MODELOS PARALELOS DE ALGORÍTMOS GENÉTICOS NO APRENDIZADO DE REDES NEURONAIS ARTIFICIAIS

Mauricio Gonzalo Solar Fuentes

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 DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profa. Sueli B. Teixeira Mendes, Ph.D.
(Presidente)

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Luis Alfredo V. de Carvalho, D.Sc.

Prof. Ricardo José Machado, D.Sc.

Prof. Jayme Luiz Szwarcfiter, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
AGOSTO DE 1992

SOLAR F., MAURICIO G.

Modelos Paralelos de Algoritmos Genéticos no Aprendizado de Redes Neuronais Artificiais

[Rio de Janeiro] 1992

XIV, 152 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia em Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1.- Inteligência Artificial

2.- Sistemas Conexionistas

3.- Algoritmos Genéticos

4.- Processamento Paralelo

I. COPPE/UFRJ. II. Título (série).

Pode-se, na ciência aprender o máximo
estudando aquilo que parece ser o mínimo.

A minha filha NICOLE.
A minha esposa MARIELA.

AGRADECIMENTOS

Muitas pessoas contribuiram para a realização deste trabalho, seja pela amizade ou pelas trocas de idéias.

Primeiro, desejo expressar meus agradecimentos ao PROGRAMA DE ENGENHARIA DE SISTEMAS E COMPUTAÇÃO/COPPE da UFRJ, e em especial ao POVO DO BRASIL (através do CNPq e CAPES) pela possibilidade de realizar a pós-graduação.

Desejo expressar meus mais sinceros agradecimentos a Professora SUELI MENDES, pela sua confiança, apoio e orientação acadêmica.

Aos meus professores da COPPE, que me ensinaram o caminho da pesquisa, onde há muita coisa para ser feita.

Aos professores da banca pelos seus significativos comentários e contribuições, em especial ao Dr. RICARDO MACHADO.

Aos meus colegas da COPPE pela convivência e amizade que é indispensável no estudo: Adelina, Cláudias, Clicia, Cristina, Hércules, Hugo, Inês, Lorena, Márcia, Max, Monica, Rui, Victor, etc.

A minha família: MARIELA pelo seu amor e sacrifício de ter suportado estes anos de estudo; e NICOLE, quem me deu a energía de continuar meus estudos e pesquisa.

Aos meus pais e irmãos, pelo amor.

Em geral, a todas as pessoas que me apoiaram e ajudaram desinteressadamente: Manolo, Maria Angélica, à turma do Río Rugby. Obrigado.

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

MODELOS PARALELOS DE ALGORITMOS GENÉTICOS NO APRENDIZADO DE REDES NEURONAIS ARTIFICIAIS

Mauricio Gonzalo Solar Fuentes

Agosto de 1992

Orientador: Ph.D. Sueli B. T. Mendes
Programa: Programa de Engenharia de Sistemas e Computação

Esta tese apresenta um modelo paralelo de um Algoritmo Genético Conexionista (AGC) adequado para tarefas que operam num meio ambiente dinâmico. O modelo é baseado nos mecanismos de busca dos Algoritmos Genéticos (AGs) e em resultados provenientes das Redes Neuronais Artificiais (RNAs). As propriedades e os parâmetros do AGC permitem ao sistema se adaptar às mudanças do meio ambiente, respondendo adequadamente às situações para as quais não foi treinado. O AG é capaz de superar as limitações de aprendizado automático das RNAs, permitindo alterar o conhecimento na medida que o meio ambiente muda. O AGC apresenta propriedades desejáveis de ambos os campos, ou seja capacidade de generalização das RNAs e capacidade de adaptação dos AGs.

As limitações de usar as RNAs num meio ambiente dinâmico, como no caso do controle de processos, é produto da dificuldade destas em capturar todos os padrões na fase de aprendizado. Daqui surge a necessidade de dispor de um sistema que possa se adaptar facilmente às mudanças do meio ambiente.

Várias propostas têm sido apresentadas como uma solução a este problema. Uma delas é o uso dos Algoritmos Genéticos, os quais têm mostrado ser uma boa alternativa pela sua flexibilidade na adaptação. Mas, estes AGs "puros" não possuem um mecanismo de aprendizado eficiente que oriente a busca, dado que a forma de avaliar as soluções (função objetivo) é muito complexa de obter.

A solução proposta neste trabalho é um modelo paralelo de Algoritmo Genético Conexionista que combina ambas as técnicas, onde é possível modificar os pesos das RNAs de tal forma que o conhecimento armazenado nas conexões, seja alterado na

medida que o meio ambiente muda. Isto permite herdar as propriedades desejáveis de ambas as técnicas.

O AGC opera com uma família (população) de RNAs (indivíduos) da mesma arquitetura, cujo conhecimento armazenado nos pesos é diferente. Cada processador contem uma parcela da população (população local). O processamento é feito localmente em cada processador em forma independente do resto da população, permitindo aumentar a velocidade de processamento.

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

PARALLEL MODELS TO GENETIC ALGORITHMS IN LEARNING OF ARTIFICIAL NEURAL NETWORKS

Mauricio Gonzalo Solar Fuentes

August of 1992

Thesis Supervisor: Ph.D. Sueli B. T. Mendes
Department: Programa de Engenharia de Sistemas e Computação

This thesis presents a parallel model to the combination of Artificial Neural Networks (ANNs) with Genetic Algorithms (GAs), producing a flexible and powerful learning paradigm, called Conexionist Genetic Algorithm (CGA).

In Computer Science, where sequential algorithms are usually paralleled by means of tricks and contortions, it is altogether ironical that Genetic Algorithms, which are inherently parallel, are performed serially, using the same unnatural, non-elegant tricks. This thesis presents a parallel model for a Conexionist Genetic Algorithm, mapped naturally on a parallel machine (Intel's iPSC and INMOS Transputer Network), with good results in terms of communication, process and response time. The model is based on GA's search mechanisms and on results from ANNs. The properties and parameters of CGA allow for the system to adapt to changes in the environment, so that it responds adequaly in situations for which it was not trained. The GA is able to overcome the limitations of automatic learning of the ANNs, so that the knowledge can be modified to the extend of changes in the environment. The CGA has desirable properties of both fields, i.e., a capacity for generalisation of ANNs and a capacity for adaptation of GAs. The CGA operates with a family (population) of ANNs (individuals) with a fixed architecture, but with different knowledge stored in their weigths.

Starting from the hypothesis of building blocks, the cromosomic representation of the ANNs guarantees the search for the optimum of the solution. The set of synapses having an effect on the learning of a single neuron on the output layer forms a block, which must be contiguous with the other blocks having an influence on the same neuron.

The results obtained with the parallel model of automatic learning show that the initial

population size is independent of the length of the cromosome, and that the processing response is appropiate for real time applications in dynamic environments.

ÍNDICE

ÍNDICE DE FIGURAS

NOMENCLATURAS

AG: Algoritmo Genético
AGC: Algoritmo Genético Conexionista
BC: Base de Conhecimento
IA: Inteligência Artificial
L: Comprimento do Cromossoma
N: Tamanho da População
n_i: neurônio, nó ou elemento de processamento
RNA: Rede Neuronal Artificial
SE: Sistema Especialista
TDS: Transputer Development System
TP: Tamanho da População

CAPÍTULO I

I.1.- INTRODUÇÃO

O controle de processos foi sempre de grande interesse para os pesquisadores teóricos e práticos. Estes sistemas são tipicamente caraterizados pelo seu parcial entendimento da dinâmica não linear, falta de conhecimento dos verdadeiros parâmetros do sistema, subsistemas de ruído e um grande montante de incertezas na interação entre o processo e o meio ambiente. Nestas últimas décadas, os teóricos do controle clássico e moderno têm oferecido novas e sofisticadas técnicas para cobrir efetivamente algumas destas dificuldades. No entanto, um tratamento matemático rigoroso é restrito usualmente por um conjunto de suposições. Para garantir a aplicação de certas teorias, deve-se verificar continuamente as suposições que foram colocadas, o que do ponto de vista prático é geralmente impossível. Esta é uma das razões pelas quais muitas das indústrias de controle de processos nos últimos 40 anos têm baseado suas decisões de controle automático nas ações da simples estrutura do controlador PID (Proportional-Integral-Derivative), os quais não resolvem todos os problemas do controle adaptativo [LEE et alii, 1992].

Existem diversos sistemas desenvolvidos com técnicas de Inteligência Artificial (IA) que conseguem superar algumas desvantagens dos sistemas convencionais, mas estes sistemas com técnicas de IA convencional têm algumas desvantagens que é possível superar com as RNAs.

Neste trabalho é discutido o uso de Redes Neuronais Artificiais (RNAs) como uma alternativa nos sistemas de controle. A aplicação das RNAs no controle vêm dos anos 1962. WIDROW (1987), da Universidade de Stanford, mostrou uma RNA que aprendeu com sucesso o algoritmo de controle para balancear um cabo de vassoura ("broomstick"). Desde então, muitos outros problemas de controle tem sido resolvidos com técnicas das RNAs.

As RNAs têm a característica de se adaptar bem às situações que consegue associar com algum padrão que armazenou na fase de aprendizado. O problema surge quando não consegue identificar ou classificar um padrão de entrada. Então, a RNA não responde adequadamente ou simplesmente entrega uma resposta errada. Uma solução é parar a operação da RNA e entrar na fase de aprendizado novamente, para ensinar a nova situação e logo após continuar com a fase de operação (ou "recall"). Isto não é aceitável para problemas onde este tipo de situações ocorre continuamente, como no

caso de uma aplicação que interage com um meio ambiente dinâmico, sendo um exemplo o controle de processos, onde a interação do ser humano não é uma solução para o controle automático.

Pela necessidade de desenvolver um modelo matemático rigoroso para sistemas físicos, é essencial ter em mente que o comportamento de sistemas de controle no mundo real é extremamente difícil, senão impossível para descrever com um modelo matemático exato. Neste caso, o papel dos Algoritmos Genéticos (AGs) é importante para compensar a falta de informação para obter um ótimo comportamento do aprendizado dinâmico em RNAs.

As contribuções deste trabalho são:

(1) Apresentar uma solução para o problema de aprendizado dinâmico. A solução proposta faz uso dos AGs para guiar o aprendizado em RNAs. Ou seja, um AG que opera sobre uma população de RNAs da mesma arquitetura, realizando as operações genéticas para encontrar o ótimo numa situação determinada. O modelo de Algoritmo Genético Conexionista (AGC) é inspirado no modelo apresentado por MONTANA e DAVIS (1989), mas a técnica de codificação é diferente;

(2) Estudar e propor alguns modelos paralelos para o AGC, e selecionar um modelo viável e simples de mapear numa máquina paralela real, como por exemplo um Hipercubo baseado em Transputers;

(3) Verificar o comportamento do modelo selecionado, e determinar parâmetros tais como tamanho do cromossoma e tamanho da população;

(4) Mostrar que a solução proposta consegue superar o problema do tratamento seqüencial (VON NEUMANN), acelerando o processamento do modelo, e permitindo o seu uso em controle de processos em tempo real.

I.2.- CONTEÚDO DO TRABALHO

O Capítulo II apresenta uma discussão blibliográfica das RNAs e as vantagens e desvantagens de serem usadas no controle de processos. A discusão blibliográfica mostrada no Capítulo III apresenta os AGs como uma solução ao aprendizado automático em situações onde o meio ambiente é dinâmico. No Capítulo IV

se mostra a solução deste problema, onde se detalha um modelo de um Algoritmo Genético Conexionista (AGC) baseado numa RNA "feedforward", mas podendo usar outros modelos de RNAs, tais como aprendizado competitivo ("clusters"). Para estes casos se propõem as diferentes funções objetivos que devem ser usadas.

Como foi dito, a operação em tempo real do modelo proposto só é possível com o uso da computação paralela, para o qual no Capítulo V mostram-se vários modelos paralelos do AGC com multiprocessamento, fazendo uma avaliação do tempo de execução, tempo de comunicação e quantidade de processadores necessários nos diversos modelos. No fim apresenta-se uma implementação em arquiteturas reais, tais como o hipercubo iPSC da Intel e uma Rede Hipercúbica de Transputers.

No fim, no Capítulo VI, mostram-se as conclusões, os comentários do modelo, e trabalho futuro a realizar.

CAPÍTULO II
APRENDIZADO NAS REDES NEURONAIS ARTIFICIAIS

Este capítulo objetiva mostrar os problemas apresentados pelas Redes Neuronais Artificiais (RNAs) para implementar sistemas de controle de processos em tempo real. Estes sistemas precisam ter adaptabilidade dinâmica para responder às mudanças do meio ambiente no qual devem operar continuamente. Para isso é necessário dispor de habilidade para aprender por si só, com a experiência e a capacidade de tratar com analogias. São estudados os algoritmos de aprendizado adaptativo, e são indicadas as condições exigidas pelos sistemas de controle, com o objetivo de ter um aprendizado adaptativo em tempo real, de tal forma que o sistema não entre em crise e possa operar continuamente sob qualquer circunstância.

A principal abordagem do capítulo é mostrar os problemas que devem ser resolvidos no desenvolvimento de uma RNA aplicada ao controle de processos. A descrição faz ênfase na seleção dos paradigmas conexionistas que devem ser escolhidos na implementação de uma RNA. No estudo conclui-se que a RNA com aprendizado competitivo é uma boa solução para ser usada em conjunto com os Algoritmos Genéticos (AGs) com o objetivo de obter um aprendizado adaptativo e incremental.

II.1.- INTRODUÇÃO

Na teoria, os neurocontroladores podem ser considerados para serem aplicados em qualquer área onde os engenheiros deseje construir circuitos para controlar ações (ou decisões) de uma forma inteligente [WERBOS, 1989b]. O exemplo mais obvio está na robótica [KAWATO et alii, 1988; MILLER III, 1989], mas existem outras aplicações interessantes envolvendo controle de processo de manufatura [RANGWALA e DORNFELD, 1989], controle de plantas [WERBOS, 1989a], veículos autônomos [TOURETZKI e POMERLEAU, 1988], aviação, propulsão, teleoperação [RAUCH e WINARSKE, 1988; HIRAMATSU, 1989], ajuda aos desabilitados, incluindo síntese de voz [SEJNOWSKI e ROSENBERG, 1988], braços artificiais [ECKMILLER, 1988], ajuda na mobilização [KUPERSTEIN e RUBINSTEIN, 1988], e também próteses neuronais. Muitas aplicações das RNA têm sido desenvolvidas nas diferentes grandes indústrias, incluindo finanças, manufatura, defesa, aeroespaço e controle. Isto indica que se tem suficientes resultados empíricos para apresentar as principais desvantagens das RNAs na área de controle de processos, e propor uma metodologia de desenvolvimento preliminar com o objetivo de identificar as tarefas importantes de um processo que

geralmente carece de estrutura e organização. Os benefícios óbvios são o controle de custo, incrementando a precisão e a consistência, e o uso eficiente dos recursos, incrementando a confiança gerencial, e uma maior satisfação do usuário.

Muitas das metodologias de desenvolvimento propostas para as RNAs são adaptadas dos sistemas de software convencionais e mais recentemente dos Sistemas Especialistas (SE). Em contraste, o desenvolvimento das RNA tem uma ênfase mais forte na experimentação e desenvolvimento simultâneo de múltiplas alternativas, um refinamento iterativo dos parâmetros da RNA, o reprojeto do problema e a sua reformulação, e o início de soluções gerais que permitem estreitar o conjunto das aproximações possíveis [BAILEY e THOMPSON, 1990].

Os modelos tradicionais para resolver sistemas de controle possuem certas dificuldades, tais como [SUDDARTH, SUTTON e HOLDEN, 1988]: (1) o modelo deve-se definir completamente para todas as circunstâncias nas quais se espera que opere; (2) o modelo não pode se adaptar bem num meio ambiente que muda além da definição original; e (3) o modelo deve ser apresentado para o computador numa forma de lógica completamente explícita.

Muitas destas desvantagens podem ser superadas com o treinamento por exemplos numa RNA, obtendo os seguintes benefícios: (1) é facil obtê-la rapidamente, e sua qualidade pode melhorar com o tempo; (2) é adaptativo ao meio ambiente para o qual não foi especificamente programado; e (3) aceita conhecimento humano na forma de lógica explícita e na forma de treinamento baseado na experiência [ECKMILLER, 1988]. Finalmente, se estes modelos são implementados usando um hardware específico para RNAs, estes sistemas podem atingir altas velocidades de operação.

Este trabalho mostra uma comparação entre os SEs e as RNAs na implementação de sistemas de controle em tempo real, fazendo ênfase nas características principais que estes sistemas precisam ter para operar num meio ambiente dinâmico. A seguir são definidos os conceitos básicos das RNAs, e mostram-se os métodos e algoritmos de aprendizado para este propósito. Depois, são apresentadas as considerações mais importantes nos sistemas em tempo real. No fim, apresentam-se algumas respostas às questões: É viável o uso de RNAs?; Quais paradigmas usar?; Como especificar os paradigmas do projeto?; Que algoritmo de aprendizado usar?; Como implementar um modelo de RNA?. Como conclusão se propõe o uso dos AGs para guiar o aprendizado dinâmico em RNAs em tempo real.

II.2.- TEMPO REAL: RNAs v/s SISTEMAS ESPECIALISTAS

O conceito de controle inteligente tem como necessidade dispor de componentes inteligentes que operem em tempo real. Os três principais esquemas que têm demonstrado ter capacidade para o controle inteligente são:

(1) os SE como elementos adaptativos num sistema de controle;
(2) a realização de cálculos "fuzzy" como elementos produtores de decisões num sistema de controle; e
(3) as RNAs como elementos de compensação.

A seguir são mostradas as características dos sistemas de controle inteligentes e as vantagens e desvantagens de serem implementados com SEs.

II.2.1.- Sistemas Especialistas e o Controle de Processos

Numa aplicação real, em centros de controle, pode-se identificar as características ideais de um sistema de processamento em tempo real, resumidas a seguir:

.capacidade de identificar situações de alteração;
.capacidade de alterar dinamicamente a prioridade dos eventos;
.capacidade de reconhecer e tratar os eventos esperados;
.capacidade de suprimir a enunciação de determinados eventos (filtros);
.capacidade de reconhecimento automático de eventos;
.capacidade de eliminação automática de eventos;
.tempo de resposta;
.facilidade de tradução de processos heurísticos da operação em tempo real;
.consideração do tempo de continuidade de uma situação de alarme.

Em todo sistema de controle em tempo real, as três últimas características determinam a eficiência e o desempenho do sistema para responder às situações reais.

Os objetivos de um sistema de controle em tempo real são: oferecer mais segurança e agilidade no processo de tomada de decisão. Quando não existe agilidade na tomada de decisões o sistema pode entrar em crise, onde o operador deve enfrentar situações novas que dependem do seu julgamento; e também enfrentar muitos dados e não a informação que esses dados podem conter.

Normalmente, esses sistemas operam em três estados (Normal, Emergência e Restaurativo) e muitos eventos indicam transições entre estes estados. O operador deve realizar certas operações com os eventos, tais como: reconhecer ou eliminar um evento ou um bloco de eventos, onde as operações por bloco podem incorrer em erros ou falta de informação.

Na ocorrência de um evento o operador deve:

.perceber a anunciação do evento;

.entender o seu conteúdo;

.analisar as conseqüências do evento;

.procurar determinar a seqüência de ocorrências que levou à condição de emergência;

.determinar uma ação a ser efetuada para normalizar a situação, se necessário.

Os sistemas tradicionais de controle, os quais operam com grande incerteza, tipicamente dependem da intervenção humana para funcionar apropriadamente. Contudo, a intervenção humana é inaceitável em muitas aplicações em tempo real para o qual devem ser desenvolvidas técnicas automáticas para manipular a incerteza. Numa aplicação típica de controle, tal como robótica, muitos problemas são enfrentados como genéricos no projeto de controladores para grandes sistemas dinâmicos. Alguns problemas destas aplicações são [BAVARIAN, 1988]:

.sobrecarregar o sensor de dados, o que pode decorrer de dados redundantes ou dados especializados raramente usados pelo sistema;

.fusão de dados de sensor, e mapeamento dentro do laço de realimentação do controle;

.sistema não suficientemente forte para manipular grandes excursões dos parâmetros;

.sistema incapaz de manipular informação heurística;

.sistemas que não podem ser usados para controle em tempo real de alta velocidade porque eles consomem muito tempo nos cálculos de IA;

.sistemas onde a alternativa de sensor não foi resolvida [ANDERSON, 1988; FALK, 1988].

Os novos conceitos e as ferramentas novas procuram corrigir as deficiências anteriores ou aumentar o escopo das funções. Os SE resolvem em parte esses problemas, e com o objetivo de mostrar a viabilidade de aplicar técnicas de IA, existem várias aplicações de SE para centros de controle, que têm demonstrado ter melhor desempenho que os sistemas usados tradicionalmente [SOLAR et alii, 1989].

II.2.2.- Problemas dos Sistemas Especialistas

Embora um SE possa, em tese, chegar mesmo a superar o seu correspondente humano num determinado assunto, algumas deficiências intrínsecas normalmente impedem que se alcance tal desempenho, tais como [WILLIAMSON, 1985]:

.**inabilidade** para aprender por si só com a **experiência**;
.**incapacidade** de tratar com **analogias**;
.falta de intuição;
.os especialistas humanos sabem avaliar em que ocasiões as regras devem ser quebradas.

Estas deficiências dos SE podem ser superadas por outro tipo de sistemas, as RNAs, que possuem as seguintes características gerais:

.**adaptabilidade dinâmica**: que é a capacidade de ajuste no tratamento dos eventos de modo a acompanhar mudanças no estado do sistema de controle. Tais mudanças podem ser:
- ocorrência de um evento;
- existência de uma combinação de eventos;
- ocorrência de um determinado número de eventos num período de tempo;
- declaração explícita do operador.

.aprender com a **experiência**;
.capacidade de tratar com **analogias**;

Por outro lado, temos que o desenvolvimento de um SE envolve tarefas muito complexas tais como:

.**aquisição do conhecimento**, extrair a experiência do especialista na solução de problemas (num domínio restrito);
.**representação do conhecimento**, estruturar o conhecimento nas formas clássicas de representação (quadros, regras, redes semânticas, etc.);
.**desenvolver mecanismos de inferência** com os quais será acessado o conhecimento ("forward" e "backward chaining", etc.).

A implementação destas tarefas envolve a programação nas linguagens de IA, tais como LISP ou Prolog, que permitem obter protótipos de SE em alguns meses (6 até 12 meses). Estas linguagens se caracterizam pela sua facilidade para manipular

símbolos e resolver os problemas de IA, mas sua velocidade de operação é lenta. Isto motivou os pesquisadores a desenvolverem técnicas de paralelismo para estas linguagens, permitindo uma maior velocidade na execução dos programas. Contudo, conseguiram-se alguns avanços neste aspecto, mas mesmo assim, o tempo consumido em programar um SE é muito elevado, mesmo usando as ferramentas disponíveis e desenvolvidas especialmente para este propósito.

OBERMEIER e BARRON (1989), afirmam que o desenvolvimento de uma configuração de RNA é feito só numa fração do tempo que seria usado para configurar e construir um SE para a mesma aplicação.

II.2.3.- RNAs e o Controle de Processos

Na IA existem duas tendências de pesquisa: a simbolista e a conexionista. A simbolista possui alguns métodos de aprendizado já implementados em máquinas seqüenciais, tais como SOAR [LAIRD et alii, 1987], mas com muitas limitações como mostra MICHALSKI et alii (1986). A conexionista está atualmente sendo muito pesquisada, pelas grandes vantagens demonstradas nas várias implementações, não só em aprendizado [LAIRD, 1988; HINTON et alii, 1984], como também em visão [FUKUSHIMA et alii, 1983; MILLER III, 1989], controle de robôs [KAWATO et alii, 1988), reconhecimento de padrões e associação [WIDROW et alia, 1988], memória associativa e endereçável por conteúdo [KOSKO, 1988], produção de voz [SEJNOWSKI e ROSENBERG, 1986], formação de categorias [PENG et alia, 1989] e otimização global [RANGWALA et alia, 1989].

As RNAs de auto-processamento, segundo REGGIA et alia (1988), são fundamentalmente diferentes dos modelos de processamento de informação mais tradicionais em IA e ciência cognitiva em pelo menos dois aspectos essenciais. Primeiro, elas são de auto-processamento. Os modelos de processamento de informação tradicionais consistem tipicamente de uma estrutura de dados passiva (redes associativas, regras de produção, etc.), a qual é manipulada por um procedimento ativo externo (mecanismo de inferência, intérprete de regras, etc.). Nas RNAs, os nós e conexões numa RNA de auto-processamento são agentes ativos de processamento e tipicamente não existe um agente ativo externo que opere sobre ela. Segundo, as RNAs de auto-processamento mostram um comportamento global do sistema que emerge das interações locais concorrentes dos seus múltiplos componentes. O processo externo que manipula as estruturas de dados definidas nos modelos convencionais em IA e ciência

cognitiva têm tipicamente um acesso global para toda a rede ou conjunto de regras, e parte do processamento é forte e explicitamente seqüencial (por exemplo, resolução de conflitos nos sistemas baseados em regras).

Esta noção de inteligência como uma propriedade emergente é um princípio fundamental que é aproveitado com as RNAs no problema de controle. Neste caso, o controle é considerado como um problema de reconhecimento de padrões, onde os padrões a serem reconhecidos são "mudanças" de sinais que são mapeadas em sinais de "ação" para um sistema de funcionamento específico [HECHT-NIELSEN, 1987].

Um controlador inteligente baseado em RNAs pode reconhecer e isolar padrões de mudanças em tempo real e aprender da experiência para reconhecer mudanças mais facilmente, mesmo com dados incompletos [BAVARIAN, 1988].

II.3.- CONCEITOS BÁSICOS DAS RNAs

O cérebro é um computador incrivelmente poderoso. O cortex possue mais de 10^{10} neurônios, cada um ligado com outras centenas. É provável que todo o conhecimento humano esteja armazenado nos pesos das conexões entre os neurônios, os quais realizam todas as complexas tarefas em tempo real [HINTON, 1985].

A velocidade de operação dos computadores modernos é um milhão de vezes mais rápido que a dos neurônios, os quais estão na ordem dos milisegundos. Isto significa que o cérebro, um dispositivo composto de elementos neuronais, é capaz de resolver problemas difíceis de visão e processamento de linguagem em algumas centenas de milisegundos (500 mseg). Sendo que o melhor programa de IA para estas tarefas está muito longe desta velocidade [FELDMAN et alii, 1988].

É obvio que o cérebro é um dispositivo alta e massiçamente paralelo, e também distribuído. Isto gerou uma polêmica entre os pesquisadores de IA, pois de um lado haviam os que se preocupavam não somente com o que o cérebro é capaz de fazer, mas também como o faz [ROSENBLATT, 1962] (conexionistas), e de outro lado, pesquisadores que tentavam simular o cérebro sem se preocuparem realmente com o "como" [CARBONEL, 1989; HAYES-ROTH et alii, 1978] (simbolistas).

É importante mencionar que os conexionistas tentam imitar a forma na qual funciona o cérebro baseados nos fundamentos biológicos do neurônio, sendo que os

simbolistas tentam simular as funções do cérebro modelando sistemas que não se baseiam nestes fundamentos.

O estudo das RNAs indica que a implementação destes sistemas conexionistas é feito em duas fases: a primeira, corresponde ao aprendizado da RNA, chamada treinamento; e a segunda, ao uso do conhecimento da RNA naquilo para o qual foi treinada, a operação ou recuperação ("recall").

A fase de treinamento simplifica muito a tarefa de aprendizado, dado que a RNA aprende por si mesma, não sendo necessário realizar as tarefas de aquisição e representação do conhecimento. Dai surgiram vários algoritmos de aprendizado e várias técnicas com diferentes arquiteturas de RNAs.

II.3.1.- Definição dos Conceitos

Dado que usualmente na literatura são definidos os elementos conceitualmente similares com diferente simbologia, a seguir mostra-se a definição dos termos que serão usados ao longo do texto para eliminar qualquer dúvida referente ao significado de cada elemento (figura II.1).

n_i: elemento ou unidade de processamento, nó ou neurônio i;

in_j: entrada j do neurônio;

$a_i(t)$: estado de ativação do neurônio n_i no instante t;

$A(t)=(a_1(t),a_2(t),...,a_N(t))$: padrão de atividade da RNA no tempo t;

$o_i(t)$: resposta ou saída no tempo t em função de seu estado de ativação $a_i(t)$;

$O(t)$: vetor de saída $(o_1(t),o_2(t),...,o_N(t))$;

f_i: regra de saída que relaciona o estado de saída em função do estado de ativação, $o_i(t)=f_i(a_i(t))$. Geralmente a função f_i é determinada pelo limiar ("threshold") do n_i.

w_{ij}: eficiência sináptica entre o axônio do neurônio n_j e um dendrito do neurônio n_i;

W: padrão de conexão (matriz de conexão), que é o conjunto de valores w_{ij};

g_i: regra de propagação que integra as entradas recebidas pelo neurônio n_i num impulso total de entrada $u_i(t)=g_i(in_i(t),W)$. Classicamente é usada a regra $g_i=\Sigma(w_{ij}\cdot in_j(t))$; com $(1\leq j\leq N)$;

h_i: regra de ativação que fornece o novo estado de ativação $a_i(t+1)$ no instante t+1, $a_i(t+1)=h_i(a_i(t),u_i(t))$. Classicamente h_i é a função sigmoidal.

c_r: resposta correta dada externamente por um professor, se o aprendizado é supervisionado (ver seção II.4 adiante).

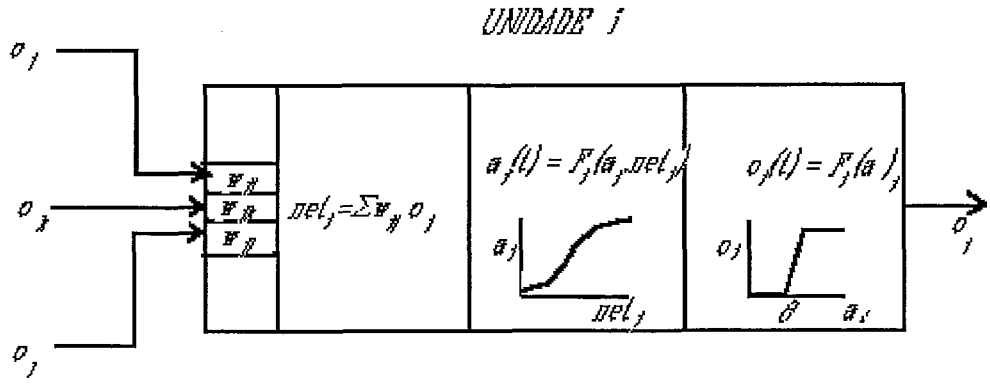

Fig. II.1: Neurônio Elementar

A seleção destes parâmetros determina o desempenho da RNA para o qual na seqüência tenta-se dar uma resposta às seguintes questões: Como a RNA obtém a informação?; Quão rápido o sistema processa a informação?; Qual é a capacidade de armazenamento de uma RNA?; Quantos padrões podem ser armazenados numa RNA com N elementos?. Tudu isto tem relação com funções de energia, convergência, estabilidade e pontos de equilíbrio, os quais serão tratados ao longo deste Capítulo.

II.3.2.- Modelos de Neurônios

Muitos trabalhos em computação neuronal atuais baseiam-se no neurônio de McCULLOCH e PITTS (1943) (muitas vezes acrescido por alguma regra de aprendizado), ou no modelo integrador (conhecido como o neurônio de HOPFIELD (1982) se está conectado com outros neurônios numa forma simétrica) [ARBIB, 1989].

O modelo de McCULLOCH e PITTS é um modelo síncrono e discreto no tempo. A saída $o_i(t)$ de cada neurônio i pode ter o valor 0 ("off", não disparado) ou 1 ("on", disparado). A linha de saída (axônio) se ramifica repetidamente, terminando em pontos de união (sinapses) com outros neurônios ou efetuadores. O neurônio em si pode receber sinapses de outros neurônios ou receptores. Denota-se a saída do neurônio como $o_i(t)$, as N entradas $in_1(t),...,in_N(t)$, correspondentes às sinapses com pesos $w_{i1},...,w_{ij},...,w_{iN}$, e um limiar θ. A sinapse é excitatória se o seu peso é positivo, e inibitória se é negativo. Neste modelo o neurônio não possui memória das entradas, como nas regras de aprendizado dos modelos mais recentes que usam a memória para ajustar os pesos. Com os pesos correntes, e as entradas no tempo t é determinada a saída no tempo t+1 de acordo a seguinte regra:

$$o_i(t+1)=E_i(t)- k\cdot I_i(t) \qquad \begin{cases} a_i(t+1) = 1 \text{ sse } in_i(t) \geq \theta_i \\ a_i(t+1) = 0 \text{ sse } in_i(t) < \theta_i \end{cases}$$

onde $E_i(t)$ é o número de in_i em "on", $I_i(t)$ é o número de in_i em "off", e k é uma constante maior que 0.

