). Coarse-grain parallel genetic algorithms: Categorization and new approach, *VI IEEE Symposium on Parallel and Distributed Processing*.

Gordon, S. V. & Whitley, D. (1993). Serial and parallel genetic algorithms as function optimizers, *V International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 177–183.

Grama, A., Gupta, A., Karypis, G. & Kumar, V. (2003). *Introduction to Parallel Computing*, 2nd edn, Addison Wesley.

Gropp, W., Lusk, E. & Skjellum, E. (1994). *Using MPI : Portable Parallel Programming with the Message Passing Interface*, Cambridge,MA,MIT.

Haykin, S. (2001). *Redes Neurais : princípios e prática*, 2nd edn, Bookman.

http://www.mpi-forum.org (2005).

Ignácio, A. A. V. & Filho, V. J. M. F. (2002). Mpi : Uma ferramenta para implementação paralela, *Pesquisa Operacional* **vol. 22**(n. 1): p. 105–116.

Jong, K. A. D. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD thesis, Universidade of Michigan.

Jordan, H. F. & Alaghband, G. (2003). *Fundamentals of Parallel Processing*, Prentice Hall.

Lacerda, E. & Carvalho, A. (1999). Introdução aos algoritmos genéticos, *Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação*, Vol. 2, pp. 51–126.

Levine, D. (1994). A parallel genetic algorithm for set partitioning problem, *Technical report*, ANL-94/23 Argonne National Laboratory.

Loureiro, S. M., Margoto, L. R., Varejão, F. & Queiroga, R. M. (2005). Um mecanismo automático para busca de parâmetros de técnicas de classificação utilizando algoritmos genéticos, *Technical report*, Centro Tecnológico - Departamento de Informática - Universidade Federal do Espírito Santo.

Medeiros, F. L. L. (2003). *Algoritmo Genético Hibrido como um Método de Busca de Estados Estacionários de Sistemas Dinâmicos*. Dissertação de M.Sc., INPE-CAP.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.

Michalewicz, Z., Logan, T. D. & Swaminathan, S. (1998). Evolutionary operators for continuous convex parameter spaces, *Technical report*, University of North Carolina.

Omer, B. (1995). Genetic algorithms for neural network training on transputers, Departament of Computing Science, University of Newcastle.

Pacheco, P. (1996). *Parallel Programming with MPI*, Morgan Kaufmann Publishers.

Pacheco, M. A. C., Vellasco, M. M. R., Lopes, C. H. P. & Passos, E. P. L. (1998). Extração de regras de associação em bases de dados por algoritmos genéticos, *Technical report*, Departamento de Engenharia Elétrica - PUC/RJ.

R.K.Ursem (2002). Diversity-guided evolutionary algorithms, *Proceedings of PPSN 2002*, pp. 462–471.

Sambatti, S. B., de Campos Velho, H. F., Leonardo D. Chiwiacowsky, S. S. & Preto, A. J. (2005). Epidemic genetic algorithm for solving inverse problems : Parallel approach, *Technical report*, Laboratório Associado de Computação e Matemática Aplicada - Instituto Nacional de Pesquisas Espaciais.

Sarma, G., Zacharia, T. & Miles, D. (1998). Using hpf for parallel programming, *Computers and Mathematics with Applications*, Vol. 35, Elsevier Science, pp. 41–57.

Schwehm, M. (1996). Parallel population models for genetic algorithms.

Soares, G. L. (1997). *Algoritmos Genéticos : Estudo, Novas Técnicas e Aplicações*. Dissertação de M.Sc., Programa de Pós-Graduação em Engenharia Elétrica - UFMG.

Soch, M. & Tvrdik, P. (1998). Performance evaluation of message passing libraries on ibm sp-2, *Technical report*, Department of Computer Science and Engineering, Czech Technical University.

Spears, W. M., Jong, K. A. D., Back, T., Fogel, D. B. & Garis, H. (1993). An overview of evolutionary computation, *European Conference on Machine Learning*, pp. 442–459.

Tanomaru, J. (1995). Motivação, fundamentos e aplicações de algoritmos genéticos, *II Congresso Brasileiro de Redes Neurais - III Escola de Redes Neurais*.

Wright, A. H. (1990). Genetic algorithms for real parameter optimization., *FOGA*, pp. 205–218.

Zanchettin, C. & Ludermir, T. B. (2005). Técnica híbrida de otimização para pesos e arquiteturas de redes neurais artificiais, *Technical report*, Universidade Federal de Pernambuco.