A grande desvantagem deste modelo é que não possui um mecanismo de aprendizado, portanto as relações de Entrada-Saída devem ser projetadas manualmente. Mas, pode ser acrescido com regras para ajustar os pesos sinápticos, tais como a regra de HEBB (1949) para aprendizado correlacional, o aprendizado competitivo de RUMELHART e ZIPSER (1985) e GROSSBERG (1987), ou a regra de aprendizado de Perceptron [ROSENBLATT, 1962], etc.

No outro modelo, o neurônio pode ser modelado como um integrador, com tempo contínuo, e com duas quantidades que são usadas para descrever o comportamento do neurônio, sendo estes o potencial da membrana m(t) e uma razão de disparo M(t). M(t), na forma simplificada, é uma função de m(t), M(t)=f(m(t)), onde f pode ser a função de saída de passo (binária), a função de saída linear (gradual), ou a função de saída sigmoidal $k/(1+\exp^{(-m(t)/\theta)})$ (contínua), que cresce desde 0 para o seu máximo k na medida em que m(t) cresce.

Em forma resumida pode-se dizer que existem basicamente três regras que determinam o comportamento do neurônio:

(1) regra de propagação: as entradas recebidas pelo neurônio i são propagadas através das conexões com suas unidades vizinhas para outro processamento;
(2) regra de ativação: o nível de atividade a_i do neurônio é calculado a partir das entradas com pesos das unidades que chegam ao neurônio;
(3) regra de saída: a saída o_i do neurônio é calculada baseada no nível de atividade do neurônio.

As RNAs são sistemas de grande escala compostos destes elementos simples de processamento que pode ser linear ou não (neurônios). Cada neurônio é caracterizado pelo estado, que é uma lista de entradas com peso, e uma equação que governa o comportamento dinâmico do neurônio. Os seguintes são os dois modelos mais apropriados que são usadas na área de controle:

(1) A RNA binária assíncrona (memória associativa de HOPFIELD) [HOPFIELD, 1982], que é composta de N neurônios, cada um deles caracterizado pelos parâmetros a_i, o qual é o estado de atividade (binário) do neurônio i, w_{ij}, o peso da conexão do neurônio j para o neurônio i, e θ_i, que é o limiar do neurônio i. O estado binário (bipolar) do neurônio é +1 ou -1. A dinâmica da RNA é tal que os neurônios atualizam os seus estados de acordo com a seguinte regra, onde sgn(x) é o sinal algébrico de x:

$$a_i = \mathrm{sgn}(\Sigma(w_{ij} \cdot in_j - \theta_i))$$

O estado da RNA a cada instante do tempo é descrito pelos N elementos binários a_i. Existe uma função de Lyapunov na matriz simétrica w_{ij}, a qual garante que todos os pontos de atração no sistema de equações diferenciais (a_i) sejam pontos fixos. Estes pontos fixos servem para armazenar os padrões desejados (tais como comandos para o controlador). Quando a RNA é iniciada a partir de um padrão inicial que está próximo (por exemplo, em termos da distância de Hamming[1]) de um determinado ponto fixo, o estado do sistema converge rapidamente para este ponto fixo. Um padrão do meio ambiente que cair dentro da região de atração associada a um ponto fixo é chamado de recuperação ("recall") do padrão armazenado nesse ponto fixo.

Como foi indicado acima, a informação que uma RNA armazena e recupera está no seu estado estável. Isto é, a arquitetura neuronal pode servir como uma memória endereçável por conteúdo. A informação é colocada na memória através de processos ou adaptação. Estes processos atribuem valores aos parâmetros dinâmicos que fazem certos padrões serem os mínimos locais da RNA, limitando-os a uma região de atração.

(2) Os modelos da familia COHEN e GROSSBERG (1983) são uma generalização do modelo de HOPFIELD. Neste modelo, os estados dos neurônios são um número real e não binário. O estado do neurônio muda continuamente de acordo com um conjunto de equações diferenciais ordinárias. Para o neurônio i, o estado u_i é governado pela seguinte equação diferencial, onde $a_i(u_i)$ é o termo que guia o controle, $b_i(u_i)$ é o termo auto-excitatório, e, para cada j, o termo $w_{ij} \cdot d_j(u_j)$ é uma entrada com peso inibitório do neurônio j para o neurônio i.

$$du_i/dt = a_i(u_i) \cdot [b_i(u_i) - \Sigma\{w_{ij} \cdot d_j(u_j)\}]$$

[1] **Distância Hamming**: o mínimo número de trocas de bits que pode produzir um caractere para obter um outro caractere.

As propriedades de estabilidade do modelo estão em COHEN e GROSSBERG (1983). Como no modelo anterior, a principal aplicação da RNA é como memória associativa com informação armazenada nos diferentes mínimos locais do sistema.

II.4.- TREINAMENTO E APRENDIZADO NAS RNAs

O aprendizado é um aspecto intrínseco da inteligência e uma necessidade para a adaptação às mudanças do meio ambiente. Por esta razão, este tópico é um dos mais estudados pelos pesquisadores em RNAs. Os procedimentos de aprendizado conexionista podem ser divididos em três grandes classes: supervisionado [SEJNOWSKI e ROSENBERG, 1987], não-supervisionado [CARPENTER e GROSSBERG, 1988] e auto-supervisionado [KOHONEN, 1984].

O aprendizado supervisionado requer um professor para especificar a saída desejada, corrigindo as respostas erradas da RNA. É aplicável só quando o comportamento desejado da RNA é conhecido a priori, o qual nem sempre ocorre.

No aprendizado não-supervisionado, os dados são simplesmente introduzidos sem intervenção humana. Este processo permite à RNA fazer suas próprias representações internas, construindo modelos internos que capturam as regularidades da entrada sem receber informação adicional.

O aprendizado auto-supervisionado (ou de reforçamento) ocorre quando a RNA se monitora a si mesma, corrigindo os erros de interpretação dos dados pela realimentação através da RNA. Existem várias formas de transformar um tipo de procedimento de aprendizado nos outros tipos.

Quando uma arquitetura de RNA em particular tem a capacidade teórica de modelar uma função desejada, permanece a questão de quão boa é a regra de aprendizado usada para produzir uma representação apropriada [HINTON, 1985 e 1989]. Enquanto as representações de longo prazo estão distribuídas através dos pesos das conexões entre as unidades, o aprendizado de novas informações é efetuado pela modificação destes pesos de tal forma que melhore o mapeamento desejado entre os vetores de entrada e saída. Esta função de mapeamento é efetuada pelo ajuste dos pesos de acordo com regras de aprendizado bem definidas. MATHEUS e HOHENSEE (1987) identificam três classes gerais de regras. A primeira, chamada de regras correlacionais,

que determinam o ajuste individual dos pesos somente na base dos níveis de atividade entre as unidades conectadas; a segunda classe é o conjunto de regras de correção de erros, a qual conta com uma realimentação externa do valor desejado das unidades de saída. A terceira classe é composta pelas regras de aprendizado não-supervisionado, na qual o aprendizado ocorre como uma auto-adaptação às regularidades detetadas no espaço de entrada, sem uma realimentação direta de um professor.

As RNAs podem ser classificadas de acordo com o tipo de memória associativa que elas realizam. A memória associativa obtém a informação completa a partir de informação parcial, e tem um papel fundamental em reconhecimento de padrões e aplicações de processamento da informação.

Duas variantes de memórias associativas são de especial interesse nas aplicações de RNAs: memórias auto-associativas e memórias hetero-associativas. A memória auto-associativa faz um mapeamento de parcelas de dados em si mesmos, memorizando informação específica. Os paradigmas conexionistas auto-associativo mais comuns são a RNA de HOPFIELD (1982) e a teoria de ressonância adaptativa (ART) [CARPENTER e GROSSBERG, 1988], embora muitos pesquisadores tenham explorado paradigmas similares. Este tipo de memória é usado quando se quer reconstruir um padrão parcial ou um padrão contendo erros na sua forma original, ou para otimizar certos problemas da pesquisa operacional. Exemplos de aplicações auto-associativas são reconhecimento de caracteres, o ato de completar imagens, reconhecimento efetuado pela retina e reconstrução de sinais.

A memória hetero-associativa faz um mapeamento de um conjunto de padrões para um outro padrão. Os padrões, as classificações e os números reais são geralmente armazenados como saídas de RNAs hetero-associativas. Os paradigmas conexionistas que pertencem às memórias hetero-associativas incluem Memória Associativa Bidirecional (BAM) [KOSKO, 1988], o esquema de mapeamento do KOHONEN (1984), e alguns paradigmas estatísticos. Exemplos de aplicações hetero-associativas são classificação de padrões, monitoração e controle de processos.

A regra correlacional é frequentemente encontrada nas RNAs auto-associativas que efetuam aprendizado de rota de estados específicos de memória [HOPFIELD, 1982; KOHONEN, 1984].

A regra de correção de erro é predominantemente usada pelo aprendizado supervisionado a partir de exemplos enquanto ela usa conhecimento de como o sistema

se desvia dos exemplos do professor para modificar os pesos das conexões [ROSENBLATT, 1962; RUMELHART et alii, 1986; SEJNOWSKI e ROSENBERG, 1987].

A regra do aprendizado não-supervisionado é usada em situações onde a RNA deve aprender a se auto-adaptar ao meio ambiente sem receber realimentação específica do professor [KOHONEN, 1984].

II.4.1.- Regras Correlacionais

A inspiração de muitas regras de aprendizado nas RNAs é baseada nos sistemas biológicos estudados por HEBB (1949), indicando que a eficiência das sinapses entre dois neurônios é incrementada quando o nível de disparo da atividade entre eles está correlacionado. O resultado é que a conexão da unidade i com a unidade j é reforçada cada vez que o disparo da unidade i contribui no disparo da unidade j, ou seja, quando as unidades i e j são excitadas simultanemente, o peso da sua conexão se incrementa proporcionalmente ao produto dos seus níveis de atividade. A regra é definida formalmente como:

$$\Delta w_{ij} = o_i \cdot o_j$$

onde Δw_{ij} é a variação do peso da conexão da unidade j para a unidade i, e o_i e o_j são os níveis de saída das respectivas unidades. A mudança dos pesos é afetada somente pelos níveis de atividade das unidades conectadas, e não de qualquer conhecimento do rendimento global do sistema, ou seja é local.

A RNA de HOPFIELD

As RNAs auto-associativas usam tipicamente as regras correlacionais, onde o exemplo clássico é a RNA de HOPFIELD, a qual usa uma combinação das regras de aprendizado HEBB/anti-HEBB. A RNA de Hopfield é uma RNA auto-associativa totalmente conectada na qual o aprendizado é efetuado pelo armazenamento de "estados de memória" na matriz de pesos da RNA. Para armazenar um estado particular, é aplicada a seguinte regra: se duas unidades estão ambas em "on" ou ambas em "off", então incrementa os pesos das suas conexões; se uma unidade está em "on" e a outra em "off" (anti-HEBB), então decrementa seus pesos. Assumindo valores de saída binários, a equação formal para esta regra é:

$$\Delta w_{ij} = (2o_i-1)\cdot(2o_j-1)$$

O uso desta regra permite armazenar vários estados de memórias ao mesmo tempo na mesma RNA. Quando a RNA é executada, esta pode se estabelecer num estado final, o qual será o mais próximo do estado inicial (ótimo local). O número de estados de memórias permitidos na RNA de HOPFIELD, onde a degradação não é séria ocorre com menos de 0.15·N, onde N é o número total de unidades na RNA. Se os estados de memórias são representados por vetores ortogonais, então cada memória individual será perfeitamente recuperável (supondo que a capacidade da memória não foi excedida). Na prática, os estados desejados a serem memorizados são não-ortogonais, resultando num decremento significativo na capacidade de armazenamento.

II.4.2.- Regras de Correção de Erros

Esta classe de regras é a mais usada popularmente nas implementações atuais [HINTON et alii, 1984; RUMELHART et alii, 1986; SEJNOWSKI e ROSENBERG, 1987]. O esquema geral nesta regra é deixar a RNA produzir sua própria saída em resposta à entrada, depois do qual um professor externo apresenta ao sistema a saída correta ou desejada (supervisionado). Se a RNA responde corretamente, não é necessário modificar nenhum peso (embora alguns sistemas usam a informação para reforçar o resultado correto). Se existir um erro na saída da RNA, então a diferença entre a saída desejada e a saída atingida é usada para guiar as modificações apropriadas dos pesos. Dado que este método se esforça para atingir uma solução global para o problema de representar a função tomando pequenos passos na direção do melhor local maior, ele é equivalente à busca do gradiente descendente no espaço de representações possíveis.

Perceptron

Um exemplo simples da regra de correção de erros é o procedimento de convergência do Perceptron [ROSENBLATT, 1962], um modelo síncrono com atualização determinística. Este esquema de aprendizado é como segue: (1) se a saída de uma unidade está correta, então nada muda; (2) se a sua saída está em "on" quando deveria estar em "off", então os pesos das unidades de entrada em "on" são decrementados por um montante fixo; e (3) se a saída está em "off" quando deveria ser "on", então os pesos das unidades de entrada em "on" são incrementados por um

montante fixo. De acordo com o teorema de convergência do Perceptron [MINSKY e PAPERT, 1969], esta regra de aprendizado garante a convergência para a representação correta, se e somente se a função desejada é linearmente separável (embora esta restrição seja inerente à arquitetura de uma RNA com uma única camada e não é restrição da regra de aprendizado). É uma máquina de decisão linear, que requer um grande número de neurônios para resolver problemas complexos. Este aprendizado supervisionado pode ajustar os pesos das conexões da RNA com ajuda da regra Delta.

Regra Delta ou Regra Widrow-Hoff com Professor

A técnica geral da regra Delta é a mesma do Perceptron, onde o peso das conexões é acrescentado ou decrementado de acordo com a modificação que leve a resposta da unidade mais perto do valor desejado. Na regra Delta, no entanto, o montante das mudanças varia de acordo com o grau de discrepância entre o valor desejado e os estados atingidos. Formalmente, a regra define as modificações do peso entre duas unidades segundo:

$$\Delta w_{ij} = \eta \cdot \delta_i \cdot o_j$$

onde η é um parâmetro global que controla a razão do aprendizado e δ_i representa a diferença entre o valor desejado da unidade i e o seu valor atual:

$$\delta_i = (o_i^{desejado} - o_i^{atual})$$

A vantagem deste esquema sobre a regra de aprendizado do Perceptron é que o montante da mudança é proporcional ao grau de erro (no sistema binário δ_i é sempre 1 ou 0). Portanto, para pequenos erros que ocorrem quando a representação está quase correta, só mudanças menores são feitas. Para grandes erros, os quais presumivelmente indicam representações longe do valor desejado, as mudanças dos pesos são correspondentemente maiores. Esta regra, como o procedimento de convergência do Perceptron, também garante a convergência para uma solução, caso esta exista. Mas, a regra Delta só trabalha com RNAs "feedforward" de duas camadas (sem camadas escondidas), e portanto está também restrita para mapear representações de funções linearmente separáveis. Uma solução desejada é estender esta técnica para RNAs multi-camadas, mas o problema de acrescentar camadas escondidas é saber os valores de ativação corretos para as unidades escondidas usados no cálculo de $\delta_{escondida}$. Este dilema é o clássico problema da atribuição de créditos: o mundo externo não especifica o estado correto das unidades escondidas, e conseqüentemente a regra de aprendizado não possui uma forma direta para determinar quando aquelas unidades estão contribuindo positiva ou negativamente ao resultado.

Regra Delta Generalizada

A regra Delta generalizada [RUMELHART et alii, 1986] é uma extensão da regra de aprendizado Delta, a qual supera o problema de atribuição de crédito e habilita o aprendizado efetivo em RNAs multi-camadas. Esta regra trabalha propagando o erro das unidades de saída para trás através das unidades escondidas. O problema de atribuição de crédito é resolvido calculando o erro de uma unidade escondida como a soma proporcional (relativa aos pesos) dos erros das unidades que a alimentam. Portanto, trabalhando para trás a partir do erro observado nas unidades escondidas, os erros das camadas subjacentes (anteriores) das unidades escondidas podem ser calculadas recursivamente.

A atualização dos pesos seguem a regra Delta padrão:

$$\Delta w_{ij} = \eta \cdot \delta_i \cdot o_j$$

No caso generalizado, no entanto, o valor de δ_i difere dependendo do tipo de unidade. Para unidades de saída, δ_i é similar ao da regra Delta padrão, diferindo unicamente pelo fator da derivada da função de ativação (razão pela qual a regra Delta generalizada requer que o sistema use uma função de ativação que tenha uma derivada finita contínua). As unidades escondidas, no entanto, são manipuladas numa forma mais complicada, envolvendo ambas a derivada da função de ativação e a soma dos δ_i das unidades do nível superior seguinte, multiplicado pelo peso respectivo. Formalmente:

$$\delta_i = f'(u_i) \ \Sigma(\delta_k \cdot w_{ki})$$

onde $f'(u_i)$ é a derivada da função de ativação com respeito à sua entrada total, u_i, e os δ_k são os erros das unidades que recebem entradas das unidades i. Esta função recursiva de δ's, é implementada na prática primeiro calculando o valor de ativação como se mostrou acima, computando as mudanças das unidades de saída, e então usando os δ's das unidades de saída para calcular os valores dos erros para o nível anterior das unidades escondidas, e assim por diante para todas as unidades.

A regra de aprendizado Delta generalizada é muito importante porque provê um método efetivo para o aprendizado em RNA multi-camadas. Esta regra tem sido usada com sucesso para representar funções não-linearmente separáveis, tais como OU-Exclusivo, paridade e simetria [RUMELHART e McCLELLAND, 1986]. Resultados impresionantes obtiveram-se no sistema NETtalk [SEJNOWSKI e ROSENBERG, 1987], o qual aprende a ler textos em inglês. NETtalk começa sem nenhum

conhecimento prévio da pronúncia das palavras e gradualmente aprende a ler palavras do texto verbalmente através da exposição da fala livre e palavras num dicionário. Uma desvantagem desta regra (como outros modelos de gradiente descendente) é que não escala bem, o qual funciona bem para um número relativamente pequeno de unidades, e é exponencialmente custoso na medida que as conexões são acrescentadas em algumas centenas. A regra Delta generalizada requer espaço adicional para armazenar os estados de todas as unidades na decisão da RNA, como também armazenar todos os δ's usados na propagação do erro para trás.

A Máquina de BOLTZMANN

Uma alternativa da regra de aprendizado Delta generalizada é encontrada na máquina de BOLTZMANN [HINTON et alii, 1984], a qual é uma RNA com uma regra de aprendizado baseada na mecânica estatística de Resfriamento Simulado ("simulated annealing") [KIRKPATRICK et alii, 1983]. O objetivo deste sistema é construir uma representação interna do mundo externo como uma distribuição de probabilidade dos eventos. Aprende relações entre as entradas e as saídas que refletem sua ocorrência relativa no meio ambiente.

A regra de aprendizado de BOLTZMANN é muito complexa, envolvendo uma coleção de informação estatística dos estados de equilibrio segurado ("clamped") e não-segurado ("unclamped"). Um evento, representado como um par de vetores de entrada/saída, é apresentado ao sistema e as unidades externas são seguradas (mantêm-se no seu estado inicial). As unidades escondidas da RNA obtém permissão para atingir o equilíbrio através do "simulated annealing", e então são colecionadas as estatísticas da freqüência dos pares de unidades em "on" num determinado número de ciclos. O sistema é então executado livremente, não-segurado, e de novo a estatística das atividades correlacionadas são registradas no equilíbrio. Os dois valores estatísticos para cada par de unidades são subtraídos e o sinal resultante destes números é usado para acrescentar ou decrementar os pares dos pesos (simétrico) por um montante fixo (apesar de que esta regra de aprendizado seja do tipo de HEBB no seu uso das atividades correlacionadas, é classificada como correção de erro, porque faz uso da diferença entre as atividades correlacionadas esperada e atual para determinar em que direção atualizar os pesos). Este processo é repetido várias vezes no conjunto de eventos representativos. Depois do treinamento, quando o sistema recebe um vetor de entrada, a máquina de BOLTZMANN reproduz como resposta a saída mais provável baseada na sua representação interna generalizada do mundo.

A regra de aprendizado da máquina de BOLTZMANN provê um método efetivo para aprender os pesos das unidades escondidas e, como tal, é capaz de aprender funções de mapeamento complexas e linearmente não-separáveis. A maior crítica da regra de aprendizado de BOLTZMANN (e do "simulated annealing", em geral) é que ela é extremamente lenta. Um requerimento inerente aos processos de "simulated annealing" é que o "resfriamento" do sistema na medida que atinge o equilíbrio deve ser feito lentamente, de outra forma, poderá somente alcançar o mínimo local. Resumindo, o procedimento de aprendizado de BOLTZMANN é basicamente um método de gradiente descendente e portanto, como outros sistemas que aplicam gradiente descendente, requer a exposição de um grande número de exemplos para aprender as relações desejadas. Segundo HINTON et alia (1986), para um problema relativamente simples com 8 unidades em total (um decodificador de 4 unidades para 4 unidades), o tempo de aprendizado pode exceder os 1800 ciclos de aprendizado, com cada ciclo requerendo de 40 avaliações de estados de ativação, o qual é inviável para aplicar em controle de processos.

II.4.3.- Aprendizado Não-Supervisionado

Esta forma de aprendizado envolve auto-organização num meio ambiente completamente não-supervisionado. Nesta categoria de regras de aprendizado, a atenção não é focalizada em como as unidades de saída atuais casam com uma determinada saída desejada externamente, e sim que os pesos sejam adaptados para refletir a distribuição dos eventos observados. O esquema do aprendizado competitivo proposto por RUMELHART e ZIPSER (1985) oferece um esquema orientado para este objetivo. Neste modelo, as unidades da RNA são colocadas num "pool" determinado, no qual as respostas das unidades por um padrão de entrada é inicialmente aleatória. Na medida que são apresentados os padrões, as unidades dentro do "pool" são permitidas competirem entre si ao direito de responder. Aquela unidade que responder mais fortemente ao padrão é designada como vencedora. Então, os pesos da RNA são ajustados de tal forma que a resposta da unidade vencedora é reforçada, fazendo-a mais provável de identificar-se com aquela qualidade particular da entrada no futuro. O resultado do aprendizado competitivo é que com o tempo, as unidades individuais envolvidas dentro de esquemas detectores diferentes podem ser usadas para classificar o conjunto de padrões de entrada.

Esquema de Mapeamento Auto-Organizado

O esquema de mapeamento auto-organizado de KOHONEN (1984) demonstra uma forma similar de aprendizado não-supervisionado, cujo objetivo é prover algum discernimento em como os caminhos sensoriais, que acabam no córtex cerebral, muitas vezes mapeiam topologicamente aspectos de estímulos físicos. Embora isto possa ser explicado em parte pelo padrão de crescimento do axônio pré-determinado geneticamente pela codificação. Isto parece indicar que esta organização dinâmica vem do aprendizado em si. KOHONEN provê uma possível explicação propondo regras de aprendizado que favorecem este tipo de auto-organização dirigida topologicamente. Neste modelo, os pesos da RNA começam a representar gradualmente qualidades ou características das entradas de tal forma que as unidades são fechadas topologicamente para que a RNA responda similarmente aos exemplos similares de entrada. Resumindo, as respostas da RNA como um todo distribuem-se de tal forma que cobre mais efetivamente a distribuição de probabilidades do espaço característico.

No mapeamento de KOHONEN, a camada das unidades de entrada é totalmente conectada com uma camada superior de unidades de saída. Para cada unidade de saída é atribuído um vetor de peso, W, que corresponde aos pesos do vetor de entrada para essa unidade. Este vetor representa o ponto no espaço característico para o qual a unidade está melhor ajustada, por exemplo, à entrada mais sensível. Um vetor de entrada, que é idêntico ou similar ao vetor de pesos da unidade, excita essa unidade que responde melhor à entrada. Quando um vetor de entrada in de dimensão N é apresentado durante o aprendizado, se calcula uma medida de distância, δ_j, para cada unidade, indicando o quão perto os seus pesos estão da entrada. A fórmula desta medida é:

$$\delta_j = \Sigma(in_i - w_{ij})^2, \quad 0 \leq i \leq N\text{-}1$$

A unidade com menor distância é selecionada como vencedora, e os seus vetores de peso são modificados para fazer a unidade mais sensível ao vetor de entrada in. Associada à cada unidade de saída existe uma vizinhança das unidades mais próximas, definida por um raio que decresce lentamente de tamanho no tempo. Quando uma unidade vence a competição e tem os seus pesos modificados, o vetor de pesos das unidades da vizinhança imediata são modificados similarmente, colocando-a mais perto do padrão de entrada. Formalmente, a regra de modificação dos pesos é:

$$\Delta w_{ij} = \begin{cases} a(o_i - w_{ij}), \forall i \in N_c \\ 0, \quad \forall i \notin N_c \end{cases}$$

onde N_c é a vizinhança da unidade vencedora com raio c.

O resultado desta regra de aprendizado é que com o tempo, as unidades específicas de saída dentro da RNA aprendem a responder a qualidades particulares da entrada. Resumindo, as unidades da vizinhança são também levadas a uma resposta similar, fazendo com que as unidades da RNA se ordenem topologicamente com respeito às características da entrada. Isto é, o sistema de aprendizado não-supervisionado deste tipo não aprende no sentido de achar uma representação de uma função de mapeamento particular. Antes, a intenção deste sistema é ramificar o espaço de eventos em regiões de alta atividade de entrada, e atribuir unidades para responderem seletivamente a estas regiões.

Regra de Aprendizado Competitivo sem Professor

Esta regra desenvolvida por VON DER MARLSBURG (1973), GROSSBERG (1976), FUKUSHIMA (1975), FUKUSHIMA e MIYAKI (1982), e KOHONEN (1982), permite às unidades competirem de alguma forma pelo direito de responderem a um subconjunto dado de entrada. As unidades numa camada dada são divididas em conjuntos de "clusters" não-superpostos. Cada unidade num "cluster" inibe cada uma das outras unidades desse "cluster". Os "clusters" são do tipo "vencedor leva tudo", tal que aquela unidade que recebe a maior entrada atinge seu valor máximo (1), enquanto todas as outras unidades no "cluster" são levadas para seu valor mínimo (0).

O montante total fixo de peso para a unidade j é:

$$\Sigma(w_{ij}) = i$$

Uma unidade aprende passando peso das suas linhas de entrada inativas para as linhas ativas.

$$\Delta w_{ij} = \begin{cases} 0 & \text{se a unidade j perde a competição do elemento k} \\ g\cdot(C_{ik}/n_k) - g\cdot w_{ij} & \text{se a unidade j vence a competição do elemento k} \end{cases}$$

onde C_{ik} é 1 se no padrão de estímulo S_k, a unidade i na camada anterior está ativa, e 0 em outro caso. n_k é o número de unidades ativas no padrão S_k, g é uma constante de proporcionalidade.

GROSSBERG (1976) desenvolveu a seguinte regra similar:

$$\Delta w_{ij} = \eta \cdot a_i(t) \cdot [O_i(t) - w_{ij}]$$

II.5.- CONSIDERAÇÕES DAS APLICAÇÕES DAS RNAs EM TEMPO REAL

Existem no mínimo cinco paradigmas básicos usados em neurocontrole. Muitos destes paradigmas podem-se considerar como uma extensão da teoria de controle para problemas novos, onde são necessário a aproximação e o processamento distribuído. Os cinco paradigmas são [WERBOS, 1989b]: controle supervisionado, dinâmica inversa, estabilidade do sistema, "backpropagation" através do tempo e adaptação crítica e aprendizado reforçado. Na seqüência são detalhados cada um destes paradigmas.

II.5.1. Controle Supervisionado

No reconhecimento de padrões, a saída do sistema usualmente representa uma classificação correta dos padrões de entrada. No neurocontrole, as entradas podem ser um conjunto de leituras de sensores, enquanto as saídas são vetores de ações desejadas. Existem várias formas de obter a saída desejada, por exemplo, ela pode vir da monitoração do movimento de um humano que esta efetuando a tarefa desejada [RUMELHART et alia, 1986].

WIDROW (1989), que desenvolveu praticamente o primeiro neurocontrolador, comparou o esquema de controle supervisionado usado pela Fuji no Laboratório de Robôs com um SE. O humano que provê as ações desejadas, está agindo essencialmente como um professor que transfere a sua experiência de uma forma que é impossível de fazer com palavras.

Um elemento básico de aprendizado supervisionado é o "backpropagation", o qual é bom para alguns problemas. No entanto, alguns trabalhos [WERBOS, 1989b] sugerem que um módulo que combina "backpropagation" generalizado com certas características de memória associativa permitem um melhor desempenho e maior flexibilidade. Existem muitas outras opções similares, apropriadas para alguns problemas, tais como ART [CARPENTER e GROSSBERG, 1988], o método de KOHONEN (1984), MADALINE [BARTO, SUTTON e ANDERSON, 1983], BAM

[KOSKO, 1988], CMAC, etc., mostrados no trabalho de CARPENTER (1989). Em neurocontrole são usados aqueles módulos e não são produzidos, e sim tenta-se usar os recursos disponíveis para desenvolver sistemas maiores, sem entrar nos detalhes dos projetos.

II.5.2. Dinâmica Inversa

As propriedades dinâmicas de uma RNA, segundo GALLANT (1988), são que um modelo conexionista deve especificar quando uma unidade computa um valor novo de ativação e quando a mudança da saída dessa unidade deve ser feita. Em alguns modelos, as unidades são visitadas numa ordem fixa, onde cada unidade reavalia e faz as mudanças das suas ativações antes que as próximas unidades sejam visitadas. Em outros modelos, todas as unidades computam os seus valores de ativação simultaneamente e então fazem as mudanças das unidades de saída simultaneamente. E outros modelos computam o novo valor de ativação de uma unidade escolhida aleatoriamente, e logo muda imediatamente a saída antes que qualquer outra unidade compute seu novo valor de ativação. No modelo paralelo é importante a sincronização destes eventos para permitir uma evolução correta da RNA.

Na dinâmica inversa, o estado observado X(t) de um sistema supõe-se que é uma função das ações atuais u(t) e do estado anterior do sistema, por exemplo, X(t)=F(u(t),X(t-1)). Não é necessário conhecer a função F, mas é importante que tenha uma função inversa com respeito a u(t), ou seja, que possa resolver u(t) como uma função de X(t), para qualquer X(t-1).

A fase de treinamento ou adaptação é usualmente diferente das aplicações comuns de RNAs treinadas (dada que pode ser feito em tempo real). Na fase de adaptação, são dados exemplos de X(t) que resultaram de u(t) atuais obtidos em algumas experiências. É usado o aprendizado supervisionado para adaptar a RNA de tal forma que u(t)=H(X(t),X(t-1)), onde H é a função inversa de F.

Na fase de operação, a RNA é usada numa forma diferente. Dada uma trajetoria desejada $X_d(t)$, a RNA H tem como entradas $X_d(t)$ e X(t-1), e dá como saída um vetor de controle u, o qual guia o sistema a casar com a trajetória desejada. MILLER III (1989) e GUEZ et alii (1988) usaram uma variante deste esquema, usando diferentes módulos de aprendizado supervisionado com um preprocessador fixo.

KAWATO et alii (1988) descrevem problemas práticos que podem ocorrer quando F não é invertível. Contudo, a simplicidade deste esquema é um ponto forte a seu favor em muitas aplicações práticas.

GOLES e MARTINEZ (1990) apresentam estudos e modelos matemáticos para analisar e predizer a dinâmica de uma RNA. Com estes modelos, tenta-se ter uma predição do comportamento do sistema no futuro.

II.5.3. Estabilidade do Sistema

Na robótica e na teoria de controle, existe uma grande literatura de controle adaptativo, os quais usam a palavra "adaptativo" em forma muito diferente do significado tratado aqui. Existem projetos padrões, tais como os "reguladores auto-afinados" (aprendizado não supervisionado) e o "controle adaptativo por referência" (aprendizado supervisionado), os quais projetam principalmente a adequação e a estabilidade do sistema, e cujos objetivos secundários são a minimização do custo e a maximização do produto (controle ótimo).

Na prática, nas companhias industriais, dá-se grande importância à estabilidade, pelo fato de que a instabilidade pode ser muito custosa em certas aplicações (por exemplo, em reatores nucleares).

O trabalho de COHEN e GROSSBERG (1983) apresenta os modelos matemáticos para analisar a estabilidade de uma RNA, permitindo eliminar as oscilações da RNA.

II.5.4. "Backpropagation" através do Tempo

Supondo que se tem uma RNA que descreve ou emula a dinâmica do sistema ou da companhia que se deseja controlar, por exemplo, o modelo deveria predizer que R(t)=F(R(t-1),u(t)), onde R(t) é um vetor que descreve o estado da realidade no tempo t, u é o vetor de controle (como foi mencionado acima), e F é a função de avaliação do vetor que implementa a RNA. Em outras palavras, as entradas da RNA são R(t-1) e u(t), e a saída é uma predição para R(t). (Com "backpropagation" generalizado, F deve ser uma função diferenciável para representar uma RNA).

Outra suposição é que não existe ruído aleatório na fábrica, para todos os propósitos práticos, e que se tenta minimizar (ou maximizar) certa quantidade U(R), que é uma função diferenciável em R(t) em qualquer instante t.

Usando "backpropagation" no tempo, pode-se calcular a derivada de U com respeito a u(t) em qualquer instante (ou com respeito aos pesos de uma RNA gerando u(t)). Pela adaptação de u(t) em resposta a esta derivada, como em "backpropagation" básico, podem-se encontrar ações ou pesos ótimos.

Nas aplicações em tempo real, existem certas dificuldades no uso do "backpropagation" no tempo, mas existem várias formas de superar estas dificuldades como é mostrado em WERBOS (1989a, 1989b).

II.5.5. Adaptação Crítica e Aprendizado Reforçado

O termo "adaptação crítica" foi definido no trabalho de BARTO, SUTTON e ANDERSON (1983) como aquela família de projetos que inclui basicamente os que de alguma forma resolvem o problema com aprendizado reforçado no tempo.

Alguns projetos de sistemas adaptativos críticos requerem que a função F seja conhecida, enquanto outros não o fazem. Alguns requerem o uso de "backpropagation", mas nenhum requer o uso de "backpropagation" através de períodos múltiplos de tempo. Alguns trabalham melhor quando o número de variáveis é pequeno, mas outros são mais apropriados para problemas maiores. Todos estes sistemas requerem certas aproximações que podem ser feitas com mais precisão que com "backpropagation" no tempo, como nos casos onde o ruído pode ser abandonado por qualquer razão.

Atualmente, o caso de aprendizado reforçado é realmente um caso especial de máxima utilização. Se o valor u(t) for conhecido para cada t, então pode-se tratar este valor como um valor de entrada de sensor, por exemplo, pode-se fazer $X_1(t)=U(t)$, onde X_1 é a primeira componente do vetor X. Neste caso, pode-se definir simplesmente $U(R)=U(X)=X_1$. Em principio, é melhor trabalhar com o caso mais geral da máxima utilização, porque permite explorar o conhecimento da função U, quando este está disponível.

II.6.- RNA PARA O CONTROLE DE PROCESSOS

Este Capítulo mostra no início os fundamentos teóricos das RNAs e das aplicações em tempo real, onde pode-se concluir que existem áreas de aplicações que não são boas de serem implementadas com RNAs, tais como aqueles onde é necessário a precisão matemática, ou aplicações precisas, como aquelas em que é necessário carregar dados temporários, reportar informação, e aplicações que requerem dedução ou lógica.

Nos ítens anteriores foram colocados os problemas das RNAs existentes [SOBAJIC e PAO, 1988; CARDOZO e TALUKDAR, 1988; CHRISTIE e TALUKDAR, 1988] no tratamento de alarmes e quais as características desejáveis de um sistema de controle de processos em tempo real para operar continuamente sob qualquer circunstância, sem entrar em crise.

Apresenta-se um estudo das RNAs como uma opção para implementar um sistema de controle de processos que cumpra as características necessárias para estes sistemas. Segundo as pesquisas das aplicações feitas até hoje com as RNAs [PENG e REGGIA, 1989; RANGWALA e DORNFELD, 1989; WERBOS, 1989; ASTROM, 1989; LANG e HINTON, 1988; GALLANT, 1988; FELDMAN et alii, 1988; GUEZ et alii, 1988; HECHT-NIELSEN, 1987; BAVARIAN, 1988; PSALTIS et alii, 1988; KUPERSTEIN e RUBINSTEIN, 1989; ANDERSON, 1989; ECKMILLER, 1989; KOWATO et alii, 1988; PASSINO et alii, 1989; RAUCH e WINARSKE, 1988; MILLER III, 1989], tem-se uma base sólida para verificar o bom desempenho de algumas implementações de RNAs no controle de processos. Mas estas aplicações em tempo real não possuem a característica de adaptação num meio ambiente dinâmico.

O trabalho desenvolvido aqui usa dois modelos de RNAs para sua aplicação com AGs, um modelo baseado em aprendizado supervisionado e outro não supervisionado. A RNA com aprendizado supervisionado que será usada no desenvolvimento do trabalho é baseada em processamento "feedforward" com três camadas totalmente interconectadas, pela seguinte razão: qualquer arquitetura de RNA pode-se converter numa RNA com estas características. O modelo não-supervisionado é baseado em competição, dado que o uso de um modelo conexionista baseado em competição permite achar o ótimo global do sistema através dos ótimos locais de cada "cluster".

Usar-se-á um modelo com entradas inibitórias e excitatórias, porque permite realizar várias funções difíceis de implementar, apesar de requerer grande espaço de

memória e também ter a desvantagem de aumentar fortemente o "fan out" dos elementos, sendo este um fator muito importante para acelerar a computação (elimina processamento), especialmente com grandes RNAs [PENG & REGGIA, 1989]. Para o qual é necessário o uso de processamento paralelo.

O uso de RNAs com camadas de processamento "feedforward", sacrifica certa quantidade de paralelismo, dado que só os processadores na mesma camada podem operar em paralelo [RANGWALA & DORNFELD, 1989]. As camadas posteriores devem ser computadas serialmente. Apesar de perder algum paralelismo, esta RNA é muito popular, dado que existe um esquema de aprendizado bem sofisticado para treiná-la. A RNA típica de estrutura em camada consiste de uma camada de entrada, uma camada de saída e uma camada escondida entre as camadas de entrada e saída. O objetivo desta RNA é mapear padrões de entrada em padrões de saída. Os nós nas camadas escondidas são necessários para implementar mapeamentos não-lineares entre os padrões de entrada e saída. Uma RNA linear (cujos nós são lineares) é bem aproveitada para casos onde o conjunto de padrões de entrada é linearmente independente, mas pode-se mostrar que uma RNA multi-camada com nós lineares pode ser convertida em uma RNA equivalente de duas camadas, perdendo desta forma as vantagens das unidades escondidas. Para melhorar o aprendizado, a extração das características e habilitar a RNA para implementar mapeamentos não-lineares complexos entre os padrões de entrada e saída, é necessário usar uma função não-linear para a entrada dos nós, tal como a função sigmóide com limiar.

KUPERSTEIN & RUBINSTEIN (1989) apresentam uma teoria e um protótipo de uma RNA que aprende a coordenar os sensores de um motor a partir da sua própria experiência, chamado INFANT. A grande vantagem deste sistema é a sua tolerância a falhas quando ocorrem erros de hardware. KUPERSTEIN et alia indicam que um projeto de RNA que controla adaptativamente o motor de uma fábrica deve ter certas restrições:

(1) O projeto do controlador neuronal não deve possuir informações da fábrica ou das características do atuador. Sendo assim, ou seja independente de qualquer tipo de fábrica, este pode ser aplicado genericamente para outras fábricas;
(2) A RNA deve aprender a associação entre a imagem de um objeto e o padrão dos sinais do motor para todas as articulações do robô;
(3) A computação da RNA deve ser paralela, de tal forma que possa ser generalizada para várias articulações sem acrescentar tempo de processamento.

Neste ponto é necessário destacar que uma arquitetura de RNA tem complexidade em tempo paralelo O(1) na busca do vizinho mais perto, no qual o tempo de busca é independente do número de ítems armazenados (padrões, conhecimento, etc.). Também pode obter operações paralelas de memória associativa e operações de provas de hipótese.

O Capítulo III mostra a natureza dos Algoritmos Genéticos (AGs) [HOLLAND, 1984; GOLDBERG, 1989], detalhando suas características de auto-adaptação às mudanças do meio ambiente e sua tolerância a falhas.

O Capítulo IV mostra que o uso dos AGs [OOSTHUIZEN, 1987; MONTANA e DAVIS, 1989] como algoritmo de aprendizado de uma RNA pode apresentar melhor desempenho que os usados normalmente, dado que não é necessário treinar a RNA para usá-la. O aprendizado é feito incremental e dinamicamente em forma adaptativa e baseado na experiência do sistema para resolver os problemas. No IJCAI (1990) foi feita uma réplica mostrando que o "backpropagation" superou o AG quando teve seus parâmetros melhor sintonizados.

Uma contribuição desse trabalho consiste no desenvolvimento de um modelo paralelo de um algoritmo de aprendizado com a técnica dos AGs, para ser implementado com diferentes parâmetros nas diferentes arquiteturas de RNAs, tais como: a RNA de uma camada ou multi-camadas, ou uma RNA totalmente inter-conectada ou totalmente intra-conectada.

A principal contribuição da tese apresenta-se no CAPÍTULO V, onde se estuda e analisa o modelo em máquinas paralelas.

CAPÍTULO III

ALGORITMOS GENÉTICOS

O objetivo deste Capítulo é apresentar o modelo dos Algoritmos Genéticos (AGs), que são algoritmos de otimização que se baseiam nos mecanismos análogos aos da seleção natural da teoria de Darwin e da genética moderna. Mostram-se o seu esquema, os operadores genéticos clássicos, os parâmetros do sistema e algumas aplicações. Estes AGs são usados como sistemas de aprendizado dinâmico, chamados Sistemas de Classificação (SC).

III.1.- INTRODUÇÃO

A idéia de descobrir os mecanismos com os quais o homem aprende é a motivação da pesquisa nos sistemas adaptativos. Uma destas tentativas são os chamados Algoritmos Genéticos (AG).

O que são os AGs ? Os AGs são algoritmos de otimização que se baseiam nos mecanismos análogos aos da seleção natural da teoria de Darwin e da genética moderna. Um problema de otimização é determinar dentre inúmeras soluções possíveis aquela que é a melhor (ótima) segundo um critério pré-estabelecido (função objetivo).

Os AGs foram desenvolvidos por HOLLAND (1986) e seus colaboradores na Universidade de Michigan. O objetivo da sua pesquisa é duplo: (1) abstrair e explicar rigorosamente os processos adaptativos dos sistemas naturais; e (2) projetar software de sistemas artificiais que pudessem aproveitar estes importantes mecanismos dos sistemas naturais.

Os AGs se diferenciam dos outros métodos de otimização por quatro fatores:

(1) Não trabalham com o problema original, mas transformam-no em uma codificação usando símbolos de um alfabeto escolhido. Esta codificação deve ser em forma de uma cadeia ("string") de símbolos com o objetivo de se assemelhar à seqüência de gens existentes nos cromossomas;
(2) Procuram a solução ótima partindo simultaneamente de várias soluções iniciais, enquanto os métodos clássicos de otimização baseiam sua busca em uma única

solução a cada instante. Desta forma, os AGs possuem um paralelismo explícito que lhes permite encontrar a melhor solução dentre todas as possíveis soluções (ótimo global);

(3) Utilizam como informação básica apenas o valor da função objetivo e não suas derivadas, permitindo então a otimização de funções descontínuas. Normalmente os métodos clássicos de otimização buscam uma solução na direção oposta ao gradiente da função objetivo, conseqüentemente necessitando calcular derivadas;

(4) AGs são métodos probabilísticos que, utilizando os princípios da evolução e da genética, conseguem direcionar suas buscas de maneira mais eficiente que um processo aleatório.

Os sistemas onde é mais interessante o uso dos AGs são: sistemas cognitivos, sistemas robóticos em meios reais, sistemas ecológicos, sistemas econômicos, sistemas de controle em tempo real e outros. Tais sistemas estão imersos em meio-ambientes que estão continuamente mudando, onde esperar por uma intervenção externa é praticamente impossível. Uma única opção é o aprendizado (ou uma adaptação).

O objetivo de um sistema de aprendizado (natural ou artificial) é a expansão do seu conhecimento quando possui incerteza. Isto significa que deve generalizar sobre a experiência do seu passado. Esta experiência pode guiar as novas ações futuras só quando existirem certas regularidades no meio ambiente do sistema.

O problema de extrair regularidades (em IA) é o problema de descobrir representações úteis ou categorias. O descobrimento destas categorias é só a metade do trabalho; o sistema deve também descobrir que classe de ações são apropriadas para cada categoria.

Resumindo, um sistema de aprendizado apresenta os seguintes problemas:

(1) novos dados fluem perpetuamente no meio ambiente, geralmente ruídoso e sem relevância;
(2) as ações exigem continuidade (geralmente em tempo real);
(3) os objetivos são definidos implicitamente ou são inexatos;
(4) as exigências de seqüências longas de ações têm um ganho ou reforço esparso.

Para resolver estes problemas os sistemas de aprendizado devem:

(1) inventar categorias que descubram as regularidades relevantes aos objetivos no seu meio ambiente;
(2) usar o fluxo da informação encontrada no caminho até o objetivo para reforçar seu modelo do meio ambiente;
(3) atribuir ações apropriadas para colocar na estrutura as categorias encontradas no caminho para o objetivo.

Mudanças contínuas e sempre novas deixam pouca oportunidade para a otimização, portanto a solução está orientada mais para satisfazer do que otimizar o problema.

Muitas pesquisas em AG têm-se concentrado nas seguintes suposições:

- O meio ambiente, suas entradas e saídas podem ser representadas por uma cadeia de símbolos de um comprimento fixo sobre um alfabeto A. Este alfabeto é geralmente {0,1} (segundo HOLLAND (1975), existem algumas evidências de que o alfabeto binário é ótimo. mas CARUANA et alia (1987) mostram empiricamente que o código "gray" tem melhor desempenho que o binário. GOLDBERG (1989) apresenta alfabetos arbitrários, cujos símbolos podem representar o espaço do problema, mas sua utilidade e eficiência são resultado de um grande estudo);

- Cada ponto no espaço do problema pode ser considerado como um indivíduo representado unicamente dentro do sistema por uma cadeia gerada do alfabeto do meio ambiente. Esta cadeia (cromossoma) serve como "material genético" com posições específicas (locus) na cadeia, contendo símbolos únicos ou "tokens" (gens) tomando valores (alelos);

- Em qualquer instante t de tempo, o sistema mantém uma população P(t) de cadeias representando o conjunto atual de soluções do problema. O processo começa com uma geração aleatória ou com uma população inicial especificada no projeto;

- A única realimentação disponível para uma estratégia de adaptação é o valor da medida do desempenho do processo (ajuste). Esta realimentação é usualmente chamada realimentação de "ordem-zero"; a última informação pedida ou demandada para adaptação é uma indicação da medida do desempenho do processo adaptativo;

- O tempo é medido em intervalos de tempo discretos chamado gerações.

- Nenhuma informação a priori é necessária para uma estratégia de adaptação em relação ao espaço do problema, sendo que esta limitação pode ser relaxada.

Neste Capítulo mostram-se inicialmente os sistemas de aprendizado automático baseado em AG, chamado de Sistema de Classificação (SC), definindo os elementos básicos. Logo apresenta as características dos AGs, os operadores genéticos e as aplicações mais interessantes. A seguir é apresentado o paradigma conexionista dos AGs, o qual é detalhado no Capítulo IV.

III.2.- SISTEMAS DE CLASSIFICAÇÃO

Os Sistemas de Classificação (SC) aprendem regras para executar alguma tarefa específica. A tarefa é definida por exemplos apresentados ao sistema, e uma função objetivo determina quão bem o sistema está executando a tarefa. A função objetivo não dá respostas corretas, mas dá uma certa quantidade de reforço ou uma quantidade de punição. Cada regra tem associado um grau de adaptação que indica a sua eficiência em executar a tarefa desejada.

Segundo SAMUEL (1959), um sistema de aprendizado deve resolver dois problemas essenciais: a atribuição dos créditos e a invenção de novas regras ou predicados. Nos SCs, a atribuição dos créditos é efetuada pelo ajuste do grau de adaptação das regras usando um algoritmo, chamado "Bucket Brigade" [HOLLAND, 1986]. A invenção das regras, ou aprendizado, é efetuada pela evolução das regras usando simulação genética e seleção natural, o chamado AGs.

O SC é composto por quatro partes principais: o sistema de gerenciamento de regras e mensagens; o algoritmo de atribuição de créditos; a heurística de aprendizado; e a definição do domínio da tarefa.

Para definir o domínio da tarefa, é necessário dar uma descrição dos detectores (entrada do meio ambiente); dos efetuadores (saída para o meio ambiente); e uma função objetivo para medir a efetividade da execução do sistema.

Os detectores produzem mensagens, as quais são colocadas numa lista de mensagens com mensagens produzidas durante o ciclo prévio. A lista de mensagens é

usada pelo sistema gerenciador de mensagens e regras para casar e ativar regras, as quais produzem mensagens de saída que são interpretadas pelos efetuadores ou usadas para ativar outras regras. Os resultados dos efetuadores são julgados pela função objetivo, a qual é usada pelo algoritmo de atribuição de créditos para determinar a quantidade de reforço ou punição que deve dar às regras que ativaram os efetuadores.

A representação da tarefa é definida pela codificação dos bits nos detectores e a decodificação dos bits nos efetuadores.

III.2.1.- As Cadeias e Mensagens

Uma cadeia é uma regra com uma ação e um número arbitrário de condições, todas as quais devem casar para disparar a regra. As mensagens nos SC são cadeias de bit de comprimento fixo, onde o conhecimento representado na cadeia é determinado principalmente pelo algoritmo de codificação do detector e o algoritmo de decodificação do efetuador, e secundariamente pela representação emergente das marcas usadas nas seqüências de regras. As condições e as ações de uma cadeia são padrões de mensagens, os quais são cadeias ternárias do mesmo comprimento das mensagens. Uma posição de bit nesta cadeia é conhecida como *locus*, e o valor armazenado nesta posição de bit é conhecido como *alelo*. Os alelos nas mensagens são tomados do alfabeto binário {0,1}, enquanto o alfabeto dos alelos nas condições e ações das cadeias são tomados do alfabeto ternário {0,1,#} (ver seção III.2.3). O alelo # combina tanto com 1 quanto com 0.

III.2.2.- Algoritmos de Atribuição de Créditos

O problema de atribuir créditos é o problema de decidir, quando várias regras estão ativas a cada passo do tempo, quais destas regras ativas no tempo t são suficientes para atingir alguma saída desejada no passo t+n.

Nos SC, o crédito acumulado é representado por um grau de adaptação associado a cada cadeia. O grau de adaptação de uma cadeia é importante por dois fatores:

1.- O grau de adaptação determina em parte quais cadeias serão ativados em um determinado passo do tempo, e assim controla o comportamento de curto prazo do

sistema;

2.- O grau de adaptação é usado pelos algoritmos de aprendizado de regra para guiar a criação e eliminação das cadeias, portanto influencia o comportamento de aprendizado de longo prazo do sistema.

O grau de adaptação nos SC é ajustado pelo algoritmo "Bucket Brigade" (baseado no modelo de competição). Os indivíduos neste algoritmo são as regras. Elas competem com outras regras pelo direito de tomar uma ação. Elas tomam uma ação colocando mensagens que disparam efetuadores que mudam o meio ambiente externo e colocando mensagens que guiam as últimas cadeias já habilitadas para serem disparadas. Elas recompensam às cadeias que colocaram as mensagens previamente e que guiaram a sua ativação. Elas são punidas ou ganham recompensa pelo desempenho.

Existem dois tipos de aprendizado nos SC. O primeiro envolve a adaptação de um conjunto fixo de regras de tal forma que elas respondam melhor à tarefa desejada. Isto é executado pelo algoritmo de atribuição de crédito, efetuando o ajuste do grau de adaptação das cadeias. O segundo tipo de aprendizado envolve a criação de regras novas e é executado por três mecanismos de heurística: os AGs; os algoritmos de "Cover Detector" e "Cover Effector" [HOLLAND, 1986], os quais forçam o sistema para que responda melhor ao seu meio ambiente; e o "Triggered Chaining Operator", o qual introduz fragmentos de cadeias de regras no sistema.

Resumindo, os SC possuem três níveis de atividades (Fig. III.1):

(1) Sistema de Desempenho (SD);

(2) Sistema de Atribuição de Crédito (SAC); e

(3) Sistema de Aprendizado de Regra (SAR).

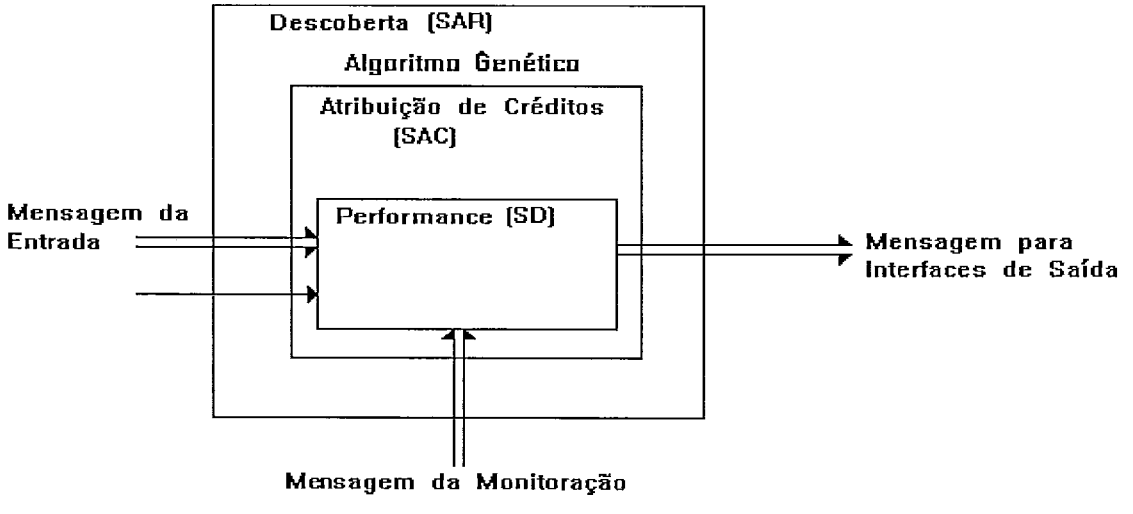

Fig. III.1: Sistema de Classificação

O SD (nível mais baixo) é a parte que interage diretamente com o meio ambiente. Geralmente, corresponde a um SE baseado em regras, mas é baseada na passagem de mensagens ("message-passing"), altamente padronizado e altamente paralelo.

O segundo nível, o SAC, determina qual das regras são efetivas, por isso deve avaliar de alguma forma as regras (atividade chamada de atribuição de crédito). Existem vários algoritmos que cumprem esta função, sendo o mais conhecido o algoritmo "Bucket Brigade" que usa somente interação local entre as regras para distribuir os créditos. Este algoritmo ajusta o grau de adaptação das regras. Outros mecanismos de avaliação são: recompensa (ou punição) desde o meio ambiente e taxas variáveis [HOLLAND, 1986].

O terceiro nível, o SAR, é necessário depois que o sistema tenha avaliado efetivamente milhões de regras e testado só uma parcela muito pequena. A seleção da melhor regra desta pequena parcela dá pouca confiança, dado que nenhuma das regras testadas é muita boa para ser selecionada. Aqui é onde o sistema deve estar habilitado para gerar novas regras para substituir pelas que têm atualmente. É importante mencionar que se é usado um procedimento que independe da experiência, a busca da nova regra pode ser muito lenta. O SAR usa AGs para gerar novas regras aproveitando a experiência acumulada pelo sistema através da história.

Um SC consiste de 4 partes básicas (Fig. III.2):

- Interface de Entrada: traduz o estado corrente do meio ambiente em uma mensagem padrão (Detectores);

- Cadeias: regras usadas para definir os procedimentos para processar as mensagens;

- Lista de Mensagens: contém todas as mensagens correntes;

- Interface de Saída: traduz algumas mensagens em ações que modificam o estado do meio ambiente (Efetuadores).

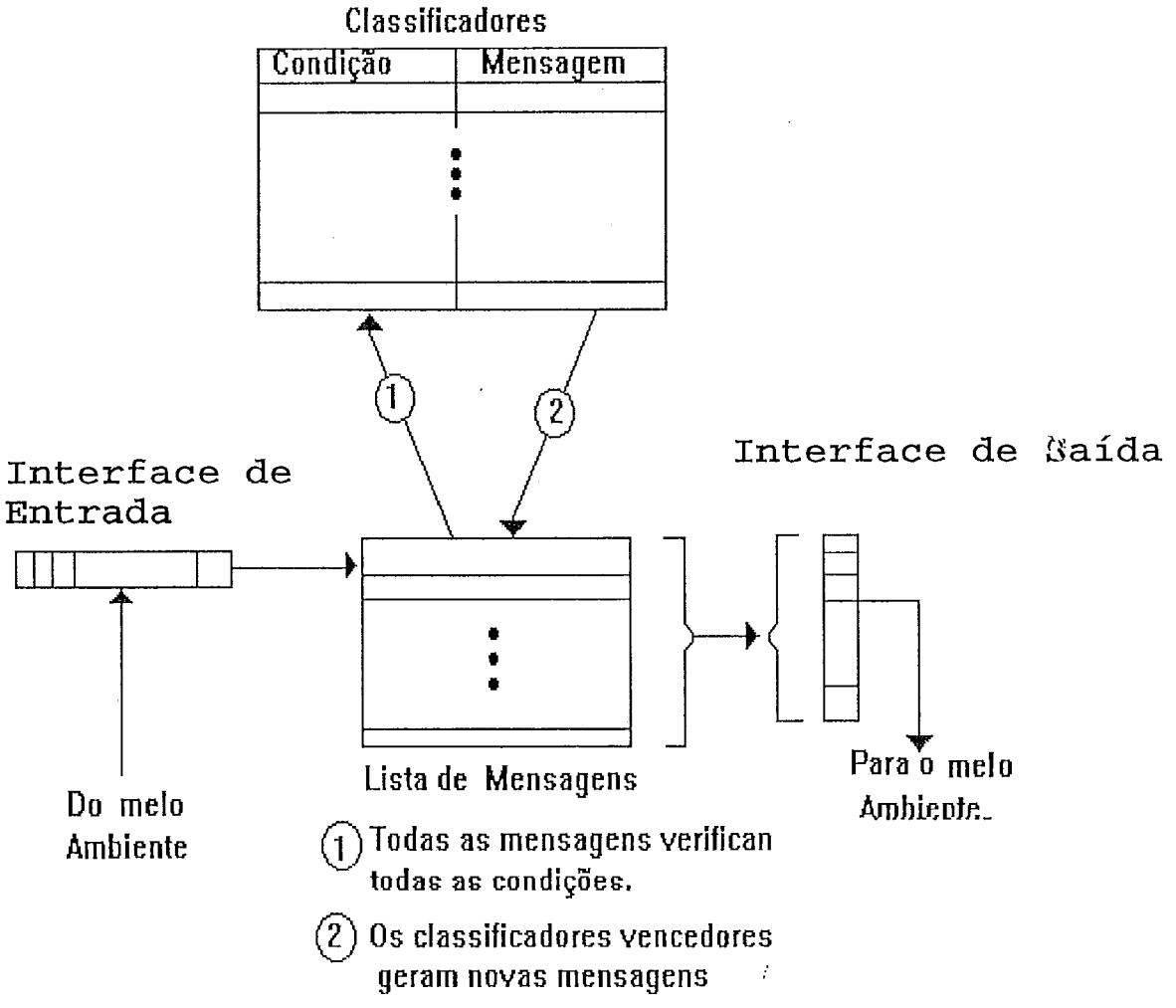

Fig. III.2: Componentes de um SC.

Na versão mais simples, todas as mensagens devem ser de um tamanho fixo (L) sobre um alfabeto específico, tipicamente uma cadeia binária de L bits.

O ciclo de execução básico de um SC consiste nos seguintes passos:

1. Coloca todas as mensagens da interface de entrada na lista de mensagens;
2. Compara todas as mensagens da lista de mensagens com todas as condições de todas as cadeias e grava as que foram satisfeitas;
3. Para cada conjunto que coloca a parte condicional de alguma cadeia, soma a mensagem especificada pela sua parte ação (regra = condição - ação) em uma lista de mensagens novas;

4. Substitui todas as mensagens da lista de mensagens pela lista das novas mensagens;
5. Traduz as mensagens da lista de mensagens para as necessidades da interface de saída, produzindo assim a saída corrente do sistema;
6. Retornar ao passo 1.

III.2.3.- Definições dos Elementos Básicos

A seguir mostra-se a definição dos termos relacionados com AGs que serão usados ao longo do texto para eliminar qualquer dúvida referente ao significado de cada elemento.

V={1,0}: Alfabeto binário;

V+={1,0,*}: alfabeto ternário, onde * pode tomar qualquer valor; 0 combina com 0, 1 combina com 1, * combina com 1 ou com 0;

L: comprimento do cromossoma;

A: cadeia que representa um cromossoma. São os indivíduos da população. Por exemplo, A=11001 tem L=5;

a: caractere. Por exemplo a cadeia $A=a_1a_2a_3a_4a_5$;

H: esquema que representa um conjunto de características similares de diferentes cromossomas. Por exemplo, H=*1111, o qual combina com 11111 ou com 01111;

n: tamanho da população, com indivíduos A_j j={1,...,n};

k: cardinalidade do alfabeto, onde se tem $(k+1)^L$ esquemas diferentes;

O(H): ordem do esquema H, igual ao número de posições fixas no esquema. Por exemplo, H_1=010**1 tem $O(H_1)=4$;

d(H): tamanho definido de um esquema, igual à distância entre a primeira e a última posição fixa em H. Por exemplo, d(011**10) = 7-1 = 6;

A(t): população de indivíduos no tempo t;

m(H,t): m exemplos do esquema H no tempo t;

p_i: probabilidade de selecionar a cadeia H_i;

p_s: probabilidade de sobrevivência do esquema H_i;

p_c: probabilidade de ocorrência do operador de cruzamento;

p_m: probabilidade de mudança aleatória de uma posição da cadeia (operador de mutação);

f_i: "fitness", valor de adaptação da cadeia H_i;

$\{1,0\}^k$: conjunto das mensagens binárias possíveis com L bits;

{1,0,#} : alfabeto ternário das mensagens;

"#" : símbolo "não interessa" dado um esquema ("don't care");

"*" : símbolo "não interessa" numa cadeia ("don't care").

É facil mostrar que para que uma mensagem satisfaça uma condição é necessário comparar posição por posição a mensagem e a condição, e se todas as entradas nas posições diferentes de "#" forem iguais, então a mensagem satisfaz a condição. Considerando-se que os AGs procuram uma solução ótima dentro de uma população inteira de cadeias, e considerando-se ainda que, dentro desta população, similaridades importantes entre cadeias podem ajudar a direcionar esta busca, torna-se relevante o estudo destas similaridades entre cadeias. Colocando de outra forma, deve-se questionar como uma cadeia representa uma classe de cadeias em que as similaridades estão localizadas em posições das cadeias? Isto é resolvido com uma estrutura definida por HOLLAND como esquemas ("schemas"), uma classe especial de "blocos de armar" ("building blocks"). Os esquemas têm um papel importante nas aplicações dos AGs. A chave para entender os AGs é entender a forma na qual são manipulados os esquemas.

Os esquemas são usados como blocos de armar a partir dos quais são construídas as novas cadeias, e são também usados para facilitar a transferência sofisticada de conhecimento de uma situação para uma outra. Os esquemas são basicamente descrições gerais das categorias de cadeias.

Uma condição (ou uma ação) para uma cadeia é definida por uma cadeia A de caracteres $a_1,a_2,...a_j...a_L$ de comprimento L sobre o alfabeto ternário {1,0,#}. Todas as possibilidades de combinação podem ser definidas com ajuda de um novo símbolo "não interessa", indicado por "*". Para definir um determinado esquema, devem-se especificar os caracteres na posição de interesse, preenchendo o resto da cadeia com a nova definição de "não interessa". Um exemplo é dado pela cadeia "*0##1**...*", onde deve-se focalizar a atenção na combinação "0##1" na posição 2 até a posição 5 da cadeia. Qualquer condição que satisfaça "0##1" na posição determinada é uma instância do esquema "*0##1**...*", e esse esquema define o conjunto de todas as condições que podem ser definidas com a combinação "0##1" na posição indicada.

O conjunto de todos os esquemas é definido pelo conjunto $\{1,0,\#,*\}^L$ de todas as cadeias de comprimento L sobre o alfabeto {1,0,#,*}.

Um SC, no tempo t, tem tipicamente várias cadeias que contêm uma componente dada ou esquema H; isto é, o sistema possui algumas instâncias de H. Atribui-se o valor s(H,t) ao esquema H no tempo t como a média do grau de adaptação de suas instâncias. Por exemplo, seja C_1 uma cadeia com a condição "10##110...0" e grau de adaptação $f(C_1,t)=4$, e a cadeia C_2 com a condição "00##1011...1" e grau de

adaptação $f(C_2,t)=2$, então existem só duas instâncias do esquema H="*0##1**...*" no tempo t, para o qual o esquema recebe o valor:

$$f(H,t) = 1/2\{f(C_1,t) + f(C_2,t)\} = 3$$

A fórmula geral é:

$$f(H,t) = 1/m(H,t) \sum_{C=m(H,t)} f(C,t)$$

É interessante mostrar que esta é uma boa heurística para guiar a construção das novas cadeias. Suponha um sistema com M cadeias. Se uma condição (ou ação) simples é uma instância de 2^L esquemas, então com as M cadeias existem entre 2^L e $M\cdot 2^L$ esquemas!. Ou seja, existe muita informação para calcular uma média.

Se f(t) é a média do grau de adaptação das cadeias no tempo t, então o esquema H está acima da média se:

$$f(H,t)/f(t) > 1$$

Apesar desta ser uma possível solução, não existe uma forma direta de calcular e usar um grande conjunto de médias {f(H,t)/f(t)}. Contudo, os AGs realizam em forma implícita aquilo que é impossível de realizar em forma explícita. Para isto, é necessário detalhar mais o passo do algoritmo que gera as novas cadeias.

O algoritmo atua sobre um conjunto B(t) de M cadeias $\{C_1, C_2, ..., C_M\}$ sobre o alfabeto {1,0,#} com grau de adaptação $s(C_j,t)$ atribuídos pelos seguintes passos:

1. Computar a média do grau de adaptação f(t) das cadeias em B(t), e atribuir o valor normalizado $f(C_i,t)/s(t)$ para cada cadeia C_i em B(t);
2. Atribuir a cada cadeia em B(t) uma probabilidade proporcional ao seu valor normalizado. Então, usando esta distribuição de probabilidades, selecionar de B(t) n pares de cadeias, $n << M$, e faça cópias delas;
3. Aplicar cruzamento ("cross-over"), (e possivelmente outros operadores genéticos) para cada par copiado, formando $2\cdot n$ cadeias novas. O operador de cruzamento é aplicado a um par de cadeias como segue: Selecione uma posição aleatória i, $1 \leq i \leq L$, e logo troque os segmentos da esquerda da posição i nas duas cadeias (Fig. III.3);
4. Substituir as $2\cdot n$ cadeias de menor grau de adaptação em B(t) com as $2\cdot n$ cadeias recentemente geradas no passo 3;
5. Faça t=t+1 para preparar o uso seguinte do algoritmo e volte ao passo 1.

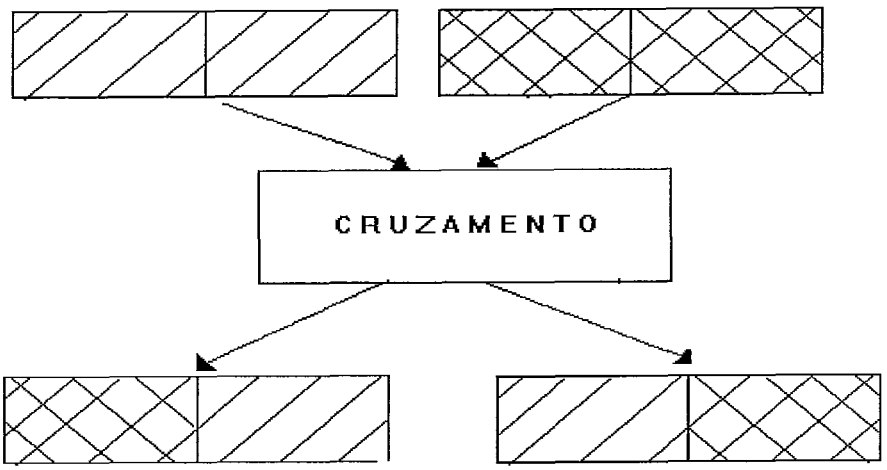

Fig. III.3: Operador de Cruzamento.

Os teoremas fundamentais para AG são procedimentos para inclinar a distribuição de probabilidades sobre o espaço {1,0,#}.

Teorema 1:

Seja p_c a probabilidade de que um par selecionado seja cruzado, e seja p_m a probabilidade que uma mutação ocorra em qualquer posição de bit (locus). Se m(H,t) é a fração da população ocupada pelas instâncias H no tempo t, então:

$$m(H,t+1) \geq [1-(p_c(d(H)/(L-1))\cdot p(H,t))]\cdot[1-p_m]^{O(H)}\cdot[f(H,t)/f(t)]\cdot m(H,t)$$

Isto representa um modelo determinístico de um algoritmo estocástico, e a prova está em [BOOKER, GOLDBERG e HOLLAND, 1989]. O teorema mostra que os esquemas fortes ocupam espaço na população em taxas exponencialmente crescentes.

Teorema 2:

Selecione um limite t sobre o erro de transcrição sob reprodução e cruzamento, e faça l tal que $l/L \leq (1/2)t$. Então numa população de tamanho $M=c_1\cdot 2^{l/L}$, obtida como uma amostra uniforme aleatória sobre $\{1,0\}^L$, o número de esquemas propagados com um erro menor que t sempre excede M^3, e a prova está em [BOOKER, GOLDBERG e HOLLAND, 1989].

É importante reconhecer que o AG só manipula M cadeias quando gera implicitamente e testa as novas instâncias de um grande número de esquemas envolvidos ($>>M^3$). No entanto, durante este procedimento, as amostras (instâncias) dos esquemas não tratadas previamente são geradas. Esta manipulação implícita de um grande montante de esquemas através das operações sobre 2·n cadeias por cada passo é chamado de paralelismo implícito.

III.3.- ALGORITMOS GENÉTICOS

O processo de aprendizado de regras dos SC usam AG. Basicamente, o AG seleciona as cadeias de alto grau de adaptação como "pais", formando uma "prole" nova pela recombinação dos componentes das cadeias dos pais. A prole elimina as cadeias fracas do sistema e entra em concorrência, para serem ativadas e testadas quando as suas condições sejam satisfeitas. Isto é, o AG imita o processo genético em evolução.

O seguinte é o esqueleto de um AG seqüêncial muito simples:

ALGORITMO_GENÉTICO_SEQÜENCIAL_SIMPLES;

```
t = 0;
inicializa P(t); -- P(t) é a população no tempo t
avalia P(t);
WHILE (não é satisfeita a condição de fim) DO
BEGIN
  t = t + 1;
  seleciona P(t);
  recombina P(t); /* aplica operadores de cruzamento e mutação */
  avalia P(t)
END.
```

O ciclo de execução básico de um AG é:

1. Do conjunto de cadeias, selecionar um par de acordo com o grau de adaptação, isto é, a de maior grau de adaptação será a mais provável de ser selecionada;
2. Aplicar os "operadores genéticos" de cruzamento e mutação ao par selecionado, criando uma nova geração de cadeias. O principal operador genético é o cruzamento, o qual simplesmente troca um segmento selecionado aleatoriamente entre os pares (ver Fig. III.3);
3. Substituir as cadeias mais fracas pela nova prole.

III.3.1.- Características dos AGs

É possível elaborar vários tipos de AG envolvendo variações, tais como operadores genéticos, tamanho variável da população, percentagem da população sujeita à mudança, etc. A seguir são apresentadas as características dos parâmetros clássicos dos AGs, segundo GREFENSTETTE (1986):

a) Tamanho da População N (TP): Uma questão em aberto no estudo dos AGs é o tamanho ótimo da população. O problema está no compromisso entre o montante da busca genética que pode ser efetuada e o montante disponível de tempo real. O TP afeta tanto o desempenho final quanto a eficiência do AG. Os AGs com pequeno TP são muito fracos [ROBERTSON, 1988], porque a população provê um tamanho de mostra insuficiente para muitos hiperplanos (uma estrutura de comprimento L representa 2^L hiperplanos). Um grande TP tem mais probabilidade de conter as representações de um grande número de hiperplanos. Ou seja, o AG pode realizar uma busca mais informada. Outro resultado é que com um TP pequeno, o AG converge prematuramente para soluções sub-ótimas. Por outro lado, um grande TP requer mais avaliações por geração, resultando possivelmente em uma razão de convergência tão lenta quanto inaceitável;

b) Razão de Cruzamento C (RC): a RC controla a freqüência com a qual é aplicado o operador de cruzamento. Em cada população nova, são cruzadas C·N estruturas. Quanto maior a RC, as novas estruturas são introduzidas mais depressa na população. Se a RC é muito alta, as estruturas de alto desempenho são descartadas tão rápido, que não chegam a serem selecionadas. Se a RC é muito baixa, a busca pode estagnar-se devido às baixas razões de exploração;

c) Razão de Mutação *M* (RM): a mutação é um operador de busca secundário que incrementa a variação da população. Depois de feita a seleção, cada posição de bit de cada estrutura da população nova, efetua mudanças aleatórias com uma probabilidade igual à *M*. Consequentemente, ocorrem aproximadamente M·N·L mutações por geração. Uma baixa RM serve para prevenir que qualquer posição de bit permaneça num único valor fixo na população completa. Uma alta RM produz uma busca essencialmente aleatória;

d) Intervalo de Geração *G* (IG): o IG controla a percentagem da população que será substituída em cada geração. Isto é, N·(1-G) estruturas de P(t) são selecionadas (aleatoriamente) para sobreviver intactas na população P(t+1). Um valor de *G*=1

significa que toda a população será substituida em cada geração. Um valor de G=0,5 indica que a metade da população sobrevive na seguinte geração;

e) Fator de Graduação W (FG): nos AG é importante manter a diversidade da população. Um super conjunto que domina a população até não poder melhorar mais, pode aparecer muito cedo num processo. Esta planura genética pode ser permitida quando o espaço do problema tiver sido suficientemente explorado. Executores similares podem aparecer mais tarde no ciclo de vida do sistema; eles devem ser avaliados e seu desempenho respectivo deve ser graduado para refletir suas respectivas adaptações;

f) Estratégia de Seleção S (ES): várias ES são possíveis. A ferramenta de AG METAMORPH considera duas delas. A primeira é uma ES pura, onde cada estrutura da população corrente é reproduzida em proporção à adaptação das estruturas. A segunda, é uma estratégia elitista: primeiro efetua uma seleção pura e logo as estruturas com melhor desempenho são escolhidas para sobreviver na próxima geração, permitindo modificá-las por cruzamento, mutação ou erro de amostra.

Um AG particular é indicado pelo valor de seus respectivos parâmetros N, C, M, G, W e S da seguinte forma AG(N,C,M,G,W,S).

III.3.2.- Operadores Genéticos

Muitas pesquisas mostram três operadores como sendo os principais reprodução, cruzamento e mutação [AUSTIN, 1990]:

Reprodução:

A população inicial $P(0)$ pode ser obtida heurística ou aleatoriamente. As estruturas das gerações seguintes são obtidas a partir de membros das gerações anteriores por um processo de seleção probabilístico, o qual garante que o número esperado de vezes que uma estrutura é escolhida é aproximadamente proporcional ao desempenho desta estrutura em relação ao resto da população. As estruturas de bom desempenho têm uma grande chance de contribuir na obtenção de novos descendentes.

Cruzamento

As estruturas são representadas por cadeias de símbolos. Se os símbolos são

binários, o cruzamento é implementado pela escolha aleatória de uma posição da cadeia (ponto de cruzamento) e logo são trocados os segmentos da esquerda ou da direita deste ponto com uma outra cadeia particionada similarmente. O cruzamento provê novos pontos para serem testados dentro do espaço do problema. Cada avaliação de uma cadeia de comprimento L soma conhecimento do desempenho dos n^L hiperplanos representados pela estrutura, onde n é o número de símbolos completos no alfabeto. Os AGs têm um alto poder computacional devido principalmente ao uso de um mecanismo simples de seleção para explorar a enorme quantidade de conhecimento eficientemente acumulado.

Mutação

A mutação cria indivíduos novos pela modificação de um ou mais valores de gens (bits) de um indivíduo existente. Não é um operador principal, dado que nos sistemas biológicos, a probabilidade de mutação dos gens é muito baixa. A mutação garante que a probabilidade de busca em qualquer região no espaço do problema nunca é zero e previne a perda completa de material genético através de seleção e eliminação.

III.3.3.- Aplicações dos AGs

Os AGs são extremamente flexíveis. Suas entradas e saídas podem representar uma larga variedade de fenômenos. Esta variedade inclui otimização combinatorial, processamento de imagens, sistemas de controle e aprendizado automático.

O sucesso do desempenho de sistemas em meio-ambientes dinâmicos sempre requer soluções adaptativas. Usualmente, a complexidade inerente do domínio (tais como ruído, alta dimensão, resposta multimodal e incerteza), dificulta determinar uma especificação que aceite uma solução a priori. Os sistemas adaptativos tentam resolver os problemas do domínio pela acumulação de conhecimento do problema e usando esta informação para gerar soluções aceitáveis. Estas soluções nem sempre são ótimas, mas geralmente são satisfatórias. Alguns desafios típicos aparecem nas áreas de configuração de sistemas complexos, tais como projeto de software, alocação de tarefas (algoritmos de itinerário), seleção de rota (problema do caixeiro viajante), e outros problemas de otimização e aprendizado automático. Alguns exemplos são mostrados a seguir:

Controle de Processos Dinâmicos: a forma clássica de resolver esta classe de

problemas é o ajuste manual do algoritmo com o perfil médio utilizado no processo. Depois do estágio inicial, o sistema usualmente é executado automaticamente até que alguma anormalidade degrade o sistema. Uma solução adaptativa direciona este desafio a modificar a base do algoritmo dinâmicamente em resposta às variações do processo [GOLDBERG, 1989].

Regras de Otimização e Indução: para esta aplicação a grande questão é o como descobrir e acentuar regras que trabalham bem, tirar aquelas que não funcionam (assumindo que a memória é limitada), e generalizar otimamente aquelas que estão retidas. Os AGs podem inferir regras para discriminar a partir de exemplos de classes de padrões [HOLLAND, 1986].

Descoberta de Novas Ligações Topológicas: nos sistemas de RNAs, os AGs podem guiar o descobrimento de arquiteturas completamente novas [DENIS e MACHADO, 1991]. Algumas arquiteturas podem não evoluir nunca, mas permanecem no ótimo para uma aplicação particular. (Esta aplicação envolve tanto a parte da engenharia dos sistemas de RNA, como os modelos biológicos dos neurônios). As pesquisas envolvem implementação de hardware e simulação de software [DRESS, 1987].

Simulação de Modelos de Evolução Biológicos: o uso dos princípios genéticos para projetar sistemas artificiais adaptativos tão sofisticados quanto os sistemas biológicos naturais é um objetivo importante desta pesquisa.

ACKLEY (1987) apresenta sete algoritmos para funções de otimização com o uso de AG. Os problemas mostrados são:
."Hillclimbing" iterado de ascensão escalonada e ascensão próxima;
."Hillclimbing" estocástico;
."Simulated Annealing" iterado;
.Busca genética iterada - combinação uniforme e ordenada;
."Hillclimbing" estocástico genético iterado;
onde são tratados os problemas de espaço linear, máximo local em intervalos largos e estreitos, espaços com planuras, etc.

III.3.4.- Desempenho dos AGs

O desempenho de um SC num problema de aprendizado pode ser avaliado de variadas formas, incluindo: o nível de desempenho atingido pelo SC no problema, o

tamanho da população de cadeias requerido para atingir um certo nível de desempenho, o número de respostas do meio ambiente requeridas para que o sistema tenha um bom comportamento, os operadores genéticos usados durante a reprodução enquanto o SC evolui no contexto do problema, e as condições dos parâmetros dado para o SC.

GREFENSTETTE (1986) apresenta duas medidas de desempenho para as estratégias de busca adaptativa: o desempenho "on line" e o desempenho "off line".

O desempenho "on line" de uma estratégia de busca s em uma superfície de resposta e é definida como segue:

$$U_e(s,T) = (1/n)\sum_{t=0..T}(u_e(t))$$

onde $u_e(t)$ é o desempenho da estrutura avaliada no tempo t. Isto é, o desempenho "on line" é o desempenho médio de todas as estruturas testadas no caminho da busca.

O desempenho "off line" de uma estratégia de busca s sobre uma superfície de resposta e é definida como segue:

$$U_e^*(s,T) = (1/n)\sum_{t=0..T}(u_e^*(t)), \qquad t=0, 1, .., T$$

onde $u_e^*(t)$ é o melhor desempenho atingido no intervalo de tempo [0,t]. O desempenho "off line" é uma medida muito importante quando a busca pode ser efetuada em forma "off line" (p.e. via um modelo de simulação), enquanto que a melhor estrutura é usada para controlar o sistema "on line".

No fim, para medir o desempenho global, define-se para um conjunto completo de superfícies de respostas E, o seguinte:

$$U_E(s,T)=100\cdot(1/n)\sum_e(U_e(s,t)/U_e(aleat,T)), \quad e \in E$$

$$U_E^*(s,T)=100(1/n)\sum_e(U_e^*(s,t)/U_e^*(aleat,T)), \quad e \in E$$

onde $U_e(aleat,T)$ e $U_e^*(aleat,T)$ são o desempenho "on line" e "off line", respectivamente, de uma busca aleatória pura sobre a superfície de resposta e.

Como U_E e U_E^* estão normalizados em 100 para buscas aleatórias, os U_E e U_E^* para estratégias de busca mais efetivas serão correspondentemente menores (para problemas de minimização).

III.4.- SISTEMAS DE CLASSIFICAÇÃO E O PARADÍGMA CONEXIONISTA

Os modelos conexionistas descrevem processos mentais em termos de ativação de padrões definidos sobre os nós numa RNA altamente interconectada [HINTON, 1985; RUMELHART et alii, 1986]. Os nós são unidades elementares que não mapeiam diretamente os conceitos. A informação não está localizada numa unidade individual em particular, sim pelas propriedades estatísticas dos padrões de atividade sobre a coleção de unidades. Uma unidade individual participa tipicamente de muitas representações do conhecimento. O conhecimento é então paralelo e distribuído sobre as múltiplas unidades. O principal princípio de armazenamento da informação são as conexões e não células de memória.

Numa RNA o papel de uma unidade de processamento mental é definido pelo peso das suas conexões, ambos excitatório e inibitório, com outras unidades. Neste sentido o conhecimento está nas conexões. O aprendizado, dentro desta técnica, consiste no ajuste dos pesos das conexões entre as unidades.

Os SC quando usados como algoritmos de aprendizado, não são ajustados para consistência, dado que as regras são tratadas como hipóteses parcialmente confirmadas e os conflitos são resolvidos por concorrência.

O lado esquerdo de cada regra é uma expressão conjuntiva simples, pelo qual não pode ser usado para expressar arbitrariamente, relações gerais entre atributos.

A coerência pode ser difícil de atingir em computação distribuída.

Um sistema deve construir dinâmicamente e modificar a representação do problema por si mesmo. Para isso é necessário flexibilidade nos níveis mais básicos dos conceitos, das relações e a forma na qual elas estão organizadas.

Os SCs usam grupos de regras como representação. A estrutura do conceito é modelada pela organização, variabilidade e distribuição do grau de adaptação entre as regras. A modularidade do conceito o faz simples de usar e também de modificar.

Este esquema distribuído para representar o conhecimento é similar à forma em que os conceitos complexos são representados nas RNAs [HINTON et alii, 1986]. Ambos os métodos usam uma coleção de elementos básicos de computação como blocos de armar. Os SCs usam regras condição/ação que interagem pela troca de

mensagens. As RNAs usam unidades simples de processamento que enviam sinais excitatórios ou inibitórios para as outras unidades. Os conceitos são representados em ambos os sistemas pela ativação simultanea de vários elementos de computação. Cada elemento de computação está envolvido na representação de vários conceitos, e a representação de conceitos similares compartilham elementos. A atualização de um conceito é um processo construtivo que ativa simultaneamente elementos que melhor casam com o contexto corrente. Esta técnica tem a importante vantagem de que algumas generalizações são atingidas automaticamente. A modificação dos elementos de uma representação afeta automaticamente todas as representações similares que compartilham aqueles elementos.

Existem também diferenças importantes entre os SCs e as RNAs, principalmente nas propriedades para construir os blocos que eles usam. A interação entre os elementos de computação numa RNA usa como operação primitiva a busca "melhor ajuste". A atividade em um padrão parcial de elementos é equivalente a uma especificação incompleta de um conceito. Tais padrões são automaticamente estendidos para um padrão completo de atividade que representa o conceito mais consistente com a especificação dada. Uma memória endereçável por conteúdo pode ser implementada eficientemente. A mesma capacidade é atingida nos SC pelo uso de apontadores e marcadores ("tags") para ligar regras relacionadas. Aqui é necessária uma ativação extensa dirigida para recuperar eficientemente o conceito apropriado.

Outra diferença está relacionada com a forma em que as induções são obtidas. A modificação do peso da conexão é o único mecanismo indutivo disponível nas RNAs [HOPFIELD, 1982]. E mais, as regras para atualizar os pesos são parte do projeto do sistema inicial que não pode mudar, salvo o refinamento de alguns poucos parâmetros. Os SCs permitem um amplo espectro de mecanismos de indução que vão desde aqueles para ajustar os grau de adaptação até analogias. Muitos destes mecanismos podem ser controlados (ou podem ser facilmente expressos em termos de) regras de inferências. Estas regras de inferências podem ser avaliadas, modificadas e usadas para construir conceitos de mais alto nível na mesma forma em que os blocos construídos são usados para construir conceitos de baixo nível.

Os SCs são semelhantes às RNAs pela ênfase nas microestruturas, coação múltipla (ação conjunta das múltiplas unidades) e a emergência de computações complexas a partir dos processos simples. Por outro lado, os SCs usam regras como unidade básica epistêmica, pelo qual evita uma redução do conhecimento para um conjunto de pesos conectados.

Os SCs ocupam uma parte importante no meio do paradigma conexionista e simbolista.

III.4.1.- Algoritmos Genéticos Conexionista e Grafos Induzidos

Um algoritmo de aprendizado baseado no conexionismo para representação do conhecimento pode ser usado para supervisionar e ampliar as aplicações dos operadores genéticos sobre a base da população analisando os esquemas fundamentais das cadeias.

O método de aprendizado usado por OOSTHUIZEN (1987), chamado algoritmo GRAND (GRAph iNDuction), usa o conhecimento "profundo" para dirigir as aplicações dos operadores genéticos. Este algoritmo combina as capacidades indutivas dos AGs e a indução de grafos, obtendo um sistema de aprendizado competente que atua sobre as categorias mais representativas das cadeias.

Cada alelo pode ser considerado como a definição de uma categoria completa de si próprio (a classe de todas as cadeias na população contendo aquele alelo x na posição y). Cada alelo pode ser usado para identificar um conjunto de cadeias. A interseção de dois destes conjuntos, dá como resultado um terceiro conjunto identificado pela co-ocorrência dos dois alelos. Este terceiro conjunto pode-se descrever por um esquema contendo unicamente os dois alelos envolvidos e o símbolo '*' nas outras posições. Por exemplo, o esquema "01*" contêm a interseção dos conjuntos identificados pelos esquemas "*1*" e "0**".

O objetivo é criar e manter os esquemas (nós no grafo) unicamente quando eles são necessários. No nível mais baixo, cada membro da população é atribuído ao seu próprio conjunto (sem o símbolo '*'). Este esquema pode-se então descrever em termos da interseção de k conjuntos (onde k é o comprimento da cadeia). Estes k conjuntos atingiram o nível dos esquemas de 1 alelo. O resultado é um grafo de dois níveis: o nível alto contendo esquemas de 1 alelo e o nível mais baixo contendo esquemas de k alelos.

O algoritmo GRAND efetua o seguinte procedimento: a estrutura do grafo é restrita de tal forma que responde à regra de que a interseção de dois conjuntos qualquer no grafo deve conter um único conjunto. Isto resulta numa política de fatorização consistente de dados. Por exemplo:

$$000 \cup 100 = (0** \cap *00) \cup (1** \cap *00)$$
$$= (0** \cup 1**) \cap *00$$
$$= (\#** \cap *00)$$

O interessante do algoritmo GRAND é que mantêm um regime de integração maximal dos esquemas. São identificados os esquemas comuns maximais (conjuntos) e são agrupadas as cadeias. A aplicação destas transformações na estrutura da rede representa o aprendizado. Na medida que as mudanças ocorrem, os nós e relações são constantemente acrescentadas e retiradas, as transformações são continuamente disparadas, e os novos esquemas são formados, dando como resultado uma rede de "auto-aprendizado".

O objetivo do SC Supervisionado é descobrir mais facilmente e tornar mais transparente, os aspectos de uma cadeia que fazem-na uma "boa" cadeia, para, então, se concentrar nestes aspectos. Da mesma forma devem-se tratar os maus aspectos de uma cadeia.

O SC Supervisionado opera da seguinte forma: para cada conjunto de esquemas é armazenado o número atual de membros, como também seu valor médio atual de $v(C_i)$, o ajuste de cada cadeia individual C_i que é membro de Q. Em cada caso, o valor armazenado pertence ao nó terminal (cadeia) da árvore seguido de um nó particular (esquema) do qual o nó é a raiz. Para melhorar a eficiência do método não são consideradas todas as cadeias da população (50% é uma boa percentagem). Na medida que as cadeias no grupo de melhor desempenho são substituídas pelas novas, alguns esquemas podem eventualmente ser redundantes no sentido de não ter nenhum membro, sendo retirados do grafo.

A operação de cruzamento opera com as cadeias com as seguintes características: dois esquemas que tem grau de adaptação acima da média, nenhum deles é subconjunto do outro, e ambos representam um número adequado de cadeias (os dois esquemas representam dois grupos de cadeias, sendo maximais e sistematicamente selecionadas).

O operador de cruzamento mantém as características de cada cadeia envolvida. Usa-se o chamado padrão de cruzamento, em vez do ponto de cruzamento. O padrão de cruzamento é obtido pela substituição dos alelos numa cadeia do esquema E com as características do esquema B e vice versa. Os AGs sempre preservam os bons

esquemas e geram instâncias deste. Só as melhores cadeias são inseridas no grafo induzido e só as estruturas mais comuns dos esquemas mais fortes são selecionadas para serem recombinadas pelo operador de cruzamento de padrão.

A Mutação Supervisionada opera com os esquemas das seguintes características: se um esquema A entra dentro de um outro esquema B na estrutura da árvore inserida do grafo induzido, significa que o esquema A é um subconjunto de B. Ou seja, o esquema A é mais específico que o B e portanto contém características não presentes no esquema B. Se a média(A)>média(B), (onde média(Q) é o grau de adaptação médio das cadeias em Q) isto significa que um traço extra em A atualmente melhora o ajuste das cadeias. A operação da mutação supervisionada combina os traços extras presentes em A com aqueles da cadeia B'.

A dificuldade de encontrar uma estratégia poderosa para a aprendizagem supervisionada em RNA com camadas escondidas é a motivação para o desenvolvimento de estratégias de aprendizado não supervisionado [KHANNA, 1990]. Esta é a razão pela qual o Capítulo IV apresenta um modelo de um AG Conexionista baseado nesta idéia de aprendizado supervisionado.

CAPÍTULO IV
MODELO DO ALGORITMO GENÉTICO CONEXIONISTA

Neste Capítulo apresenta-se um modelo de Algoritmo Genético Conexionista (AGC) inspirado no trabalho de MONTANA e DAVIS (1989), mas com um mecanismo de codificação diferente. O AGC é adequado para tarefas que operam num meio ambiente dinâmico. O modelo é baseado nos mecanismos de busca dos Algoritmos Genéticos (AGs) e em resultados provenientes das Redes Neuronais Artificiais (RNAs). As propriedades e os parâmetros do AGC permitem ao sistema se adaptar às mudanças do meio ambiente, respondendo adequadamente às situações para as quais não foi treinado. O AG é capaz de superar as limitações de aprendizado automático das RNAs, permitindo alterar o conhecimento na medida em que o meio ambiente muda. O AGC apresenta propriedades desejáveis de ambos os campos, ou seja capacidade de generalização das RNAs e capacidade de adaptação dos AGs. O AGC opera com uma família (população) de RNAs (indivíduos) com arquitetura idêntica, cujo conhecimento armazenado nos pesos é diferente.

IV.1.- INTRODUÇÃO

As Redes Neuronais Artificiais, (RNAs) [HOPFIELD, 1982; RUMELHART et alii, 1986], têm duas fases de operação: a fase de aprendizado e a fase de consulta. Na fase de aprendizado, a RNA constrói representações internas complexas de seu meio ambiente, sendo este um dos maiores objetivos da pesquisa de RNA (obter procedimentos eficientes de aprendizado [MACHADO e ROCHA, 1989]).

O algoritmo de aprendizado descobre os pesos que fazem a RNA apresentar o comportamento desejado, com base num conjunto de exemplos. O algoritmo de aprendizado deve ser capaz de modificar os pesos das conexões de tal forma que as unidades internas que não são parte da entrada ou da saída venham a representar aspectos importantes do domínio do problema. O conjunto de exemplos (o qual não é simples de determinar), geralmente não representa todos os padrões do domínio. A situação piora quando o meio ambiente é dinâmico (dado que é mais complexo), e na fase de consulta, a RNA falha dado que não é capaz de responder às mudanças do meio ambiente [DAVIS, 1987].

As limitações de usar as RNAs num meio ambiente dinâmico, como no caso do controle de processos, são produto da dificuldade destas em capturar todos os

padrões na fase de aprendizado [HINTON, 1989]. Daí surge a necessidade de dispor de um sistema que possa se adaptar facilmente às mudanças do meio ambiente.

Várias propostas têm sido apresentadas como uma solução a este problema. Uma delas é o uso dos Algoritmos Genéticos (AG) [DeJONG, 1980; GREFENSTETTE, 1986; HOLLAND, 1986; DAVIS, 1987; BOOKER et alii, 1989; GOLDBERG, 1989; AUSTIN, 1990], os quais têm mostrado ser uma boa alternativa pela sua flexibilidade na adaptação. Mas, estes AGs "puros" não possuem um mecanismo de aprendizado eficiente que oriente a busca, dado que sua forma de avaliar as soluções (função objetivo) é muito complexa de obter. Outro problema que apresentam estes AGs "puros" é o tamanho inicial da população, o qual chega a ser tão grande, que é inviável a sua implementação pelo espaço de memória requerido e pelo tempo de processamento necessário [SPIESSENS e MANDERICK, 1991; SCHIFFMANN e MECKLENBURG, 1991].

As linhas de pesquisa da aplicação de AGs em RNAs são as seguintes:

1.- Busca dos pesos ótimos, onde a RNA tem uma topologia fixa [MONTANA e DAVIS, 1989; MÜHLENBEIN, 1991; HEISTERMANN, 1991]; e
2.- Busca da topologia ótima [DENIS e MACHADO, 1991; SCHIFFMANN e MECKLENBURG, 1991]

A solução proposta neste trabalho é um modelo de Algoritmo Genético Conexionista (AGC) [OOSTHUIZEN, 1987; PETTEY et alii, 1987; MONTANA e DAVIS, 1989; MACHADO et alii, 1990; SOLAR, 1991; PALAGI, 1991] que combina ambas as técnicas, onde seja possível modificar os pesos das RNAs de tal forma que o conhecimento representado nas conexões, seja alterado na medida que o meio ambiente exija. Isto permite herdar as propriedades desejáveis de ambas as técnicas. O Capítulo define inicialmente os elementos básicos do AGC e suas características, tais como os operadores genéticos e os parâmetros, baseados no modelo de MONTANA e DAVIS (1989). No Capítulo V apresenta-se o estudo deste modelo em máquinas paralelas.

IV.2.- MODELO DO ALGORITMO GENÉTICO CONEXIONISTA

Um AG opera com uma população de indivíduos que representa as soluções do problema no qual está imerso. O modelo de AGC tem como população uma família de RNAs com uma arquitetura fixa. Cada RNA é representada como uma cadeia, cujos

componentes correspondem aos pesos das conexões entre os neurônios. O AGC opera sobre esta população escolhendo aqueles que melhor se adaptam à entrada proveniente do meio ambiente, gerando uma saída, quando existir uma RNA que responda àquela situação. Se na população de RNAs não existir nenhuma RNA que responda adequadamente, então os operadores genéticos combinam algumas destas RNAs para obter novas soluções (com idêntica arquitetura, mas com pesos diferentes) e elimina os piores indivíduos da população (substituição elitista).

O modelo do AGC se concentra nas seguintes suposições:

- O meio ambiente, cujas entradas e saídas podem ser representadas por uma RNA com uma arquitetura determinada: uma camada de nós de entrada, uma camada de nós de saída e uma camada de nós escondidos. Cada RNA corresponde a uma cadeia de símbolos de um comprimento fixo (L) sobre um alfabeto A. Este alfabeto binário é $\{0,1\}$. No modelo de AGC cada cadeia representa uma arquitetura de RNA com seus pesos;
- Cada ponto no espaço do problema (espaço total: $N=2^{L}\cdot 2$) pode ser considerado como uma combinação de pesos de uma RNA (indivíduo) representado unicamente dentro do sistema por uma cadeia gerada do alfabeto do meio ambiente. Nesta cadeia (cromossoma) do modelo de AGC cada posição específica (locus) representa uma conexão da RNA, e seus valores (alelos) indicam um bit do peso de uma conexão;
- O sistema mantém uma população P(t) fixa de RNAs representando o conjunto atual de RNAs do problema. O processo começa com uma população inicial especificada no início;
- O tempo é medido em intervalos discretos chamado gerações.

Independente de ser aprendizado supervisionado ou não, o AGC conhece as regras para executar a tarefa específica. A tarefa é definida por exemplos apresentados ao sistema, e uma função objetivo determina quão bem o sistema está executando-a. Cada RNA tem associado um grau de adaptação (dado pela função objetivo) que indica a eficiência dela em executar a tarefa desejada.

IV.2.1.- Modelo Básico

O modelo deve construir dinamicamente e modificar a representação do problema por si mesmo. Para isso é necessário flexibilidade nos níveis mais básicos dos conceitos, das relações e a forma pela qual elas estão organizadas. O AGC usa a família de RNAs como representação, e a estrutura do conceito é modelada pela organização,

variabilidade e distribuição dos pesos entre os neurônios de cada RNA.

Codificação do Cromossoma (Indivíduos): Um indivíduo é uma RNA com uma topologia definida e com conhecimento armazenado nos pesos das conexões. Uma conexão da RNA é codificado como um número real de 8 bits (-12.7 até 12.8) (Fig. IV.1). Cada indivíduo do AGC é composto por uma cadeia de comprimento fixo (*L*), onde a cadeia representa as conexões entre os nós de entrada, os nós de saída e os nós escondidos da RNA. Uma posição nesta cadeia (*locus*) corresponde a um bit da conexão entre dois neurônios (nó). O valor armazenado nesta posição (*alelo*) é um número binário tomado do alfabeto {0,1} que indica um bit do peso da conexão (o peso é composto de 8 bits). Os blocos de sinapses que têm efeito sobre um único neurônio da camada de saída devem ser contíguos aos outros blocos que tem influência sobre o mesmo neurônio. Isto indica que a estrutura deve conter contiguamente todos os blocos importantes ao aprendizado de cada neurônio de saída. A representação cromossômica garante a busca do ótimo da solução [PALAGI, 1991]. Desta forma a estrutura fica resumida na figura IV.1.

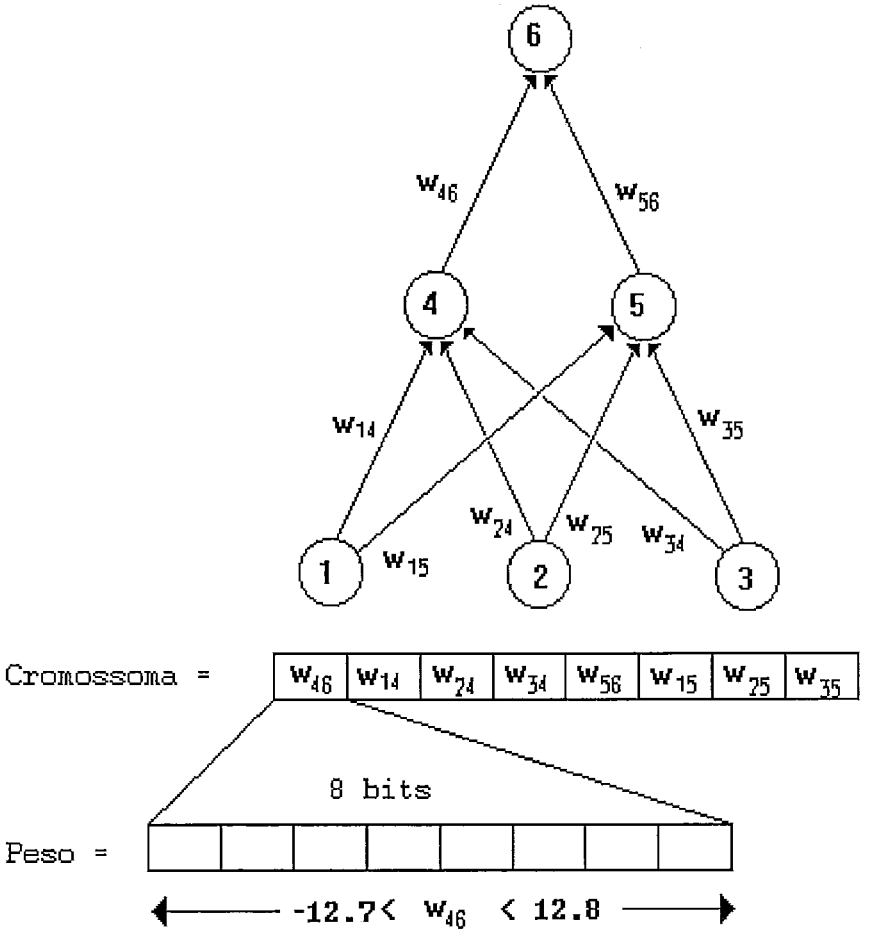

Fig. IV.1: Codificação do Cromossoma no AGC.

Seja uma RNA do tipo "feedforward" com três camadas, onde a camada de entrada possui l neurônios N_i, $1 \leq i \leq l$, a camada escondida possui m neurônios N_j, $1 \leq j \leq m$, e a camada de saída possue n neurônios N_k, $1 \leq k \leq n$.

Um bloco de sinapses é uma estrutura criada com o objetivo de reunir coerentemente as sinapses que influenciam um determinado neurônio de saída. Define-se o bloco de sinapses que influencia o neurônio N_k da camada de saída conectado ao neurônio N_j da camada intermediária pela lista resultante da concatenação da sinapse que une N_j a N_k com todas as sinapses que convergem para N_j:

$$\text{Bloco}(k,j) = \left[\mathop{\Big\|}_{i=1}^{l} w_{kj}, \left[w_{ji} \right] \right]$$

onde o símbolo $\|_{i=1}^{t}[w_{ji}]$ significa a concatenação dos valores A_i, $1 \leq i < t$.

A escolha de uma RNA "feedforward" se baseia principalmente na existência do Teorema de KOLMOGOROV sobre o Mapeamento em RNAs, mostrado a seguir e cuja prova está em HECHT-NIELSEN (87):

TEOREMA DE KOLMOGOROV: Dada qualquer função contínua f, tal que $f:[0,1]^n \rightarrow R^m$, $f(x)=y$, f pode ser implementada exatamente por uma RNA "feedforward" de três camadas, tendo l neurônios na primeira camada (entrada), $m=(2 \cdot l+1)$ neurônios na camada intermediária, e n neurônios na última camada (saída) [HECHT-NIELSEN, 90].

A existência deste teorema indica que a representação escolhida permite mapear qualquer função contínua sobre a arquitetura de RNA no cromossoma do AGC. O comprimento L do cromossoma é dado por:

$$L = (n \cdot m \cdot (l + 1) \cdot 8) \text{ bits}$$

Segundo o comprimento do cromossoma binário é possível determinar o tamanho, ou melhor a complexidade do problema como:

$$N = 2^{L} \cdot 2$$

Função Objetivo: A função objetivo para o modelo AGC deve guiar o aprendizado da RNA na direção do erro mínimo das suas saídas, quando existe um reforço que vem do meio ambiente. Para possibilitar a solução do aprendizado com AGs faz-se necessário definir uma função objetivo de forma que seu máximo represente o menor erro de

aprendizado. Como foi apresentado no Capítulo II pode-se definir o erro de aprendizado de uma RNA como o erro médio quadrático entre a resposta efetiva (o) e a resposta desejada (c) de cada neurônio N_k da camada de saída somado para cada um dos v padrões a serem aprendidos [PALAGI, 1991]. Ou seja:

$$e = \sum_{p=1}^{v} \sum_{k=1}^{n} \frac{1}{2} \left(c_{pk} + o_{pk}\right)^2$$

Para normalizar o erro, pode-se dividir pelo máximo erro possível n·v/2 do aprendizado dos v padrões, obtendo-se o erro médio, sendo a média tomada sobre os n neurônios e sobre os v padrões:

$$E = 2 \cdot \varepsilon / (n \cdot v)$$

A função objetivo deve maximizar a solução, para o qual se usa o inverso do erro de aprendizado (o qual deve ser minimizado), estabelecida por:

$$f(E) = 1/E^b$$

onde b tem o papel de controlar a intensidade desta relação (Fig. IV.2).

Uma solução para a função objetivo no caso de não existir aprendizado supervisionado é o uso do valor da função de KOHONEN para determinar quão bem a RNA está se comportando da mesma forma mencionada acima.

Heurísticas de Aprendizado: No AGC a adaptação de um conjunto fixo de RNAs é feito para que elas respondam melhor à tarefa desejada. Isto é efetuado pelo AG criando novas RNAs da mesma arquitetura (e pesos diferentes) e é executado pelos mecanismos de heurística dos Operadores Genéticos.

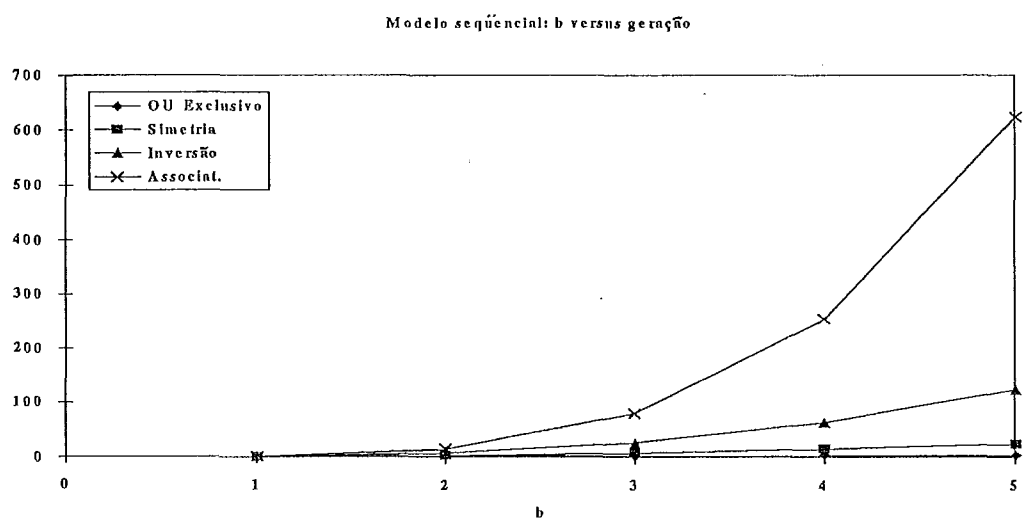

Fig. IV.2: Intensidade do Aprendizado (b).

IV.3.- PARÂMETROS DO MODELO AGC

É possível elaborar vários tipos de AGC envolvendo variações, tais como probabilidade de aplicar os operadores genéticos, tamanho fixo ou variável da população, percentagem da população sujeita à mudança, etc. A seguir são apresentadas as características dos parâmetros do AGC [GREFENSTETTE, 1986]:

Tamanho da População N (TP): GOLDBERG (1985) desenvolveu uma teoria para o ótimo do TP para AGs com código binário baseada no comprimento do cromossoma. Para um cromossoma com L=27, o TP ótimo é 77. Um cromossoma com L=27 representa uma RNA extremamente pequena (menor que uma RNA de 1×2×1 neurônios). Com um L=60, o qual ainda representa uma RNA muito pequena (2×2×2), o TP ótimo é 10.200, que é muito grande. Para uma RNA de tamanho médio é necessário um cromossoma de L=500, cujo TP é excessivamente grande, o qual não permitiria um processamento em tempo real. A teoria de GOLDBERG foi desenvolvida baseada num AG puro, ou seja o problema a ser resolvido é único. No caso das RNAs a quantidade de subproblemas a serem resolvidos depende da quantidade de padrões a serem apresentados à RNA. Portanto, por cada padrão novo, o TP ótimo deve incrementar-se proporcionalmente. Se se desejam ensinar 4 padrões numa RNA com L=60, o TP ótimo é (4)·(10.200)=40.800, o qual piora mais ainda o processamento em tempo real.

Baseado no trabalho de ROBERTSON (1987), e como se prova no Capítulo V no modelo paralelo do AGC, é possível mostrar o seguinte: no modelo sequencial do AGC um indivíduo é selecionado baseado no seu desempenho na população completa, mas no modelo paralelo um indivíduo é selecionado baseado no seu desempenho na população local. Esta diferença mostra três aspectos interessantes de antecipar: (1) O modelo paralelo do AGC é independente do TP, ocupando pouco espaço de memória; (2) A convergência do modelo é mais rápida, permitindo sua aplicação em tempo real; (3) Produz melhores resultados, dado que intercambiam os melhores indivíduos locais entre as diferentes populações.

O modelo implementado usou um TP=40 indivíduos, independente da quantidade de processadores usados.

Procedimento de Inicialização: Os pesos das RNAs da população inicial são escolhidos em forma aleatória com uma distribuição de probabilidade dada por $e^{-[x]}$. Isto é diferente da distribuição de probabilidades inicial dos pesos usualmente usados em "backpropagation", a qual é uma distribuição uniforme entre 0.0 e 1.0. A função de

distribuição de probabilidades usada é o reflexo das observações empíricas dos pesquisadores de que as soluções ótimas tem tendência a conter pesos com valores absolutos arbitrariamente grandes [MONTANA e DAVIS, 1989].

Escolha dos Pais: A escolha dos pais é feita a partir da população ordenada em forma ascendente (proporcional à função objetivo). De acordo com o valor da função objetivo são escolhidos pares de pais em forma probabilística para gerarem novos filhos. Aquelas RNAs com melhor desempenho têm maior probabilidade de serem selecionadas como pais da geração seguinte.

Probabilidade do Operador

Razão de Cruzamento (p_c): Em cada população nova, são cruzados $p_c \cdot N$ indivíduos. No AGC se efetuarão vários testes com p_c, usando os valores 0.5 até 1 com incremento de 0.1, obtendo como resultado a fig. IV.3;

Razão de Mutação (p_m): ocorrem aproximadamente $p_m \cdot N \cdot L$ mutações por geração. O valor de p_m no AGC foi escolhido empiricamente como 0.02, segundo fig. IV.4;

Intervalo de Geração G (IG): a percentagem da população que será substituída em cada geração, $N \cdot G$ indivíduos de P(t), é variável dependendo do grau de adaptação da população. O grau de adaptação é determinado pela soma do valor da função objetivo para todas as RNAs, não excedendo a 50% da população;

Fator de Graduação W (FG): No AGC é importante manter a diversidade da população, que se mede pela diferença entre o maior e o menor "fitness". O FG usado pode chegar até 0.5 da população sem perda de informação;

Estratégia de Seleção S (ES): O AGC usa a ES elitista; primeiro efetua uma ES pura e logo os com melhor desempenho são escolhidos para sobreviver na próxima geração, permitindo modificá-los por cruzamento ou mutação.

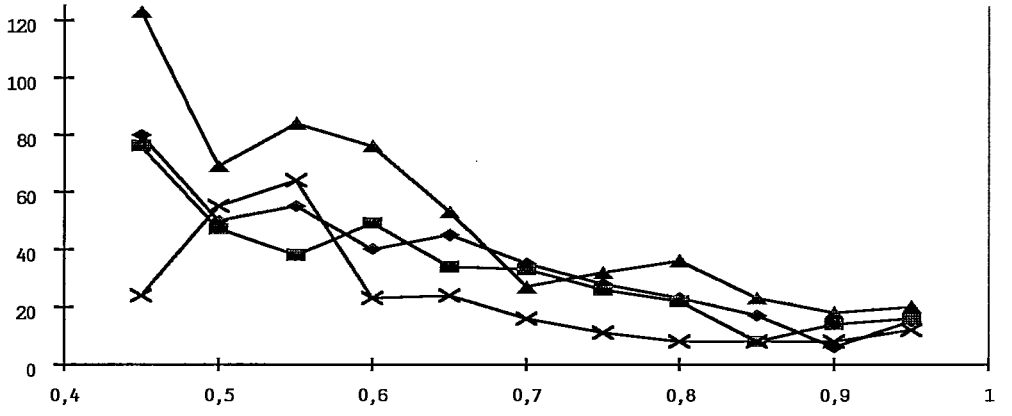

Fig. IV.3: Relação RC com número de Geração.

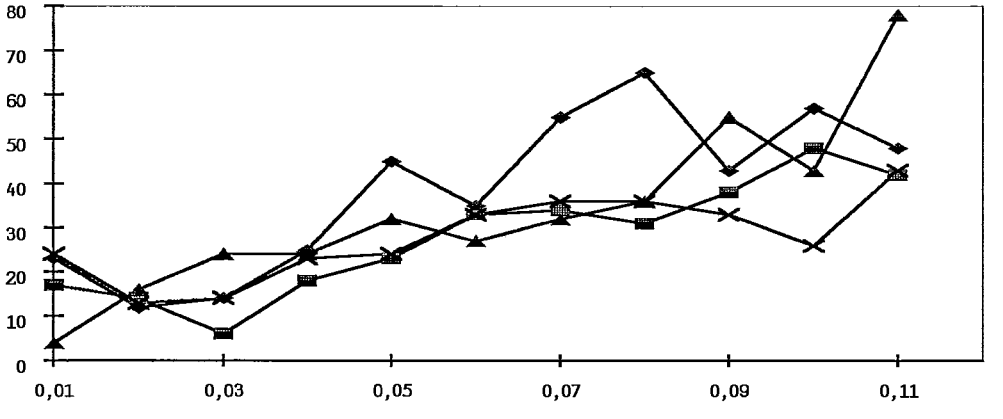

Fig. IV.4: Relação RM com número de Geração.

IV.4.- OPERADORES GENÉTICOS

O desempenho do modelo de AGC depende principalmente da boa escolha dos operadores genéticos, cujo rendimento depende da escolha dos parâmetros genéticos acima mencionados.

Com o objetivo de estudar o comportamento dos diferentes operadores nas diferentes situações, foram usados um grande número de operadores genéticos [MONTANA e DAVIS, 1989], selecionando o melhor conjunto de operadores para a

versão final do modelo. Os operadores podem ser agrupados em três categorias básicas: mutação, cruzamento e gradiente. As variações do operador de mutação são: mutação tendenciosa, mutação não tendenciosa, mutação de nós e mutação dos nós fracos. As variações do operador de cruzamento são: cruzamento dos pesos, cruzamento dos nós e cruzamento múltiplo. O operador de gradiente, toma um pai e produz um filho somando nas suas entradas um múltiplo do gradiente com respeito à função objetivo. A seguir descrevem-se as variantes dos operadores que foram usados:

Mutação não tendenciosa: para cada entrada do cromossoma, o operador substitui com uma probabilidade fixa p_m=0.04 um valor aleatório escolhido da distribuição de probabilidades de inicialização;

Mutação tendenciosa: para cada entrada do cromossoma, o operador soma com uma probabilidade fixa p_m=0.02 um valor aleatório escolhido da distribuição de probabilidades de inicialização. A mutação não tendenciosa é melhor que a tendenciosa pela seguinte razão: os pais são escolhidos com a tendência de serem melhores que a média, portanto, o grau de adaptação dos pais têm a tendência de serem melhores que uma escolha aleatória. Usando uma distribuição de probabilidades que tem tendência aos valores presentes do grau de adaptação dá melhores resultados que uma distribuição de probabilidades centrada em zero;

Mutação de nós: este operador seleciona n nós de não entrada da RNA. Para cada conexão de entrada nestes nós, o operador soma ao peso da conexão um valor aleatório tomado da distribuição de probabilidades de inicialização. Então codifica a nova RNA no cromossoma filho;

Mutação dos nós fracos: o conceito de peso dos nós é diferente do conceito de erro usado em "backpropagation". Por exemplo, um nó pode ter erro zero se todas as suas conexões de saída estão em zero, mas este nó não está contribuindo nada positivo para a RNA e não é um nó forte. A definição de peso de um nó escondido numa RNA "feedforward" é a diferença entre a avaliação da RNA intacta e a avaliação da RNA com aquele nó lobotomizado (com suas saídas em zero).

O operador de mutação de nós fracos pega uma RNA e calcula os pesos de cada nó escondido. Logo seleciona os m nós mais fracos e realiza a mutação em cada uma das suas conexões de saída e de entrada. Esta mutação não é tendenciosa se o peso do nó é negativo e é se o peso do nó é positivo. Logo codifica a nova RNA no cromossoma como um filho;

Cruzamento dos pesos: este operador coloca um valor dentro de cada posição do cromossoma filho selecionando aleatoriamente um dos dois pais e usando o valor na mesma posição que o cromossoma pai, como se fosse um cruzamento com múltiplos pontos;

Cruzamento dos esquemas: os diferentes nós numa RNA tem diferentes papéis. Para uma RNA com camadas totalmente interligada, o papel de um nó dado depende só da camada onde ele se encontra e não da posição dentro da camada. De fato, pode-se trocar o papel de dois nós *A* e *B* na mesma camada de uma RNA como segue: passa por todos os nós *C* ligados (pela entrada ou saída) com *A* (portanto com *B*). Intercambiar os pesos das conexões entre *C* e *A* com aquele da conexão entre *C* e *B*. Ignorando a estrutura interna, a nova RNA é idêntica à anterior (dada uma entrada, a saída é exatamente a mesma).

Os filhos produzidos pelos operadores de cruzamento acima mencionados afetam grandemente as estruturas internas dos pais. O operador de cruzamento de esquemas reduz esta dependência da estrutura interna fazendo o seguinte: para cada nó na primeira RNA pai, ele tenta achar um nó na segunda RNA pai, o qual tem o mesmo papel observando o número de entradas de ambas as RNA e comparando a resposta dos diferentes nós. Ele, então recombina a segunda RNA pai de tal forma que os nós que tem o mesmo papel estejam na mesma posição. Neste ponto, ele constrói um filho na mesma forma que o cruzamento tradicional. A maior utilidade deste operador com respeito aos outros operadores de cruzamento obtem-se no início da operação.

Gradiente: este operador calcula o gradiente para cada membro do conjunto de treinamento e soma este depois para obter o gradiente total. Então normaliza o gradiente dividindo-o pela sua magnitude. Os filhos são obtidos dos pais dando um passo na direção determinada pelo gradiente normalizado de tamanho "tamanho da passada", onde "tamanho da passada" é um parâmetro o qual se adapta através da execução da seguinte forma: se a avaliação do filho é pior que a do pai, o "tamanho da passada" é multiplicado pelo decaimento do tamanho da passada; se o filho é melhor que o pai, o tamanho da passada é multiplicado pelo expansor do tamanho da passada. Este operador difere do "backpropagation" no seguinte: o gradiente é normalizado de tal forma que o tamanho da passada não seja proporcional ao tamanho do gradiente.

IV.5.- OPERAÇÃO DO MODELO AGC

A operação do modelo é resumida no seguinte algoritmo:

```
ALGORITMO AGC;
INICIO
 Inicialização;
 REPETE
  Recebe Entrada;
  REPETE
   PARA cada RNA REALIZAR
   INICIO
    Dominio AG --> Dominio de RNA;
    Testar RNA com Padrão de Entrada;
    Obter Erro Quadratico Médio;
    Obter Valor da Função Objetivo;
   FIM;
   Ordenar População segundo Desempenho;
   Aplicar Operadores Genéticos;
   Indivíduo de Melhor Desempenho --> Piores Indivíduos;
  ATÉ convergência;
  Aplica Saída;
 ATÉ Fim do Processo;
FIM.
```

A Inicialização consiste em adquirir os dados iniciais do problema e inicializar a população do AGC. Segundo a ordem da RNA definida pelo usuário é calculado o comprimento do cromossoma e sua representação no domínio do AGC.

Recebe Entrada consiste em adquirir os dados do meio ambiente (o processo indica uma referência que determina a direção do controle). A entrada é considerada como um padrão que a RNA deve aprender (se ainda não o conhece).

Dado que o AG opera com uma representação cromossômica da RNA é necessário levá-la para o domínio das RNAs, no qual se realizará a fase de operação da RNA para verificar o erro médio desta com o padrão apresentado (aprendizado supervisionado). Este valor indicará a ordem da função objetivo de cada RNA.

Uma vez realizada a operação das RNAs e já calculada a função objetivo de cada indivíduo, é feita uma ordem da população desde o de melhor desempenho até o de pior desempenho. Isto é realizado para selecionar com maior probabilidade aqueles indivíduos de melhor desempenho.

São aplicados os operadores genéticos sobre a população ordenada, gerando-se uma nova população da qual serão escolhidos os de melhor desempenho para substituir os de pior desempenho da geração anterior.

Uma vez atingida a convergência é aplicada a saída determinada pela RNA obtida como ótima na solução do problema.

IV.6.- ALGUNS COMENTÁRIOS DO MODELO AGC

A técnica mais conhecida de "backpropagation" é útil nos problemas com treinamento simples. Na medida que é incrementada a complexidade do problema, o desempenho do "backpropagation" cai rapidamente. A técnica de busca pelo gradiente tende a chegar e ficar estagnada num mínimo local. O "backpropagation" com um ganho suficiente pode fugir do mínimo local, mas sem conhecer se o próximo mínimo é melhor ou pior. Quando o AGC é executado usando as representações que codificam as soluções mais satisfatórias como uma estrutura que contem contiguamente todos os blocos importantes ao aprendizado de cada neurônio de saída, os operadores podem gerar melhores filhos dos bons pais. O AGC produz populações de melhores indivíduos, convergindo finalmente para resultados muito perto do ótimo global. Os resultados obtidos com o AGC foram muito bons quanto à adaptação, mas o alto tempo usado na simulação indica o uso de outras técnicas para acelerar o processamento. Um trabalho que foi desenvolvido é a implementação de uma RNA baseada no enlace funcional [PAO, 89], a qual deu excelentes resultados num modelo de RNA "pura", onde se conseguiu reduzir significativamente o número de iterações no aprendizado.

O sucesso de sistemas em meio-ambientes dinâmicos sempre requerem soluções adaptativas. Usualmente a complexidade inerente do domínio (tais como ruído, alta dimensionalidade, resposta multimodal e incerteza) impedem as especificações de aceitarem as soluções a priori. O modelo de AGC tenta resolver os problemas do domínio pela acumulação de conhecimento do problema e usa esta informação para gerar soluções aceitáveis. Estas soluções nem sempre são ótimas, mas geralmente são satisfatórias. Alguns desafios típicos aparecem nas áreas de configuração de sistemas complexos, tais como alocação de tarefas (algoritmos de itinerario), seleção de rota (problema do caixeiro viajante), e controle de processos, onde a forma clássica de resolver estes problemas é o ajuste manual do algoritmo com o perfil médio utilizado no processo. Depois do estágio inicial, o sistema usualmente é executado automaticamente

até que alguma anormalidade degrade o sistema. O AGC é uma solução adaptativa que direciona este desafio a modificar a base de conhecimento dinamicamente em resposta às variações do processo [GOLDBERG, 1989].

Uma das principais vantagens do modelo de AGC apresentado é sua característica de trabalhar com um tamanho fixo da população, que permite ampliar a busca das possíveis novas soluções quando o mecanismo de avaliação indica que os indivíduos existentes na população não respondem bem às necessidades do meio ambiente. Isto reduz a quantidade de memória necessária para a implementação.

Na medida que as mudanças ocorrem, as RNAs são constantemente alteradas, as transformações são continuamente disparadas, e são formados os novos esquemas, dando como resultado uma RNA de aprendizado incremental.

O AGC têm duas características inerentes que o faz altamente apropriado para o aprendizado por descoberta: mantém uma variabilidade suficientemente alta para prevenir a convergência para ótimos locais; e a categorização e recombinação produz um alto paralelismo implícito.

Um AGC opera sobre uma população muito grande de vários grupos de indivíduos. Uma implementação paralela do AGC resolve o compromisso entre busca genética e tempo real. Existe um precedente para este tipo de população encontrado em espécies politípicas (como o ser humano). Uma espécie politípica é uma espécie composta de diferentes grupos isolados um dos outros (fisica e culturalmente), mas com a capacidade de produzir filhos com os outros [GOLDBERG, 1989].

No Capítulo V apresentam-se alguns modelos paralelos do AGC, aproveitando o paralelismo implícito dos AGs e das RNAs, onde se obtém um ganho considerável na aceleração no tempo de processamento, fazendo o problema independente do tamanho da população inicial.

CAPÍTULO V: MODELOS PARALELOS PARA O ALGORITMO GENÉTICO CONEXIONISTA

Na ciência da computação onde os algoritmos seriais são usualmente levados forçadamente para o paralelismo com a ajuda de truques e contorções, não deixa de ser uma ironia que os Algoritmos Genéticos (AGs) (altamente paralelos) são efetuados em forma serial com os mesmos truques pouco naturais e sem elegância. É surprendente que, até algum tempo atrás, poucos trabalhos tenham desenvolvido AGs para hardware paralelo existente ou proposto. Neste capítulo apresenta-se o estudo de diversos modelos paralelos para o Algoritmo Genético Conexionista (AGC). É apresentado um modelo "completamente paralelo e completamente distribuído", onde a ordem de processadores necessários está muito longe das máquinas paralelas reais (com uma quantidade limitada de processadores), como também o tempo de comunicação entre esses processadores é muito elevado. O estudo mostra que um modelo paralelo que possa ser mapeado naturalmente sobre uma máquina paralela real (Rede de Transputers da INMOS ou iPSC da Intel) dá bons resultados em termos de tempo de comunicação, tempo de processamento e tempo de resposta, obtendo-se conclusões interessantes.

V.1.- INTRODUÇÃO

Muitos componentes do AGC, incluindo as Redes Neuronais Artificiais (RNAs), gerenciamento das mensagens, ajuste dos pesos, e mecanismos de aprendizado são inerentemente paralelos.

Por outro lado, na seleção do tamanho ótimo da população, o problema está no compromiso entre o montante da busca genética que pode ser efetuada e o montante disponível de tempo real para aplicações em controle de processos. Se a população é muito pequena, então é possível que o AGC não ache uma boa solução pela insuficiência de esquemas na população. Se a população é muito grande, é possível que o tempo usado para efetuar todas as avaliações seja inaceitável [SPIESSENS e MANDERICK, 1991].

Uma implementação paralela do AGC resolve o compromiso entre busca genética e tempo real. Um AGC paralelo opera sobre uma população de vários grupos de indivíduos distribuídos. Existe um precedente para este tipo de população encontrado em espécies politípicas[1] (como o ser humano).

[1] Espécie politípica: espécie composta de diferentes grupos isolados (física e culturalmente) um dos outros, mas com a capacidade de produzir filhos

O trabalho de HOLLAND (1963) mostra a natureza paralela do paradigma reprodutivo e a eficiência inerente do processamento paralelo com AG puros. Mostra-se um tipo de computador celular, chamado "Computador de Circuito Iterativo" [HOLLAND 1959, 1960], para mapear a reprodução.

O primeiro trabalho do HOLLAND (1968) "Hierarchical Descriptions, Universal Spaces and Adaptative Systems", apresenta a descrição de máquinas complexas construídas a partir de um número fixo de componentes, onde o significado de esquema ou subcomponente estava dentro destas máquinas. Depois HOLLAND apresenta um sistema geral de aprendizado automático de máquina baseado em genética, chamado de processadores de esquema (schemata) em quatro fases (protótipos). Os protótipos incrementam de complexidade desde o protótipo I, uma simples máquina de Estímulo-Resposta, até o protótipo IV, um autómata complexo com estados internos e detectores e efetuadores variáveis. Estas propostas são a base do que mais tarde HOLLAND chamaria de Sistemas de Classificação (SC).

BETHKE (1976) calculou várias estimações de complexidade para um mapeamento particular de AG puro em máquinas paralelas. Conclui que o cálculo de adaptação da média da população é o principal gargalo serial em implementações de AGs desses tempos. Mas não simulou nem implementou um AG paralelo.

GREFENSTETTE (1981) examinou várias implementações paralelas de AGs puros, classificando 4 protótipos: Mestre-Servidor síncrono; Mestre-Servidor semi-síncrono; Concorrência assíncrona distribuída; e Redes.

RIOLO (1986) introduz o primeiro SC de propósito geral independente do domínio, chamado CFS-C. ROBERTSON (1987) apresenta a implementação de um SC *CFS na "Connection Machine System" (CM)[2], um computador de propósito geral massiçamente paralelo de granularidade altamente fina.

PETTEY, LEUZE e GREFENSTETTE (1987) apresentam um AG paralelo que consiste de um grupo de nós de AGs puros idênticos, um por cada nó num sistema multiprocessador. Cada nó do AG mantêm uma pequena população, a qual é uma porção de uma grande população, e funciona praticamente igual que um AG seqüencial. A única diferença entre um nó do AG e um AG seqüencial é que uma vez durante cada geração, o nó do AG se comunica com os seus vizinhos. Esta fase de comunicação

---

com os outros.

[2] A CM têm 65536 processadores, cada um com capacidade de 4096 bits de memória local, dando um total de 32 Mbytes. A topologia da rede é reconfigurada automaticamente de acordo às necessidades.

consiste em enviar os melhores indivíduos da população local para cada vizinho e receber os melhores indivíduos da população de cada vizinho. Depois de ter recebido os melhores indivíduos dos vizinhos, é necessário inserir estes novos indivíduos na população local. Esta inserção pode ser realizada pela substituição aleátoria dos indivíduos, ou dos piores indivíduos.

GOLDBERG (1989) propõe um procedimento projeto orientado a objetos para os AGs paralelos, onde existem dois projetos de modelos: o modelo de "Comunidade" e o modelo de "Planta de Polinização".

No modelo de Comunidade, o AG é mapeado em um conjunto de comunidades interligadas. As comunidades consistem de um conjunto de casas centralizadas e ligadas em um Centro. Os pais dão nascimento às gerações em suas casas e executam as funções objetivos. Os filhos são enviados para um único barzinho no Centro, onde devem achar a sua parelha. Depois do casamento, a parelha vai para o Agente de Bens Reais (no Centro) para achar uma casa. As casas são leiloadas entre as parelhas concorrentes. Se o centro está atualmente lotado, a parelha pode consultar novamente o Agente para conseguir casa numa outra comunidade. Se necessário, a parelha deve ir para Posto de ônibus para viajar para outra comunidade.

O modelo de Planta de Polinização consiste de uma série de nós de plantas ligadas numa rede de polinização. Os sementes crescem até se converter em plantas que geram polem, o qual é expelido através da rede de polinização. Cada elo da rede de polinização têm associada uma probabilidade de transmissão do polem. Esta capacidade permite às subpopulações das plantas ficarem mais ou menos isoladas umas das outras. O pólem das plantas fertilizam plantas maduras, criando mais sementes. A seleção ocorre localmente, pela seleção local das melhores sementes para fazer-se plantas maduras num modo probabilístico.

Embora estes modelos baseados em objetos pareça uma brincadeira, o intento do GOLDBERG é sério. Olhando cada subprocesso como um objeto ou entidade, é mais simples isolar o processamento, o requerimento de memória e a largura de banda da comunicação requerida para cada objeto. Esta perspectiva pode permitir um mapeamento mais eficiente dos AGs em computadores paralelos.

Existem outros trabalhos recentes de simulações paralelas e implementações de hardware, tais como JOG e VAN GUCHT (1987); SUH e VAN GUCHT (1987b); TENESE (1987) e ROBERTSON (1987) para AGs puros.

Neste Capítulo apresenta-se primeiro a justificativa do processamento paralelo e as características peculiares a nível de população e a nível de indivíduos do AGC. Logo apresenta-se a escolha da topologia de multiprocessadores a usar. Neste item, para poder fazer uma comparação entre os diversos modelos paralelos propostos, primeiro se faz uma análise do tempo de comunicação entre os processadores. Depois se faz uma análise do tempo de execução e o tempo total de processamento em termos dos parâmetros do modelo, tais como tamanho da população do AGC, ordem da RNA, tamanho da mensagem a transmitir entre os processadores e quantidade de comunicação necessária para os diversos modelos. No fim apresentam-se os resultados do modelo paralelo escolhido e implementado numa arquitetura hipercúbica (Rede de Transputers e no simulador do iPSC da Intel).

V.2.- JUSTIFICATIVA DO PROCESSAMENTO PARALELO

A essência da tese é mostrar a viabilidade de implementar em tempo real um modelo paralelo para o AGC apresentado no Capítulo IV.

A primeira argumentação é justificar a escolha de processamento paralelo e distribuído ao invés do tradicional processamento serial. A justificativa é oriunda das próprias características funcionais dos algoritmos que compõem o AGC, a dizer, os AGs e as RNAs, logo precisa-se entender como essas características contribuem para o desempenho de um ambiente distribuído e paralelo.

O AG possui o chamado paralelismo explícito, dado que é capaz de manipular 2·n cadeias em cada passada. Cada esquema representa um conjunto de cadeias no espaço do problema, que são tratadas implicitamente.

A figura V.1 mostra uma função de otimização com diversos pontos que são representados pelas cadeias no AG. Na figura V.2 se mostra uma evolução do AG depois de algumas gerações, onde é possível concluir que um tratamento paralelo e distribuído do AG consegue um ótimo da solução em poucas gerações.

Agora, dado que o AGC opera com uma população de indivíduos que representam RNAs, também é possível justificar o uso de multiprocessamento para acelerar a operação destas RNAs [CHIOU, 1991].

Finalmente, a interação entre ambos os algoritmos permite um tratamento

distribuído, enquanto é possível isolar parcelas de populações para seu tratamento independente, e logo permitir uma comunicação entre as populações distribuídas, trocando desta forma indivíduos que podem colaborar e acelerar a convergência para uma solução ótima [PETTEY, LEUZE e GREFENSTETTE, 87].

As soluções propostas neste capítulo, uma no simulador do hipercubo iPSC/2 da INTEL e a outra numa rede de Transputers, apresentam diferentes justificativas na sua implementação. Na Rede de Transputers (como se mostrará) é necessário realizar um tratamento das mensagens enviadas de um processador para um outro, evitando os "deadlocks" que poderiam ocorrer. No hipercubo iPSC/2 é possível explorar as primitivas de comunicação do sistema, sendo muito mais transparente na programação.

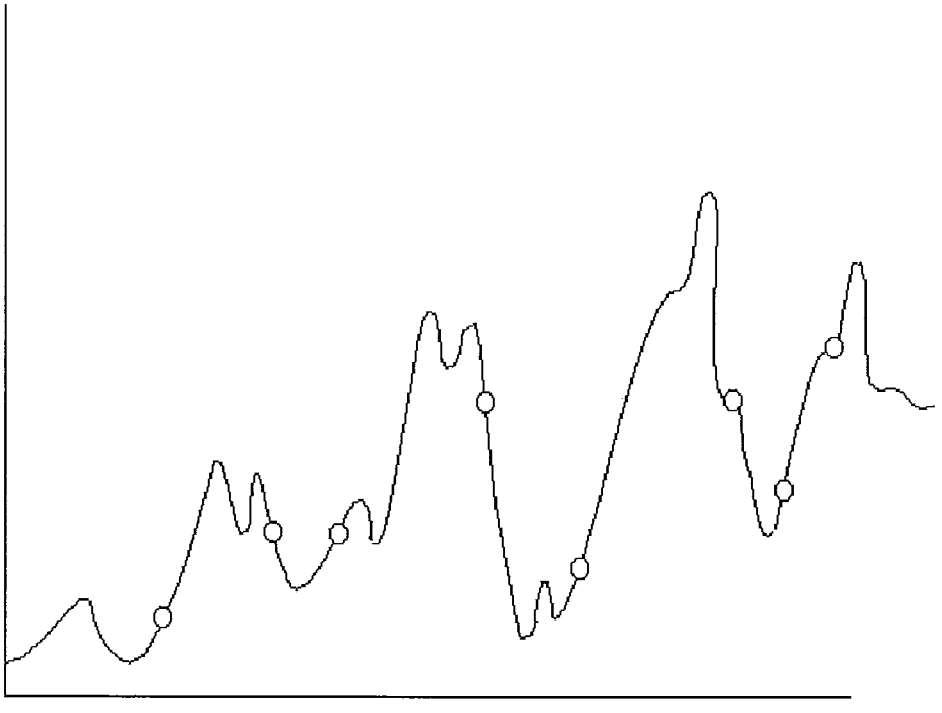

Fig. V.1: Função de Otimização com diversos Pontos.

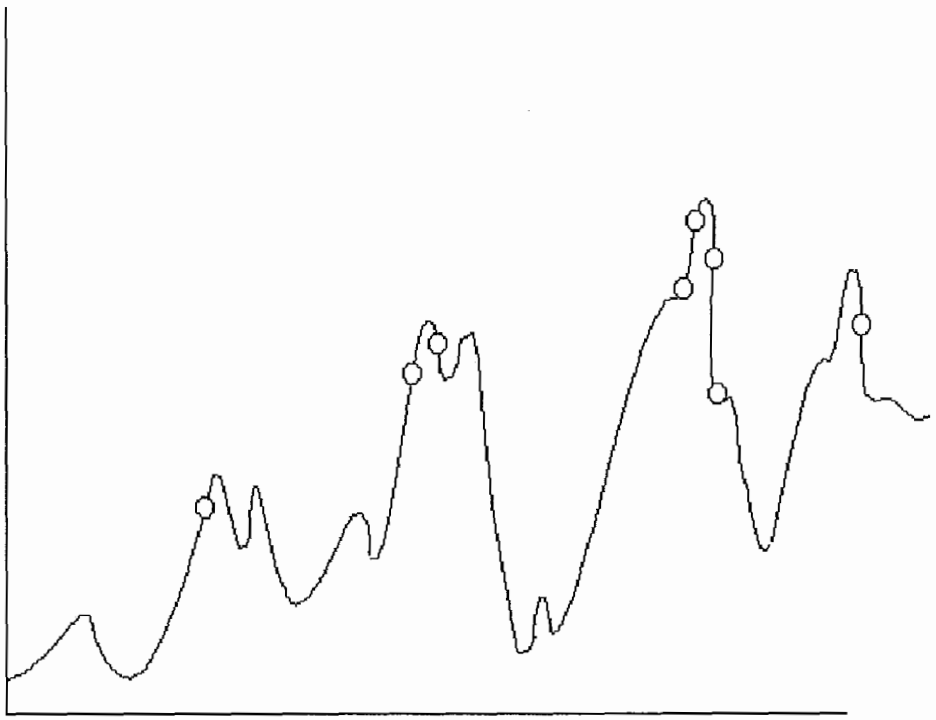

Fig. V.2: Evolução de uma Função de Otimização.

V.3.- ESCOLHA DA TOPOLOGIA DE MULTIPROCESSAMENTO

O desejo de obter computadores mais poderosos levou para o uso de arquiteturas paralelas. Uma forma de construir tais sistemas é interconectar vários processadores de propósito geral. O desempenho vai depender do balanceamento da carga e da comunicação. Ambas não são independentes: o decremento de uma, geralmente implica no aumento da outra. Os projetistas de computadores paralelos têm seguido dois caminhos diferentes: SIMD e MIMD. No modelo SIMD, a máquina é geralmente controlada por um processador central, para o qual o projetista deve propor os procedimentos internos de comunicação. Pelo contrário, os computadores MIMD são geralmente construidos em forma modular, permitindo diferentes tipos de topologias a serem usadas, para o qual é necessário desenvolver os procedimentos de comunicação baseados nas facilidades de comunicação do sistema. O modelo será desenvolvido para uma máquina MIMD, para o qual serão tratados os procedimentos de comunicação do modelo.

Agora, se os processadores estão distribuídos, mas a memória é compartilhada, a principal dificuldade é a manipulação do acesso concorrente à memória compartilhada. No entanto, se a memória é também distribuída, a transferência de dados depende da topologia. O trabalho baseia-se neste último esquema com memória distribuída, para o qual é necessário escolher uma topologia determinada.

Na escolha de uma topologia, os objetivos podem dividir-se em duas categorias bem fortes: custo e desempenho. Para o desempenho é necessário ter uma combinação de diâmetro pequeno, uniformidade, caminhos curtos e redundantes. Para o custo é necessário dispor de mínimo número de caminhos, "layout" eficiente, algoritmos simples de roteamento, grau fixo e uma construção simples com a tecnologia disponível. Ambas as listas possuem contradições e qualquer decisão é um compromiso [HILLIS, 1986]. Por esta razão, tem-se usado diferentes tipos de topologias para o projeto de computadores paralelos: hipercubo, anel, torus, grafo completo, grade, etc. O modelo desenvolvido baseia-se na topologia de Hipercubo retangular, cuja escolha se justifica pela eficiência e rendimento deste esquema em diversas aplicações.

V.3.1.- Restrições da Comunicação

Os processadores se comunicam pela troca de mensagens, mas os esquemas de comunicação na execução de um programa, é muito dependente do problema.

Afortunadamente, o estudo de algoritmos clássicos traz algumas "comunicações genéricas" que aparecem muito, por exemplo em algebra linear e não linear, ou processamento de imagem. No modelo proposto são usados os seguintes esquemas de comunicação:

Difusão ("Broadcasting"): enviar uma mensagem de um processador para todos os outros processadores;

Coleta ("Gathering"): este problema é a simetria do problema de difusão; enviar uma mensagem de todos os processadores para um processador;

Distribuição: um processador envia uma mensagem diferente para cada um dos outros processadores. O problema de coleta, onde um processador recebe uma mensagem de todos os outros é o problema inverso da distribuição. Ambas as operações são sempre usadas em situações assimétricas, onde um nó age como processador mestre e os outros como seus servidores. O mestre distribui conjuntos de dados diferentes para cada servidor, os quais realizam as computações com estes. Então o mestre coleciona todos os resultados;

Multi-Distribuição: Enviar mensagens de cada processador para cada um dos outros.

Neste trabalho se considerará um sistema de multiprocessamento com memória distribuída, sob um esquema de computação MIMD. As memórias e processadores estão conectados por uma rede de interconexão ponto-a-ponto; descrita em termos de nós e enlaces ("link").

Um nó numa arquitetura de rede típica consiste de um processador, uma memória, um barramento e vários canais DMA ("Direct Memory Access"). Cada canal DMA conecta o nó com um dos seus vizinhos. A memória e os canais DMA estão conectados no barramento, e o processador está conectado com a memória. A informação é então transmitida pelo apropriado canal DMA via o barramento para a memória do vizinho via seu barramento. O caminho de comunicação entre dois canais DMA é chamado enlace.

Na seqüência são descritas as diferentes leis de comunicação usadas nos modelos de comunicação real, para selecionar aquela que serve para os modelos paralelo apresentados.

Consideram-se os problemas das mensagens que são enviadas no modo "store-and-forward" ou "packet-switched", onde um nó não pode usar o conteúdo da mensagem até receber todos os bits (caso oposto ao modo "circuit-switched"). Isto, pelo

fato que uma máquina no modo "circuit-switched" também pode efetuar o modo "store-and-forward". Ainda mais, não existe nenhum algoritmo melhor para "circuit-switched" que aqueles projetados para "store-and-forward" para resolver problemas de comunicação intensiva como difusão e coleta [FRAGNIAUD et alia, 1991].

Agora, para o caso de dois processadores, p_1 e p_2, ligados diretamente, existem duas possibilidades:

(1) Só uma mensagem pode viajar entre p_1 e p_2, desde p_1 para p_2, ou desde p_2 para p_1. Os enlaces são então chamados de "half-duplex" (H);
(2) Duas mensagens podem usar o enlace no mesmo instante de tempo, uma em cada direção. Estes enlaces são chamados de "full-duplex" (F).

Existem três tipos de classificação para comunicação, dependendo de onde ocorre o "gargalo":

(1) Se, durante a comunicação, um processador pode usar só um dos seus enlaces, isto se conhece como limitado-pelo-processador ("processor-bound"), porque o processador não pode transferir em forma rápida as mensagens e limita a eficiência da rede. Também é conhecido como "1-port" ou "whispering";
(2) Pelo contrário, quando um processador pode usar todos os seus enlaces ao mesmo tempo, a comunicação se diz que é limitada-pelo-enlace ("link-bound"), porque agora é o número de enlaces que limitam a comunicação. Também é conhecido como "n-port" ou "shouting";
(3) Dentre as duas possibilidades extremas, se têm o caso quando um processador pode usar só k enlaces ao mesmo tempo; a comunicação se diz que é limitada-pelo-DMA ("DMA-bound"), porque o que limita a comunicação é o número de canais DMA diferentes que podem acessar o barramento. Um outro exemplo é quando os processadores possuem um grau limitado. Pelo uso de "crossbars" e multiplexores, é possível conectá-los via uma rede com grau maior, mas pelos enlaces internos, os processadores não podem usá-los todos ao mesmo tempo.

No trabalho apresentado, o esquema que será analisado é "full-duplex" limitado-por-enlace, descrito como o modelo F* em FRAGNIAUD (1991).

Agora, é necessário modelar o tempo de comunicação T para enviar uma mensagem desde um processador para um dos seus vizinhos. Muitas pesquisas mostram que o custo elementar T pode depender fortemente do comprimento L da mensagem

enviada. Então, o tempo de comunicação entre dois processadores adjacentes é geralmente modelado como segue (modelo linear):

$$T = \beta + L \cdot \tau$$

onde β é o custo para iniciar a comunicação ("start-up") (reconhecimento,...) e τ é o tempo da propagação de uma unidade de dado (um byte), $1/\tau$ é a largura de banda dos enlaces. Este modelo permite passar mensagens de comprimento variável que podem ser particionadas e recombinadas. Quando podem-se considerar propriedades teóricas dos grafos relacionados com comunicação, então pode-se assumir que o comprimento da mensagem é pequena de tal forma que é possível simplificar a expressão linear de T como uma expressão constante. Isto é, o tempo de comunicação entre dois processadores é igual a uma unidade de tempo (modelo constante):

$$T = 1$$

Neste modelo, sem mudar o tempo de propagação, as mensagens podem ser particionadas e recombinadas.

V.3.2.- Definição da Topologia H_d

Um hipercubo H_d é um grafo com $N=2^d$ vértices. Cada vértice é marcado por um número binário de d-bits. As arestas estão entre dois vértices cuja marca difere em precisamente um bit. O hipercubo pode ser definido recursivamente como segue. Um hipercubo de 1-dimensão é uma aresta com um vértice marcado 0 e o outro vértice marcado com 1. Um hipercubo de (d+1)-dimensões é construido apartir de dois hipercubos de d-dimensão, H_d^0 e H_d^1, pela soma de arestas desde cada vértice em H_d^0 para vértices em H_d^1 que possuem a mesma marca e logo prefixando todas as marcas em H_d^0 com um 0 e todas as marcas em H_d^1 com um 1.

O hipercubo é usado em vários computadores paralelos, tal como o NCube e o da série iPSC por seu pequeno diâmetro $D=d=\log_2 N$ e também pelo seu pequeno grau $\Delta=d=\log_2 N$. Mais ainda, o hipercubo pode simular facilmente outras topologias e tem capacidade de tolerância a falhas. Desafortunadamente, possui $(N \cdot \log_2 N)/2$ arestas.

Dado um grafo conexo e uma mensagem originada em u, o tempo de difusão do vértice u, $b_M(u)$, é o tempo mínimo requerido para completar a difusão desde o vértice u sob o modelo M (neste trabalho é o modelo F*, por tanto se tratará o caso de $b_{F*}(u)$). O tempo de difusão do grafo G, $b_{F*}(G)$, é definido como o valor máximo dos

tempos de difusão de cada vértice $u \in G$, i.e. $b_{F*}(G)=\max\{b_{F*}(u) | u \in V(G)\}$. Podem-se dar definições similares para o tempo de coleta $g_{F*}(G)$, o tempo de distribuição $s_{F*}(G)$ e o tempo de multi-distribuição $m_{F*}(G)$.

Para o modelo constante se tem o seguinte [FRAGNIAUD, 1991]:

Difusão: dado que um processador pode-se comunicar com todos os seus vizinhos, é possível usar o seguinte algoritmo: quando receber uma mensagem, envia-a para todos os seus vizinhos. Por tanto, é fácil verificar, que para qualquer grafo G de diâmetro D:

$$b_{F*}(G)= b_{H*}(G)= D$$

o qual também é ótimo.

Coleta: este problema em qualquer grafo tem como limite inferior o diâmetro do grafo. Cada nó pode simplesmente enviar suas mensagens para todos os seus vizinhos a cada passo, e o tempo necessário para colecionar é simplesmente o diâmetro.

Distribuição e **Multi-Distribuição**: dado que as mensagens podem ser particionadas e recombinadas sem qualquer perda, não existe necessidade de distinguir entre difusão e coleta ou entre distribuição e multi-distribuição.

V.3.3.- Rede de Transputers

O Transputer pertence à família de dispositivos VLSI com tecnología RISC, incluindo entre outras características, processador de ponto flutuante, processador de propósito geral de 32 bits, memória interna, quatro enlaces de comunicação. Estes enlaces permitem a conexão direta com outros Transputers. Maiores informações técnicas do Transputer aparecem em INMOS (1987, 1988a, 1988b, 1988c)

As principais vantagens das comunicações ponto a ponto entre os Transputers são:

1. Não necessitar de barramento de alto desempenho para comunicação;
2. A velocidade média de comunicação não satura quando processos extras são adicionados;
3. Sistemas de portes arbitrárias podem ser construídos, desde que todas as conexões sejam locais e distância curta; e
4. O "layout" das placas é fácil de ser projetado.

As principais desvantagens da comunicação ponto a ponto é o número de

processadores intermediários (distância entre processadores fonte e destino) que as mensagens tenham que percorrer até atingir o processador destino.

A forma de controlar o roteamento das mensagens (de fonte a destino) e de sincronizar a mudança de níveis de simulação usada na implementação aparece em CHIOU (1991).

Uma vantagem de usar o Transputer é que permite criar tantos processos concorrentes quantos foram necessários para cada modelo.

V.3.4.- Simulador do Hipercubo iPSC/2 da INTEL

A comunicação nesta arquitetura é muito transparente se comparada com o Transputer. É necessário destacar que só foi usada a comunicação síncrona que provê a linguagem C deste Hipercubo, sendo estas SENDV e RECV [INTEL, 1990].

A desvantagem deste simulador (do Sistema Operacional) é que não permite criar além de 14 processos (em XENIX) ou 23 processos (em UNIX 4.2 BSD e UNIX 5.3 ATT), o qual limita muito o tamanho da simulação, mas serve para propósitos de depuração da lógica do modelo AGC.

V.4.- MODELOS PARALELOS PARA O ALGORITMO GENÉTICO CONEXIONISTA

A paralelização do modelo AGC pode-se efetuar de diversas formas. Neste item apresentam-se as seguintes técnicas de paralelização: paralelização heurística, paralelização topológica, paralelização pela partição do espaço de busca e paralelização funcional. Cada uma destas técnicas será detalhada em separado.

É necessário indicar que a técnica de paralelização heurística é a única usada em todos os modelos apresentados. Esta consiste em parcelar a população global de RNAs em populações locais, permitindo uma independência entre as buscas efetuadas nos múltiplos processadores. Na figura V.3 pode-se ver a idéia desta heurística. Existe um processo mestre que controla a operação global do sistema, distribuindo e colecionando as soluções de cada processo servidor. Cada processo servidor faz uma busca local da solução, compartilhando só os melhores indivíduos com os outros

processos servidores. A forma na qual é processada a informação em cada população local dá origem às outras técnicas de paralelização, indo de um modelo totalmente paralelo e totalmente distribuído até um modelo que aproveita em maior grau a capacidade de cada processador, permitindo reduzir drasticamente a quantidade de processadores e o tempo de comunicação.

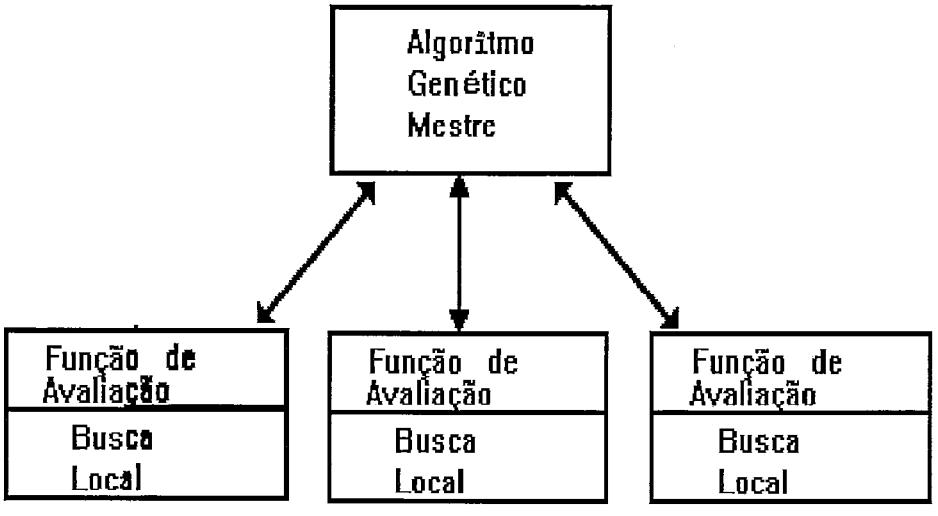

Fig. V.3: Modelo com Paralelização Heurística.

Cada busca local é efetuada na forma indicada na figura V.4, onde cada processador têm uma população local.

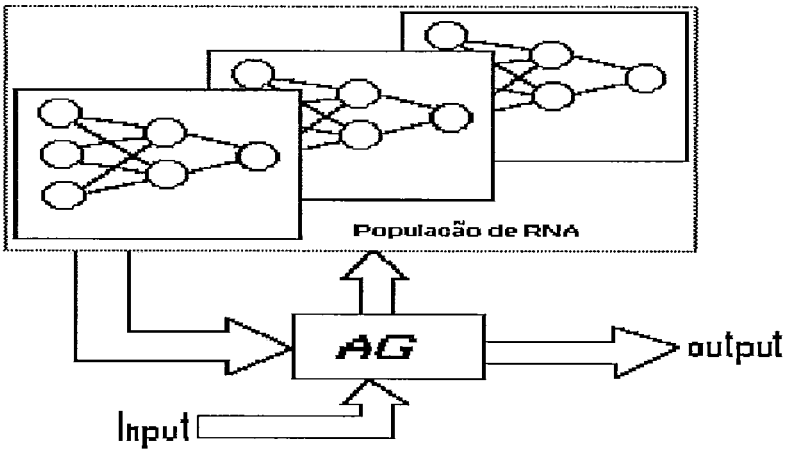

Fig. V.4: Operação do Modelo CP e CD.

V.4.1.- Modelo com População "completamente paralela e completamente distribuída"

A seguir apresenta-se a especificação do modelo com população completamente paralela e completamente distribuída (figura V.5).

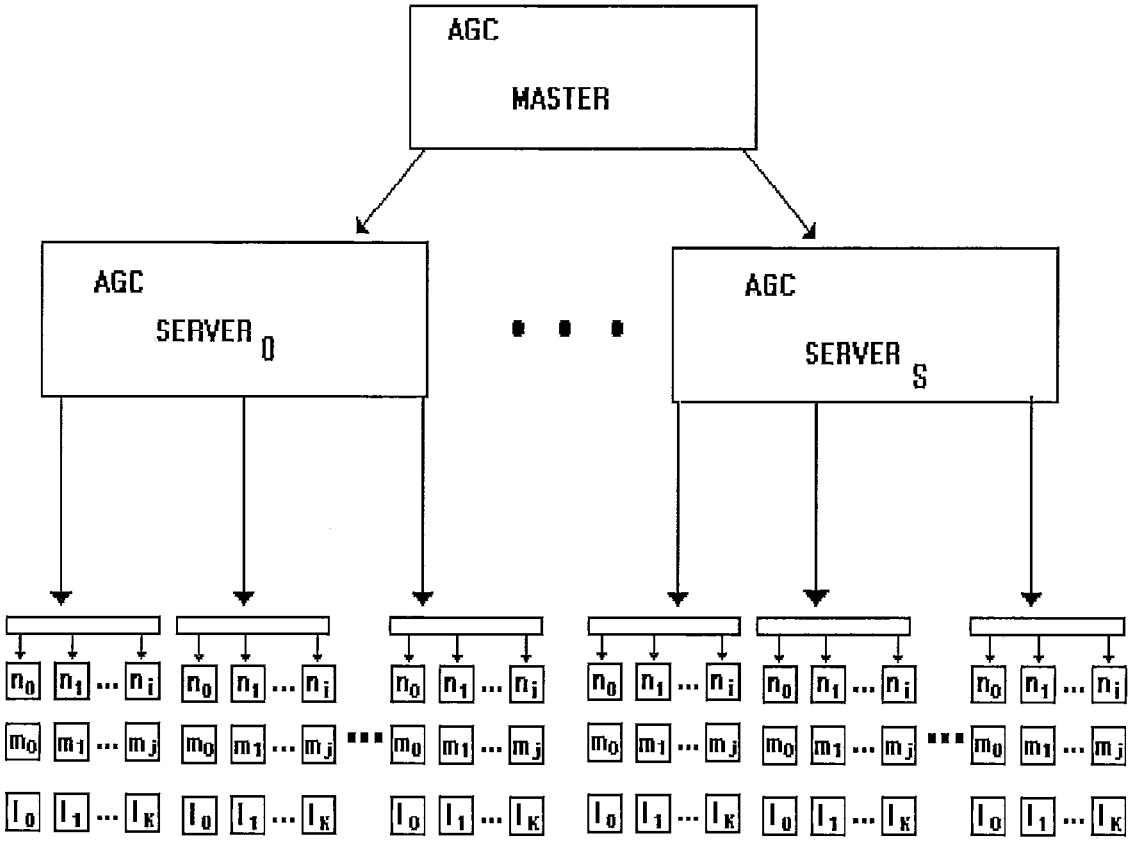

Fig. V.5: População Completamente Paralela e Distribuída.

V.4.1.1.- Operação do Sistema

O casamento, a concorrência das RNAs, a resolução dos conflitos de entrada, os operadores genéticos, e outros algoritmos de aprendizado são cada um deles aplicados em paralelo na população de RNA como um todo.

A operação do sistema é mostrada na figura V.6, onde se tem uma lista de entradas que é a comunicação com o meio ambiente. Esta lista de entradas possui as

mensagens de entradas de n bits, chamadas de Detectores. Através de um procedimento de casamento as condições desta lista são comparadas com os padrões das RNA, ativando desta forma algumas RNA, segundo seu grau de adaptação. Cada grau de adaptação é calculado a cada ciclo, castigando ou reforçando as RNAs que venceram o lance e colocaram mensagens neste ciclo. O castigo é usado para eliminar os piores indivíduos e para ajudar a prevenir que alguma RNA domine sempre. As RNAs competem pelo direito de colocar uma mensagem na lista de saídas, chamadas de Efetuadores. Os AGs são usados periodicamente para substituir alguma percentagem (parâmetro do sistema) da população de RNAs com descendentes que vêm de outras RNAs. Empiricamente têm-se provado que a percentagem da população para ser substituída não deve ser muito grande para dar tempo de estabilização à deteção da convergência; e a freqüência do uso dos AGs deve ser um número primo para evitar que os padrões se repitam ciclicamente.

O AG realiza a seguinte função em forma paralela:

a.- seleciona um conjunto de RNAs para serem substituídas e um conjunto de RNAs pais. As RNAs de menor grau de adaptação são escolhidas probabilisticamente de acordo com o inverso do quadrado do seu grau de adaptação (1/(grau de adaptação^2)). As RNAs de maior grau de adaptação são escolhidas probabilisticamente pelo quadrado do seu grau de adaptação (grau de adaptação^2);

b.- Com as RNAs selecionadas obtendo-se uma nova geração em paralelo com as operações genéticas para cruzar e mutar.

O algoritmo paralelo de seleção da adaptação aleatório opera da seguinte forma:

a.- gera um número aleatório entre 0 e o (grau de adaptação^2) para cada RNA;

b.- faz um "rank" destes números aleatórios ($\log_2 n$).

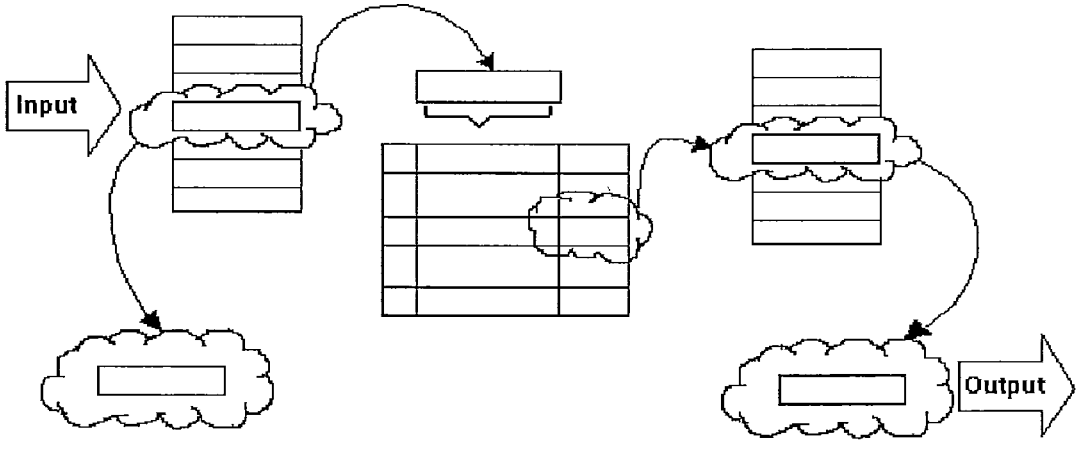

Fig. V.6: Casamento Paralelo.

O processo de casamento opera da seguinte forma:

a.- Cada mensagem da lista de M mensagens (detectores) é processada em forma seqüencial, mas é comparada em paralelo com todas as RNAs;

b.- A mensagem é comparada com os padrões de cada RNA da seguinte forma: é realizada a consulta ("recall") da mensagem com a RNA e é comparada com a saída desejada. Se ambos os resultados são iguais, quer dizer que casam, então num campo da RNA (chamado de mensagens vencedoras) é colocado em "1" o bit número i (onde i corresponde à posição da mensagem na lista de mensagens);

c.- Finalmente, no campo de mensagens vencedoras, tem-se para cada RNA, quais as mensagens que casaram com a RNA.

V.4.1.2.- Operadores Genéticos Paralelos

A união do cruzamento usa um algoritmo "rendezvous" para fazer com que cada geração aponte para o pai e vice-versa:

Algoritmo Rendezvous

- Enumera o conjunto de pais (tempo $O(\log_2 n)$), e envia (SEND) o endereço do processador de cada pai numa mensagem para uma variável rendezvous nos processadores enumerados. O SEND dos pais para os processadores enumerados vai resultar em ter uma variável rendezvous no processador 0 contendo o endereço do processador do primeiro pai (Fig. V.7);
- Enumerar o conjunto a ser substituído, logo cada descendência obtém (RECEIVE) o endereço na variável rendezvous dos processadores enumerados (esta enumeração é a mesma do "rank" da classificação, portanto pode-se usar essa, poupando tempo);
- Os descendentes enviam (SEND) o seu endereço para os pais diretamente (conhecem o endereço). A descendência é substituída por cópias dos seus pais. (Os pais enviam cópias deles mesmo para os seus descendentes). Algumas variáveis paralelas associadas com a descendência são simplesmente inicializadas, enquanto outras são copiadas dos pais. O grau de adaptação da descendência é a média entre o grau de adaptação do pai e a média da população:

$$P_{descendente} = (P_{médio} + P_{pai})/2$$

isto permite aos novos descendentes participarem do processo de oferta, sem dominar o processo.

O algoritmo de cruzamento permite a uma pequena percentagem da população (parâmetro do sistema) a realizar cruzamento nulo. Gera um número aleatório entre 0 e 100, e se estar abaixo de certo limite, então o algoritmo é usado, senão toma o valor do pai, que já foi copiado.

Para selecionar as parelhas, as RNAs são divididas em dois grupos (pares e ímpares). Assim, os pares apontam para os ímpares, e vice-versa. Gera-se um número aleatório entre 0 e o comprimento da mensagem para obter o ponto de cruzamento. Logo, é feito o intercâmbio entre ambas.

A mutação usa a tabela de distribuição de Poisson em paralelo para mutar 0, 1, 2 ou 3 gens do cromossoma. Gera-se um número aleatório entre 0 e 1000 para cada descendente. Um casamento com a tabela indica quantas mutações devem ser feitas. Para cada mutação, um número aleatório indica a posição da mutação e outro número aleatório indica o valor {0, 1, #}.

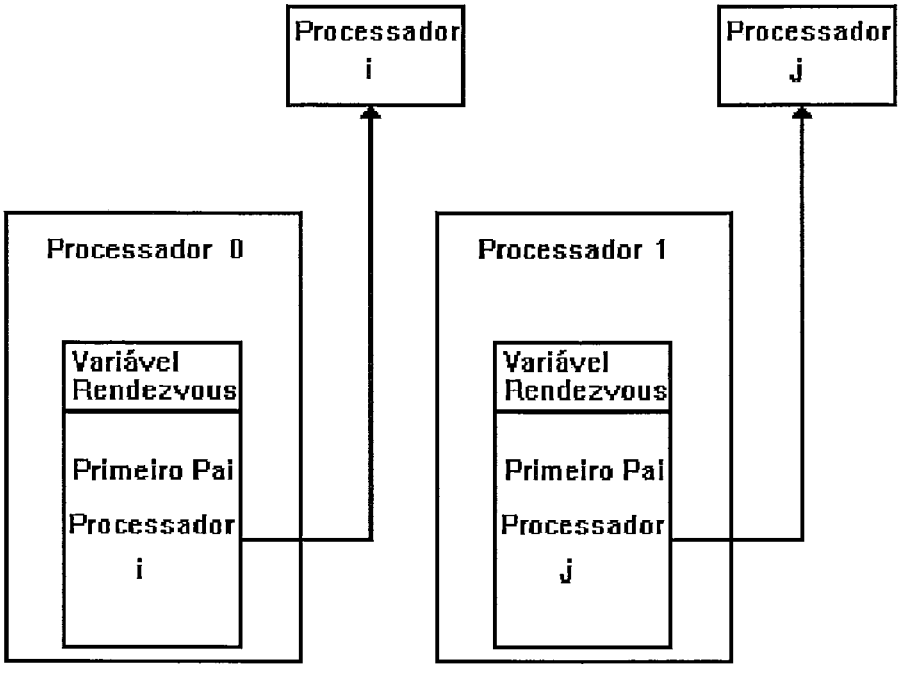

Fig. V.7: Obtenção do Endereço do Processador Pai.

Uma característica importante deste modelo é que o desempenho do sistema é independente do tamanho da população de RNAs, razão pela qual é usada a paralelização heurística em todos os modelos. Uma implementação pode explorar os mecanismos de comunicação do hipercubo iPSC, dado que pode usar as primitivas de SEND e RECEIVE explícitas e a enumeração e "rank" pode ser feita implicitamente. Isto indica também, que pode-se otimizar.

Este primeiro modelo apresentado pode-se chamar de "completamente paralelo e completamente distribuído", pelo fato de usar tantos processadores quanto forem necessários. A ordem de processadores necessários está dado pela ordem da RNA (O(N)), pela ordem do tamanho da população (O(n)), e pela ordem do tamanho do cromossoma (que depende da ordem da RNA, O(N)). Num modelo simplificado com uma RNA de tamanho O(N), uma análise simples para determinar a ordem dos processadores mostra a seguinte quantidade: Se a RNA está composta de N neurônios, então são necessários N processadores por cada RNA. Se a população total está composta de n RNAs, então são necessários nN processadores. Portanto, a ordem de processadores está dada por:

$$O(p) = O(n) \cdot O(N)$$

Uma análise do tempo necessário para comunicar dados entre os diversos processadores mostra que este depende do tamanho da mensagem. A quantidade de mensagens que devem comunicar-se é: cada processador elementar (neurônio) deve transmitir a sua saída para os outros (N/3) processadores da camada posterior, isto é $O(N^2)$ mensagens. Isto deve ser efetuado para cada comunicação entre camadas, ou seja $O(N^2)+O(N^2)$ mensagens, no caso de três camadas. O tamanho da mensagem que deve transmitir cada elemento processador é de um byte, portanto pode-se usar o modelo constante para obter o tempo total de comunicação entre os neurônios, e dado que os (N/3) neurônios de uma camada podem transmitir em forma paralela as suas mensagens, mas cada processador deve receber as mensagens, uma de cada vez, então o tempo de comunicação por camada é O(N).

Agora, cada RNA (cromossoma) deve enviar a sua resposta para o AGC, que vai determinar o desempenho das RNAs. O tamanho desta mensagem depende da ordem da RNA O(N), e são necessárias O(n) mensagens.

Desta análise simples do modelo com população "completamente paralela e completamente distribuída" obtém-se como conclusão que: é necessário uma grande quantidade de processadores; como também uma grande troca de mensagens entre os

processadores, embora o tamanho da maioria das mensagens seja de um byte, o tempo total de comunicação não justifica a implementação (embora irrealizável) deste modelo.

V.4.2. Modelo com Paralelização Topológica

Da análise anterior é possível verificar que não é aproveitada a máxima capacidade de todos os processadores disponíveis em forma paralela. Isto se deve a existência de um processamento que é claramente seqüencial, como no caso das diferentes camadas das RNAs. Não é possível começar o processamento de uma camada enquanto a camada anterior ainda não tem uma resposta, motivo pelo qual a grande maioria dos elementos processadores permanece ocioso grande parte do tempo. Esta característica permite mapear os neurônios das diferentes camadas num processador só (Figura V.8), o qual reduz significantemente a quantidade necessária de processadores como também a quantidade de intercâmbio de mensagens. Neste modelo se realiza uma partição topológica aproveitando a topologia da RNA, onde são necessários $O(N_i)$ processadores para cada RNA, onde N_i são os neurônios de entrada da RNA. O total de processadores é:

$$O(p) = O(N_i) \cdot O(n) \qquad N_i < N$$

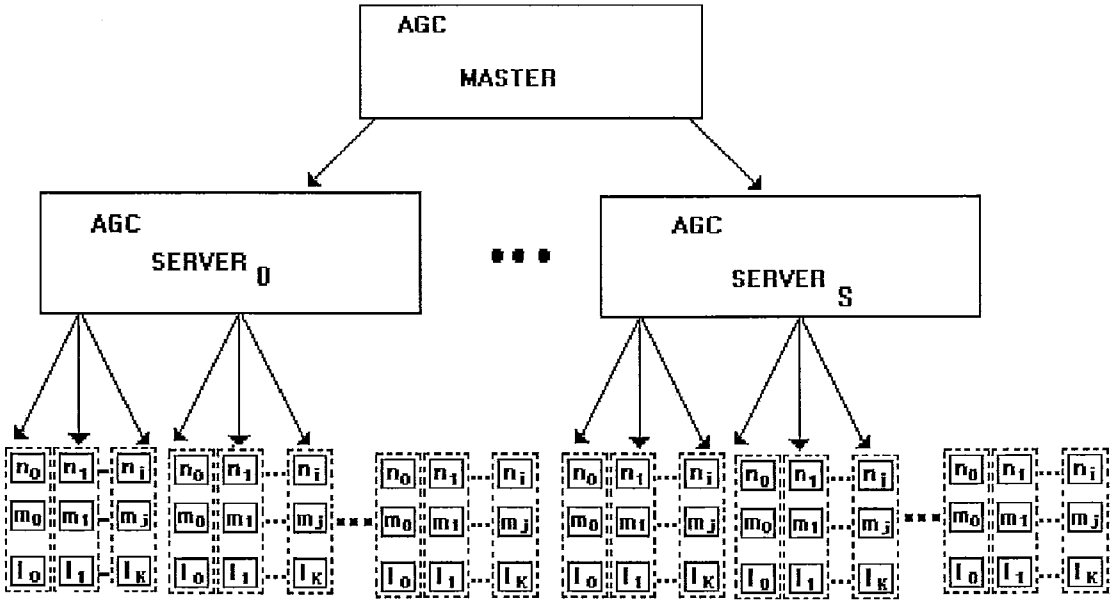

Fig. V.8: Modelo com Paralelização Topológica.

O tempo de comunicação entre processadores também se reduz, dado que não é necessário enviar mensagens entre as diferentes camadas, o que reduz em O(N) a

quantidade de mensagens entre os processadores. Neste modelo cada neurônio deve enviar (N_i-1) mensagens de tamanho de um byte para os seus vizinhos. O tempo total de comunicação entre os neurônios e o AG permanece igual ao do modelo "completamente paralelo e completamente distribuído", mas cada processador é melhor aproveitado, dado que não permanece ocioso tanto tempo como no modelo anterior. Ainda é um modelo que requer muitos processadores para a sua implementação.

V.4.3. Modelo de Paralelização pela Partição do Espaço de Busca

No modelo de paralelização pela partição do espaço de busca, simplesmente se aloca um processador por cada RNA, dado que cada RNA é um ponto no espaço de soluções. Se o tamanho da população é n indivíduos distribuídos em (S+1) AGC com uma população local de n/(S+1), são necessários (n+S+2) processadores. Cada AGC deve transmitir uma mensagem a cada RNA da população, portanto é necessária só uma mensagem, a qual é difundida para todas as RNAs locais. Uma vez feito o processamento localmente é necessário que o AGC colecione uma mensagem de cada RNA (n/(S+1) mensagens). A quantidade de mensagens se reduz fortemente, mas agora o tamanho da mensagem é de vários bytes (tamanho da camada de entrada). Dado que existem S+1 AGC em paralelo, não existe nenhuma restrição para S, além de que o processo mestre deve colecionar S+1 mensagens, para daí, escolher o melhor indivíduo e difundir para cada AGC. Este modelo é mais real, no que se refere a quantidade de processadores necessários.

V.4.3.1.- Estruturas de Dados Paralelos

Numa implementação paralela deve-se definir as estruturas de dados paralelos que são usadas pelos AGs paralelos: as RNAs e a lista de mensagens.

Cada processador possui a estrutura de dados de uma RNA, a qual contem variáveis do tipo: bit booleano, inteiros com sinal, inteiros sem sinal, etc., que representam os parâmetros de controle; e um campo que corresponde à representação da RNA num cromossoma, mostrado na figura V.9.

O mestre tem uma pequena lista de mensagens, a qual é processada em forma seqüencial. As operações com esta lista são: procedimento de casamento, criação de novas listas e ativação dos efetuadores.

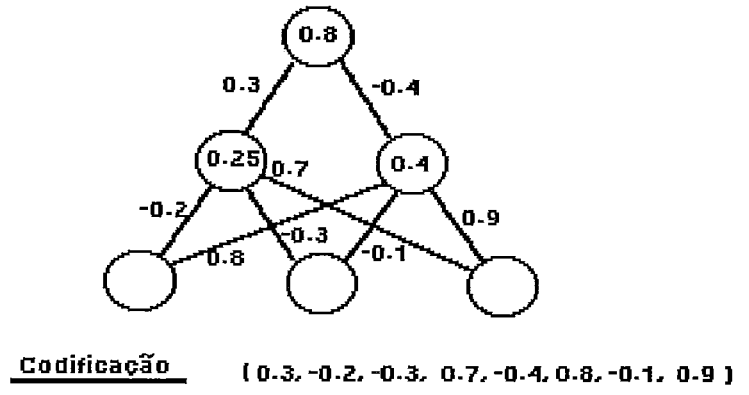

Fig. V.9: Estrutura de Dados do Cromossoma.

V.4.4.- Modelo de Paralelização Funcional

O AGC com paralelização funcional consiste simplesmente em dividir a população global de n indivíduos em (S+1) processadores. Cada processador trabalha localmente com uma população de tamanho n/(S+1). Neste modelo são necessários (S+1) processadores com uma capacidade de memória para manter n/(S+1) indivíduos. O algoritmo para cada nó do AGC é o seguinte:

ALGORITMO Nó_AG;
BEGIN
Inicialização;
Avaliação;
WHILE (faz nada)
BEGIN
Comunicação;
Seleção;
Recombinação;
Avaliação
END
END.

Neste modelo é possível identificar três protótipos, de acordo como é feito o controle dos AGC locais.

O protótipo Mestre-Servidor é mostrado na figura V.10, onde um processador atua como único mestre que coordena (S+1) processadores servidores.

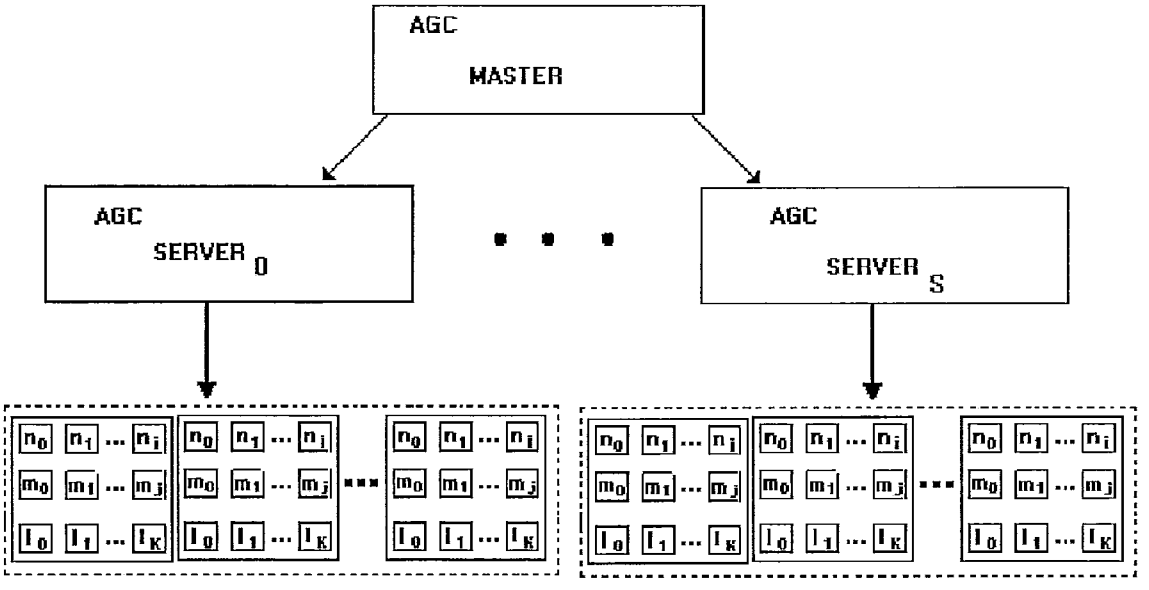

Fig. V.10: Paralelização Funcional com Mestre-Servidor.

O processo Mestre controla a seleção, casamento e desempenho dos operadores genéticos. Os servidores simplesmente efetuam as funções objetivo. O esquema é direto e relativamente fácil de implementar; mas sofre duas grandes desvantagens:

1.- Muita perda de tempo se existir uma variação muito grande no tempo das funções objetivo;
2.- O algoritmo não é muito confiável, dado que depende da saúde do processo mestre. Se o mestre morrer, o sistema pára.

A primeira destas desvantagens é resolvida pelo segundo protótipo, o Mestre-Servidor semi-síncrono. Este protótipo relaxa a necessidade das operações síncronas, inserindo e selecionando membros do sistema quando os servidores completam o seu trabalho. Este sistema opera melhor que o modelo de população superposta do DeJONG com um valor pequeno do intervalo de geração G. Este protótipo ainda depende de um processo só.

No protótipo de Rede (figura V.11), (S+1) AGs simples e independentes são executados com memórias independentes, operações genéticas independentes e funções objetivo independentes. Os (S+1) processos trabalham normalmente, com a exceção que os melhores indivíduos de uma geração são difundidos para as outras subpopulações através da rede de comunicação. Com uma comunicação relativamente intermitente, a

largura de banda das ligações é reduzida se comparado com os outros esquemas. A confiança deste esquema é alta, pela autonomia dos processos independentes.

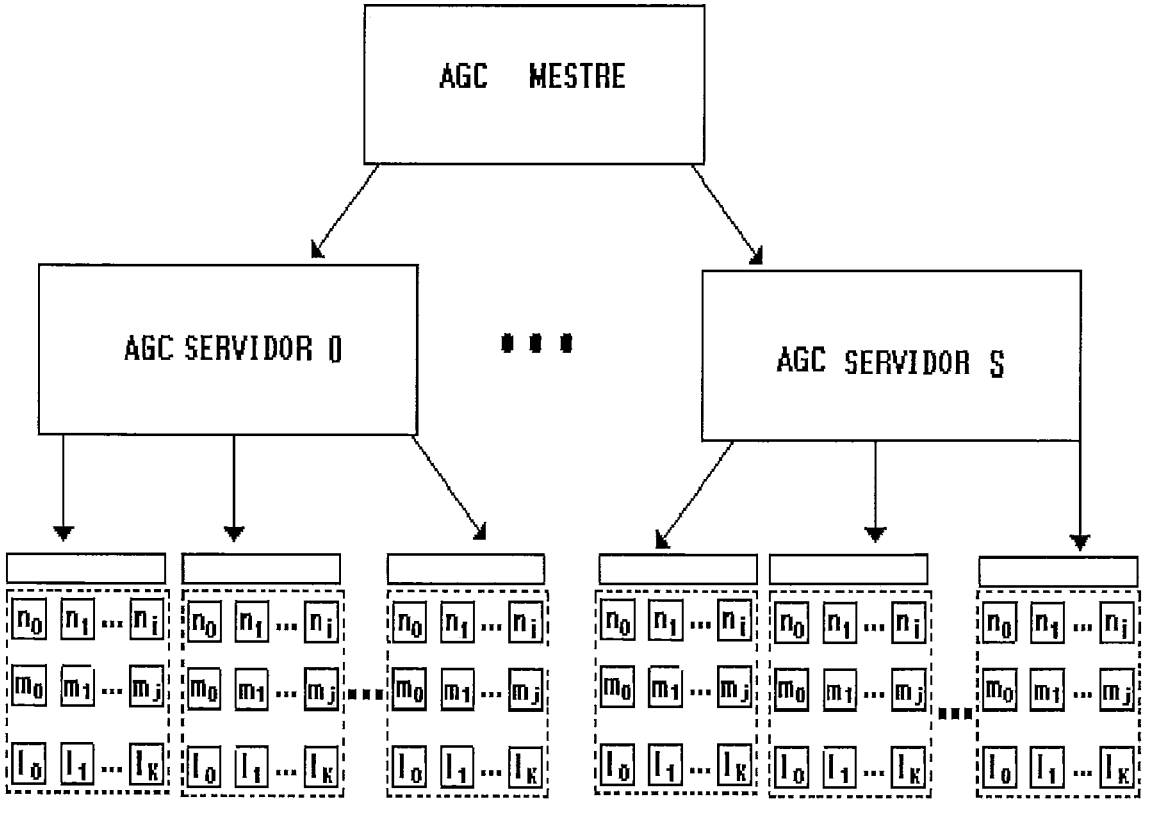

Fig. V.11: Modelo de Rede.

V.5. IMPLEMENTAÇÃO PARALELA DO ALGORITMO GENÉTICO CONEXIONISTA

Na seqüência se detalha o algoritmo do modelo com paralelização funcional, o qual opera segundo a figura V.12, cujos detalhes são apresentados nos itens a seguir. Neste modelo existe um processo mestre, o qual inicializa, coordena e supervisiona a busca da solução em cada processo servidor. No resto do texto se referirá ao processo mestre e aos processos servidores, simplesmente como mestre e servidores, respectivamente.

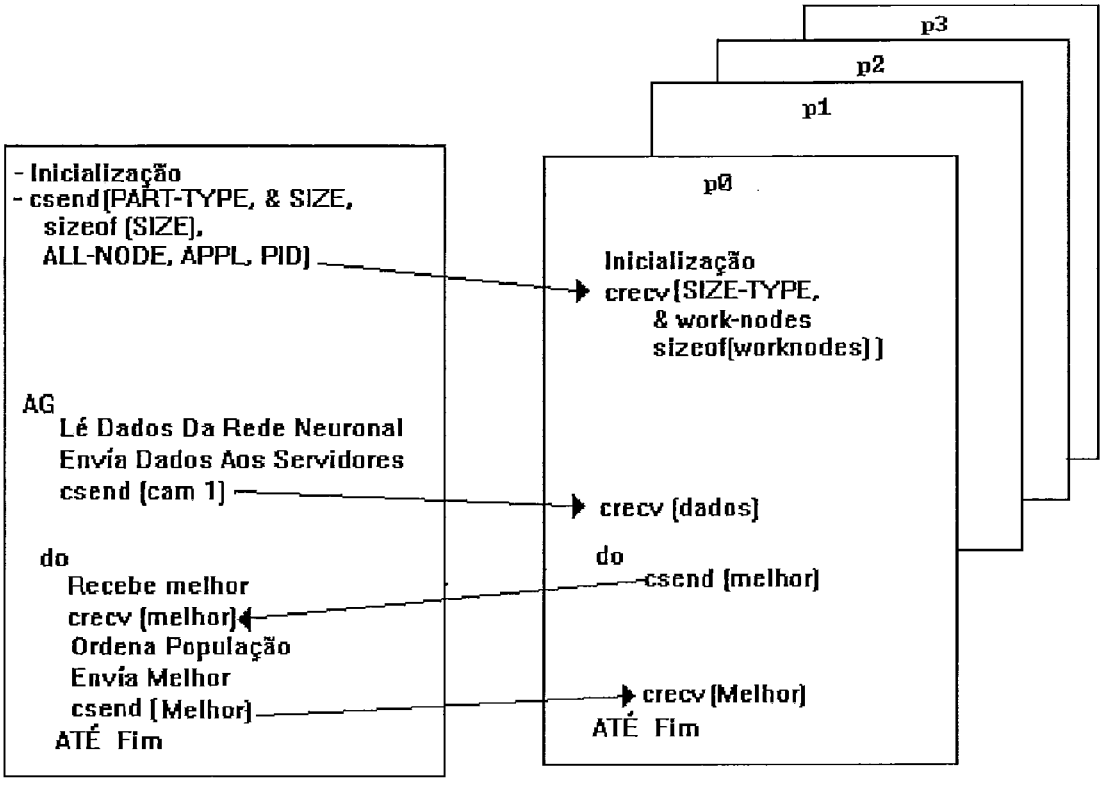

Fig. V.12: Algoritmo do Modelo com Paralelização Funcional.

V.5.1. Descrição do Processo Mestre

O algoritmo do processo mestre é como segue (no ANEXO B apresenta-se o código em C paralelo da implementação no simulador do hipercubo da iPSC/2 da INTEL):

```
main()
{
 Obtém Identificador Do Processo Mestre Na Rede;
 Carrega Cada Servidor Com Processo Servidor;
 Envia Mensagem a Servidores com número de nós na Rede;
 Repete
   Obtém Dados De Entrada Do Meio Ambiente;
   Repete
     Algoritmo Genético;
   Até Obter Solução;
   Aplica Solução Ao Meio Ambiente;
 Até FIM;
 Cancela Processos Servidores;
}
```

O mestre cumpre a função de inicializar os servidores, e cumpre papel interface com o usuário para obter a configuração inicial do sistema. Uma vez inicializado o sistema, se obtém a entrada do meio ambiente. O laço do Algoritmo Genético se repete até obter uma solução à entrada do meio ambiente. O ciclo se repete *ad infinitum*, dado que cada vez que não seja possível responder adequadamente à entrada do meio ambiente, então aplicam-se os operadores genéticos até obter uma solução.

O AG no mestre faz uma inicialização dos AGs servidores, enviando as características da configuração da Rede Neuronal. O procedimento é como segue:

```
AlgoritmoGenético()
{
Inicialização Variáveis Locais;
Lê Dados Da Rede Neuronal;
Envia Dados Aos Processos Servidores;
fazer {
   Recebe Melhor Indivíduo De Cada Servidor;
   Ordena População De Indivíduos Recebidos;
   Seleciona Melhor Indivíduo dos Melhores;
   Envia Melhor Indivíduo Para Cada Servidor;
} enquanto (fim processo != TRUE);
}
```

O laço do AG no mestre está sincronizado com os servidores, dado que uma vez inicializado cada servidor, o mestre fica no início do laço na espera de receber os melhores indivíduos de cada servidor. Cada servidor envia o seu melhor indivíduo para o mestre. A partir deste momento o mestre está em condição de fazer uma avaliação do melhor de todos os indivíduos. Esta seleção consiste simplesmente em fazer uma busca na população do melhor indivíduo adaptado, segundo seu campo de adaptação. Depois de selecionar o melhor de todos os indivíduos, o difunde para todos os servidores, voltando novamente para o início do laço. O laço do Algoritmo Genético se repete até que um servidor envie um sinal indicando que achou uma solução.

V.5.2. Descrição dos Processos Servidores

Os servidores tem como função receber o padrão de entrada enviado pelo mestre para realizar uma busca da solução aplicando os operadores genéticos. A busca no domínio genético consiste em aplicar o padrão recebido em cada RNA da população

local, para o qual é necessário transformar a representação cromossômica de cada RNA para o dominio neuronal. Uma vez no dominio neuronal é possível aplicar o padrão de entrada à RNA para obter a resposta desta (o erro). Esta resposta é usada pelo AG para determinar a capacidade de adaptação da RNA. Esse processo deve ser efetuado para todos os indivíduos da população local em cada servidor. Cada servidor efetúa um ciclo, selecionando o melhor indivíduo para enviá-lo ao mestre, e fica na espera do melhor de todos os recebidos pelo mestre. Isto permite compartilhar o melhor de todos entre os servidores. O ciclo se repete até achar a primeira solução. O algoritmo no servidor é:

```
main()
{
 Obtém Identificador do Processo; /* mypid() */
 Obtém identificador do Nó; /* mynode() */
 Recebe Mensagem do Mestre com Número de Nós na Rede;
 Repete {
   Se (Identificador do Nó < Número de Nós na Rede)
   Então Aplica Algoritmo Genético;
 }
}
```

O servidor é inicializado com os identificadores apropriados para determinar o processo e as mensagens que lhe correspondem. O ciclo do servidor se repete ad infinitum, aplicando em cada ciclo o AG, cujo algoritmo é:

```
Algoritmo Genético()
{
 Recebe Dados de Entrada;
 Inicializa Variáveis;
 fazer {
   Aplicar Operadores Genéticos;
   wg = (tamanho população)/2;
   para (i=0;(i<(tamanho população)/2);i++)
     copiar (nova população [i]) em (velha população [wg++]);
   Obter Estatísticas;
   Verificar Estabilidade;
   Enviar Mensagem com Melhor Indivíduo ao Mestre;
   Espera Mensagem do Mestre com Melhor Indivíduo;
 } enquanto (!(exista estabilidade)&&
         (geração<limite máximo));
 fim = TRUE;
 Envia Mensagem de Fim com a Solução achada;
}
```

O AG no servidor consiste em aplicar os operadores genéticos na população local. Logo substituem-se os melhores indivíduos da nova população pelos melhores

indivíduos da população velha. O melhor indivíduo da nova população é enviado para o Mestre, e fica na espera de receber o melhor indivíduo de todos os servidores. O laço se repete até atingir uma estabilidade ou até exceder um limite máximo de gerações. Quando acaba a busca, envia a solução numa mensagem indicando fim do processo.

A aplicação dos operadores genéticos responde ao seguinte algoritmo:

```
Aplicar Operadores Genéticos
{
 Ordenar População Velha pelo Desempenho;
 Fazer para toda a população {
   Selecionar 2 indivíduos para serem Pais;
   Aplicar Cruzamento (e mutação);
   Transformar Cromossomas Novos em Matriz de Conexões;
   /* Domínio Genético-->Domínio Neuronal */
   Simular RNA; /* para obter valor de adaptação da RNA */
   }
 Ordenar Nova População pelo Desempenho;;
}
```

O procedimento Transformar Cromossomas Novos em Matriz de Conexões cumpre com o objetivo detalhado em IV.2.1, onde um bloco de sinapses é uma estrutura criada para reunir coerentemente as sinapses que influenciam um determinado neurônio de saída. Agora, é necessário realizar a operação inversa, ou seja, transformar o cromossoma numa matriz de conexões da RNA para permitir a simulação da RNA. Esta operação se descreve a seguir pelo procedimento Domínio Genético-->Domínio Neuronal:

```
Domínio Genético-->Domínio Neuronal
{
 Limpa Matrizes;
 for (k=0;(k<número de neurônios na camada de saída);k++)
  for (j=0;(j<núm neurônios na camada intermediária);j++) {
     Peso2_3[k][j] = cromossoma[alelo++]);
     for (i=0;(i<núm. neurônios na camada entrada);i++)
       Peso1_2[j][i] += (cromossoma[alelo++]);
   }
 Calcula Peso médio em Peso1_2,
 segundo quantidade de neurônios de saída
}
```

Tendo a matriz dos pesos da RNA é possível realizar a simulação desta com

todos os padrões que sejam necessários. A simulação dá uma medida do comportamento da RNA com os padrões de entrada, que corresponde ao erro quadrático médio. Este valor é usado pelo AG para determinar o desempenho do cromossoma com o padrão atual.

V.5.3. Resultados Obtidos com o Modelo

Foram efetuados testes com quatro problemas:

(1) OU exclusivo com múltiplas entradas e uma saída;

(2) Inversão, onde a saída mostra a entrada invertida;

(3) Simetria com uma saída indicando se a entrada é simétrica;

(4) Auto-associatividade, onde a entrada aleatória é igual à saída.

Para cada problema se efetuaram 10 testes, obtendo a média.

V.5.3.1. Obtenção do Comprimento L ótimo

O primeiro teste efetuado consiste em fixar os parâmetros nos seguintes valores:

$$TP_{local} = 20 \text{ indivíduos}; \quad p = 4 \text{ processadores}$$

$$p_c = 0.9 \text{ (segundo fig. IV.3)}; \quad p_m = 0.02 \text{ (segundo fig. IV.4)}$$

O tamanho do cromossoma é determinado segundo:

$$L = n \cdot m \cdot (l+1) \cdot 8$$

O n está determinado pela saída da RNA. O l é um parâmetro que varia segundo o tamanho da entrada. Para cada l = (2, 3, 4); foram feito teste variando o m=(1..2l+2), obtendo-se como resultado as seguintes curvas (OU exclusivo em fig. V.13; inversão em fig. V.14; simetria em fig. V.15; associatividade em fig. V.16).

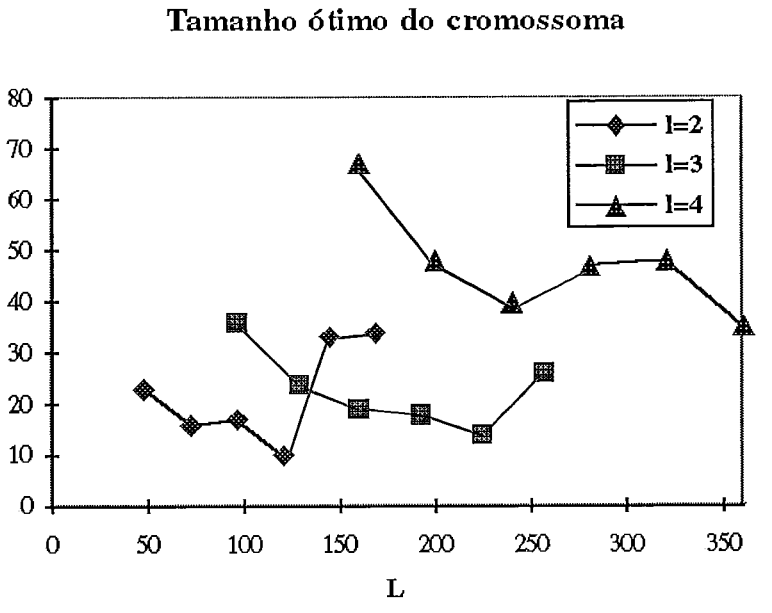

Fig. V.13: OU exclusivo

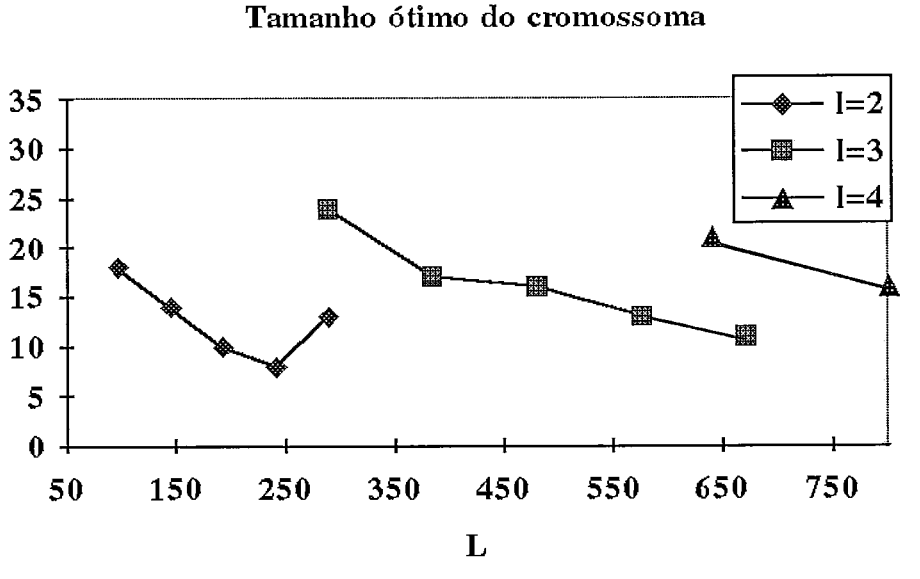

Fig. V.14: Inversão.

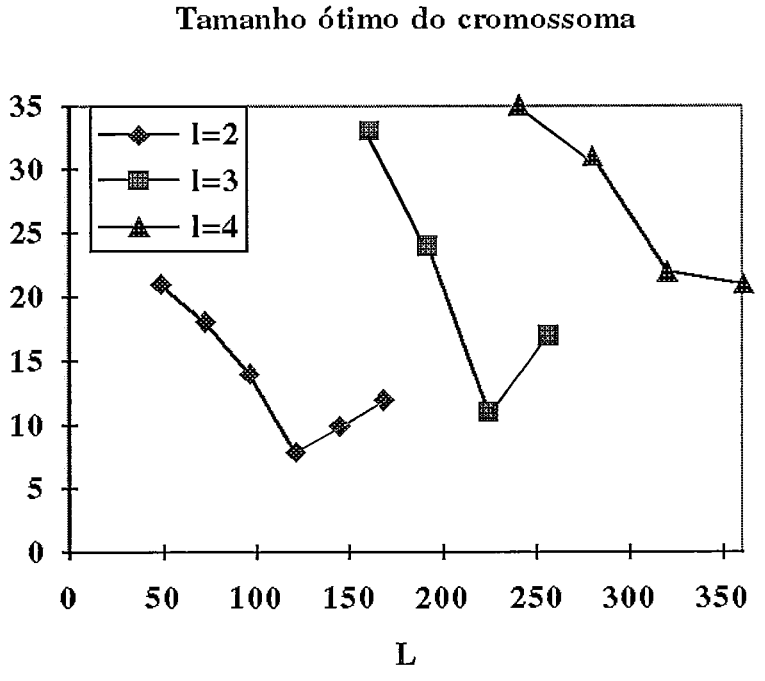

Fig. V.15: Simetria.

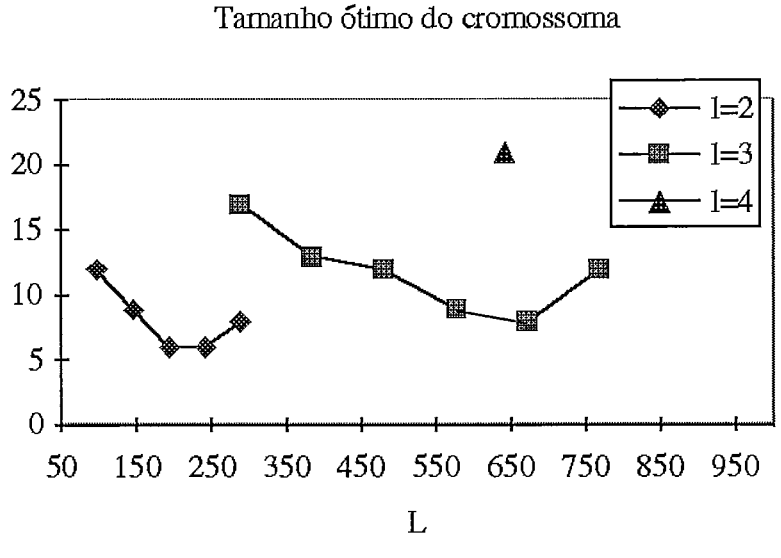

Fig. V.16: Associatividade.

Das curvas obtidas é possível verificar que em cada caso observado, o comprimento ótimo do cromossoma corresponde àquele, que em média resolveu os problemas em menor quantidade de gerações. Esse comprimento ótimo está de acordo

com o teorema de KOLMOGOROV, onde m=(2·l+1) em todos os casos. Para o resto dos testes feitos escolheu-se o comprimento ótimo obtido empiricamente das curvas acima.

Na seguinte curva (Fig. V.17) resume-se a relação entre o comprimento ótimo do cromossoma e gerações para resolver cada problema.

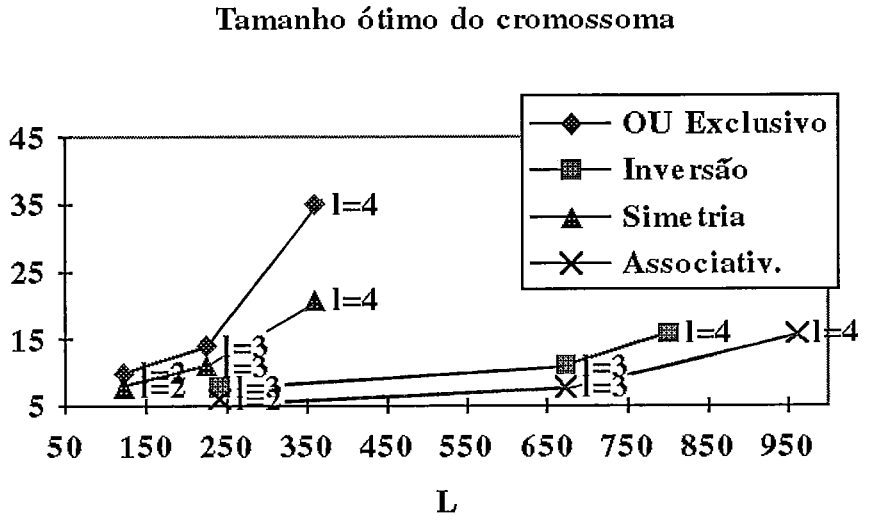

Fig. V.17: Relação Comprimento L com Geração.

Do resultado obtido na figura V.18 é simples verificar que o comportamento do modelo cumpre com: (1) para problemas mais dificeis, que envolvem maior quantidade de computação, a busca demora mais; (2) para cada comprimento ótimo L, com um TP fixo, a busca para cromossomas de maior comprimento requer maior computação.

V.5.3.2. Obtenção do TP ótimo

Para verificar o comportamento do modelo com diferentes tamanhos de população, fixou-se a entrada em n=3 para todos os casos, e os parâmetros em:

p = 4 processadores;

p_c= 0.9 (segundo fig. IV.3); p_m= 0.02 (segundo fig. IV.4)

Agora para cada problema varia-se o TP_{local}=(10,20,30,40,50) indivíduos, obtendo-se a seguinte curva da Fig. V.18.

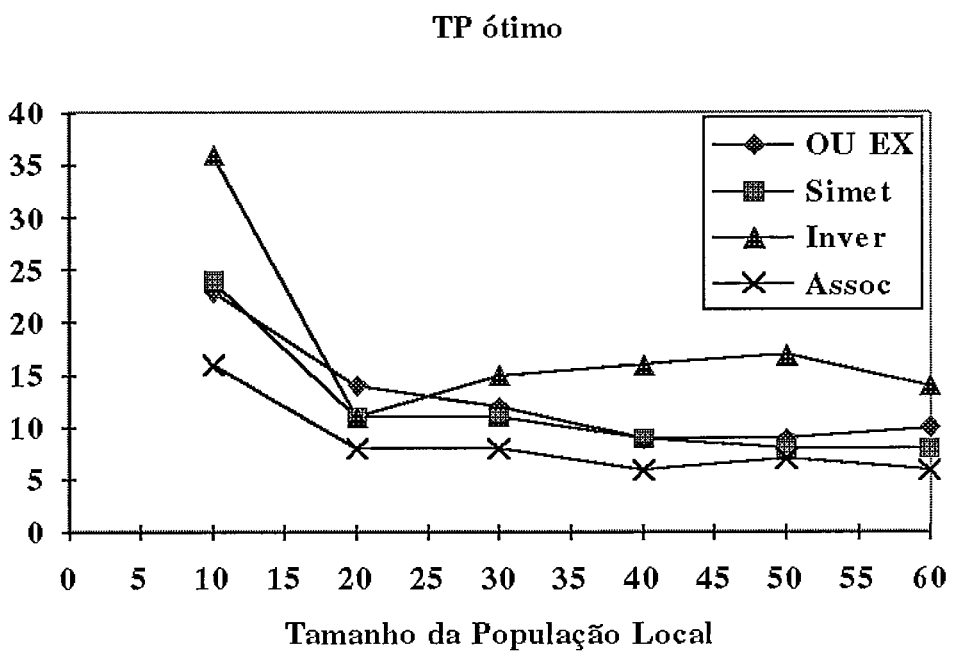

Fig. V.18: Obtenção do TP ótimo.

Dos resultados obtidos é possível concluir que:

(1) Para os problemas de OU-Exclusivo e Simetria, o comprimento do cromossoma ótimo (segundo curvas das figs. V.13 e V.14), é 224, indicando um espaço de busca de 2^{224}. Para os problemas de Inversão e Associatividade, o comprimento ótimo do cromossoma é 672, cujo espaço de busca é ainda maior, 2^{672}. Partindo deste fato é possível concluir que os primeiros dois problemas devem atingir a solução mais rápido que os outros dois problemas, supondo que o TP inicial para cada caso depende do comprimento do cromossoma. Da fig. V.18 é possível concluir que não se cumpre a teoria de GOLDBERG. O motivo disto é que os problemas resolvidos com o AGC não depende do comprimento do cromossoma.

(2) Também é possível verificar que independente do TP, todos os problemas têm tendência a atingir a solução no máximo no período de estabilização da RNA, o qual neste trabalho foi determinado em 5 gerações.

V.5.3.3. Análise para Processamento em Tempo Real

O simulador iPSC/2 da INTEL foi usado para depurar o modelo e fazer uma avaliação da comunicação necessária entre os processadores. Nesta etapa, não foi

possível fazer um levantamento do tempo de processamento do modelo. Para avaliar o desempenho do modelo, os tempos foram obtidos da implementação na Rede Hipercúbica de Transputers. Os resultados obtidos se mostram na fig. V.19.

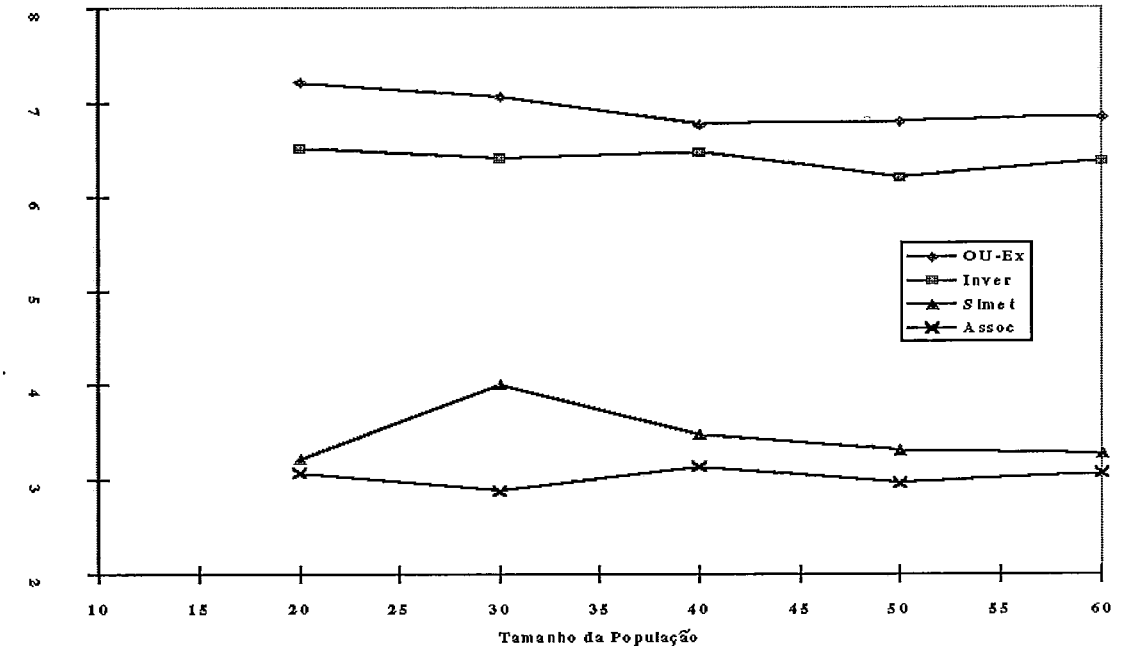

Fig. V.19: Tempo de Processamento do Modelo AGC.

O tempo de processamento mostrado na fig. V.19 corresponde à média de 10 testes feitos com cada problema. É possível verificar que o incremento do TP com 4 processadores (Rede de Transputer) não diminui o tempo de processamento em forma importante.

Os tempos de resposta obtidos no modelo parecem ser interessantes de serem aplicados em controle de processos em tempo real, dado que existe um grande escopo de aplicações, onde um tempo de resposta entre 5 e 10 segundos ainda é bom.

CAPÍTULO VI

CONCLUSÕES E PESQUISAS FUTURAS DO MODELO PARALELO

Os AGs proveêm um conjunto de busca heurística eficiente e independente do domínio para aplicações num largo escopo de desafios da engenharia. As implementações dos modelos tradicionais para otimização, busca e IA requerem uma grande quantidade de conhecimento explícito e específico do domínio. Estes sistemas clássicos são fracos, pela considerável necessidade da intervenção humana para compensar pequenas alterações no comportamento do domínio, o qual é inaceitável para controle de processos em tempo real.

Os AGs têm duas características inerentes que os fazem altamente apropriados para o aprendizado por descoberta: mantêm uma variabilidade suficientemente alta para prevenir a convergência para ótimos locais; e a categorização e recombinação produz um alto paralelismo implícito.

Da implementação na Rede de Transputers do modelo AGC apresentado pode-se dizer que se verifica que o paralelismo inerente é totalmente explorado em máquinas altamente granularizadas (mostrado pela independência do tempo com o tamanho da população), e que a velocidade depende do tamanho da lista de mensagens e é linear, dado que o processamento desta lista de entrada é seqüencial. Mostra a importância da programação paralela de dados.

Uma das principais vantagens do AGC é o pequeno requerimento de memória que ele precisa, dado que trabalha com um tamanho local da população, e a partir daí, vai gerando as possíveis novas soluções substituindo aqueles indivíduos que não tem um bom desempenho.

As aplicações de busca e otimização são problemas bem definidos, com funções objetivos conhecidas, onde as variáveis de decisão proveêm um meio simples para avaliar as alternativas do meio ambiente. Em contraste com estes problemas, os problemas de aprendizado automático são mal definidos, cuja avaliação tem critérios subjetivos e múltiplas opções de decisão, isso tudo, constitui um meio ambiente difícil de lidar para comparar ou analisar.

VI.1 CONCLUSÕES

O uso de RNAs como uma solução alternativa nos sistemas de controle apresentam basicamente dois problemas:

(1) Um problema surge quando a RNA (na fase de consulta) não consegue identificar ou classificar um padrão de entrada, onde a intervenção humana não é aceitável para processo de controle num meio ambiente dinâmico;

(2) Outro problema consiste no tempo de resposta de uma RNA, dada a grande quantidade de processsamento envolvido.

As soluções propostas neste trabalho são:

(1) O primeiro problema acima mencionado é superado pelo uso de algoritmos de otimização no aprendizado, os Algoritmos Genéticos, existindo algumas pesquisas nesta área, mas sem se preocupar com o tempo de resposta do modelo;

(2) A solução acima resolve o problema do aprendizado, mas pelo fato de operar com uma família de RNAs, o tempo de processamento desta solução é muito elevado, para o qual é indispensável o uso de computação paralela;

(3) Do estudo efetuado com os diversos modelos paralelos, conclui-se que o modelo mais adequado em termos de tempo de comunicação e processamento para máquinas paralelas reais (com um número limitado de processadores), é o modelo com paralelização funcional.

Os resultados obtidos mostram as seguintes conclusões interessantes de destacar:

(1) O modelo paralelo do AGC é independente do TP, o qual permite usar um TP relativamente pequeno, poupando espaço em memória;

(2) O comprimento ótimo do cromossoma (L) está de acordo com o Teorema de KOLMOGOROV;

(3) O número de gerações para atingir a convergência é independente do comprimento do cromossoma. Segundo a Teoria de GOLDBERG, o comprimento L determina

o espaço de busca, e o espaço de busca determina o TP inicial ótimo. Neste trabalho mostra-se que a convergência depende da dificuldade do problema a ser resolvido com a RNA, e é independente do comprimento do cromossoma;

(4) A conclusão mais importante, que prova a viabilidade do modelo paralelo AGC para seu uso em controle de processos, obtém-se da figura VI.1. Nesta figura mostram-se três modelos de AGC com parâmetros idênticos para resolver o OU Exclusivo de 2 entradas:

a. Modelo seqüencial com 3 camadas;
b. Modelo seqüencial com Enlace Funcional [PAO, 1989];
c. Modelo paralelo com 4 processadores.

O tempo de resposta do modelo paralelo é melhor em todos os casos. O tempo obtido mostra a viabilidade de aplicar este modelo paralelo AGC em controle de processos.

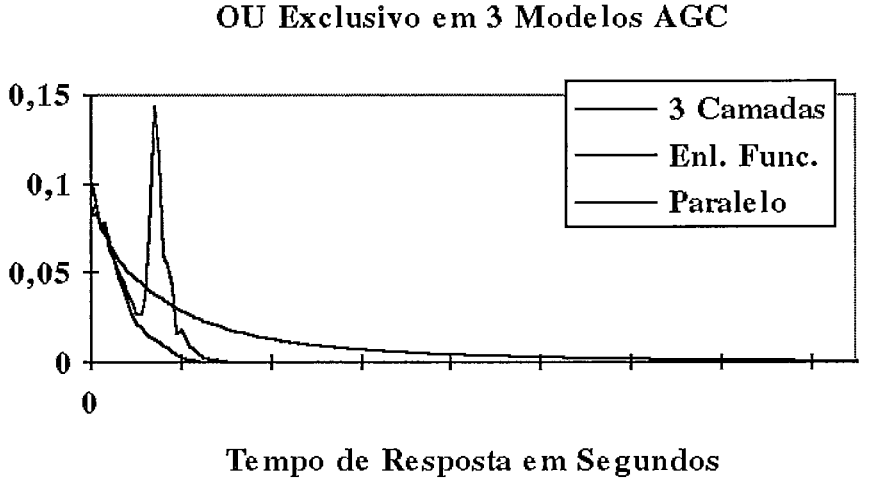

Fig. VI.1: Tempo de Resposta de Diferentes Modelos

VI.2 TRABALHO FUTURO A EFETUAR

Como um projeto interessante e promisor propõe-se a expansão da teoria, desenvolvendo uma matemática consistente que trate a interação entre os AG e as RN, resultando com isso, um largo escopo de aplicações para os sistemas adaptativos. É

interessante também, desenvolver um modelo matemático que permita prognosticar o desempenho do sistema no aprendizado automático.

A evolução do AGC em outras arquiteturas de RNAs podem trazer bons resultados, como os apresentados pela rede com enlace funcional, e a implementação de novas técnicas paralelas podem melhorar o desempenho de novos operadores genéticos.

Outra pesquisa interessante é a proposição de uma metodologia de desenvolvimento de AGC para sistemas de aprendizado em tempo real. Isto permitiria desenvolver uma ferramenta de Meta-Algoritmos Genéticos, onde o usuário pode definir as entradas do meio ambiente (detectores), as saídas para o meio ambiente (efetuadores), e eventualmente a função de avaliação (objetivo) para determinar o desempenho do sistema no meio ambiente.

REFERÊNCIAS BIBLIOGRÁFICAS

ACKLEY, D.H. (1987), An Empirical Study of Bit Vector Function Optimization, in: L. DAVIS (Ed.), Genetic Algorithms and Simulated Annealing, Morgan Kaufmann Publishers, Los Altos, CA, pag. 170-204.

AMORIM, C.L.; V.C. BARBOSA & E.S. FERNANDES (1988), Uma Introdução à Computação Paralela e Distribuída, VI Escola de Computação Brasileira, Campinas, São Paulo, Brasil.

ANDERSON, A. (1990), *Multiple Processing*, Fleet Publications.

ANDERSON, J.A. (1990), Data Representation in Neural Networks, *AI Expert*, June, pag. 30-37.

ANDERSON, C.W. (1988), Learning to Control an Inverted Pendulum Using Neural Networks, *IEEE Control Systems Magazine*, April, pag. 31-36.

ANDREWS, W. (1990), Application determines Best Approach to Multiprocessing, *Computer Design*, December 1.

ANGUS, I. & G.C. FOX (1990), *Solving Problems on Concurrent Processors, Volume II*, Fleet Publications.

ARBIB, M.A. (1989), Schemas and Neural Networks for Sixth Generation Computing. Invited Survey, *Journal of Parallel and Distributed Computing*, vol. 6, num. 2, April, pag. 185-216.

AUGUST, M.C.; G.M. BROST; C.C. HSIUNG & A.J. SCHIFFLEGER (1989), Cray X-MP: The Birth of a Supercomputer, *IEEE Computer*, January, 45-52.

AUSTIN, S. (1990), An Introduction to Genetic Algorithms, *AI Expert*, March, 49-53.

BAILEY, D. & D. THOMPSON (1990), How to Develop Neural Network Application, *AI Expert*, June, pag. 38-47.

BARTO, A.G.; R.S. SUTTON & C.W. ANDERSON (1983), Neuronlike Adaptative Elements that can solve Difficult Learning Control Problems, *IEEE Trans. on Systems, Man & Cybernetics*, vol. SMC-13, num. 5, Sept./Oct.

BAVARIAN, B. (1988), Introduction to Neural Networks for Intelligent Control, *IEEE Control Systems Magazine*, April, pag. 3-7.

BEN-ARI, M. (1982), *Principles of Concurrent Programming*, Prentice-Hall.

BOGOCH, S.; I. BASON; J. WILLIAMS & M. RUSSELL (1990), Supercomputers Get Personal, *BYTE*, May, pag. 231-240.

Borland International, (1988a), *Turbo C version 2.0 User's Guide.*

Borland International, (1988b), *Turbo C version 2.0 Reference Guide.*

BOOKER, L.B. (1987), Improving Search in Genetic Algorithms, in: L. DAVIS (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, pag. 61-73.

BOOKER, L.B.; D.E. GOLDBERG & J.H. HOLLAND (1989), Classifier Systems and

Genetic Algorithms, *Artificial Intelligence*, Vol. 40, Sept., pag. 235-282.

BURNS, A. (1988), *Programming in OCCAM 2*, Addison-Wesley.

CARBONELL, J. (1989), Introduction: Paradigms for Machine Learning, *Artificial Intelligence*, Vol. 40, pag. 1-9.

CARPENTER, G.A. & S. GROSSBERG (1988), The ART of Adaptative Pattern Recognition by a Self-Organizing Neural Network, *Computer*, March.

CARUANA, R.A. & J.D. SCHAFFER (1988), Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms, in J. LAIRD (Ed.), *Proc. of the Fifth Int. Conf. on Machine Learning*, Univ. of Michigan, pag. 153-161.

CHIOU, R.C.H. (1991), Simulação Paralela e Distribuida de Redes Neuronais para Percepção Visual de Imagens Naturais, *Tese M.Sc. COPPE/UFRJ*, Maio.

COHEN, M.A. & S. GROSSBERG (1983), Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks, *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-13, num. 5, September/October.

Computer IEEE (1987), Interconnection Networks, número especial, June.

DAVIDSON, D.B. (1990), Parallel Processing Tutorial, *IEEE Antennas and Propagation Society Magazine*, April.

DAVIS, L. (1987), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA.

DAVIS, L. & D.K. YOUNG (1988), Classifier Systems with Hamming Weights, in J.E. LAIRD (Ed.), *Proc. of the Fifth Int. Conf. on Machine Learning*, Univ. of Michigan, pag. 162-173.

DAVIS JR., G.W. (1989), Sensitivity Analysis in Neural Net Solutions, *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, num. 5, Sept./Oct.

DeJONG, K. (1980), Adaptative System Design: A Genetic Approach, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-10, Num. 9, Sept., pag. 566-574.

DENIS, F.A.R.M. & R.J. MACHADO (1991), O Modelo Conexionista Evolutivo, *Relatório Técnico CCR-128 IBM Centro Científico Rio* Rio, Brasil.

DINNING, A. (1989), A Survey of Synchronization Methods for Parallel Computers, *IEEE Computer*, July, 66-76.

DRESS, W.B. (1987), Frequency-Coded Artificial Neural Networks: An Approach to Self-Organizing Systems, *Proc. of the IEEE First Annual Int. Conf. on Neural Networks*, 2, University of California, San Diego, pag. 25-54.

ECKMILLER, R. (1988), Neural Nets for Sensory and Motor Trajectories, *IEEE Control Systems Magazine*, April, pag. 53-58.

FALK, H. (1988), AI Techniques enter the realm of Conventional Languages, *Computer Design*, October 15, pag. 45-49.

FELDMAN, J.A.; M.A. FANTY; N.H. GODDARD & K.J. LYNNE (1988), Computing with Structured Neural Networks, *Com. of the ACM*, Vol. 31, Num. 2, February, pag. 170-187.

FOULK, P. (1990), *CAD of Concurrent Computers*, Fleet Publications.

FOX, G. (1988), *THE THIRD CONFERENCE ON Hypercube Concurrent Computers and Applications, Volume 1 and 2. Architecture, Software, Computer Systems and General Issues*, ACM, California Institute of Technology.

FOX, G.C.; M.A. JOHNSON; G.A. LYZENGA; S.W. OTTO; J.K. SALMON & D.W. WALKER (1988), *Solving Problems on Concurrent Processors, Volume I, General Techniques and Regular Problems*, Prentice Hall, New Jersey.

FRAGNIAUD, (1991), comunicação pessoal.

FREY, P.W. (1986), A Bit-Mapped Classifier, *BYTE*, November, pag. 161-172.

FRIED, S.S. (1991), Personal Supercomputing with the Intel i860, *BYTE*, January.

FUKUSHIMA, K. (1975), Cognitron: A Self-Organizing Multilayered Neural Networks, *Biol. Cybernet.*, 20, pag. 121-136.

FUKUSHIMA, K. & S. MIYAKI (1982), Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position, *Pattern Recognition*, 15, (6), pag. 455-469.

FUKUSHIMA, K.; S. MIYAKI & T. ITO (1983), Neocognitron: A Neural Network Model for Mechanism of Visual Pattern Recognition, *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-18, num. 5, Sept./Oct., 826-834.

GALLANT, S.I. (1988), Connectionist Expert Systems, *Com. of the ACM*, Vol. 31, Num. 2, pag. 152-168.

GIBBONS, A. & W. RYTTER (1988), *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge.

GILBERT, D. (1987), Current trends in Parallel Processing and Supercomputing, *Belgian Institute for Automatic Control*, November, Antwerp.

GOLDBERG, D.E. (1985), Optimal Initial Population Size for Binary-Coded Genetic Algorithms, *Technical Report TCGA Num. 85001*, The Clearinghouse for Genetic Algorithms, University of Alabama.

GOLDBERG, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publ. Co.

GOLDSCHLAGER, L. (1978), A unified Approach to Models of Synchronous Parallel Machines, *Proc. of the Symposium on the Theory of Computing*.

GOLES, E. & S. MARTINEZ (1990), *Neural and Automata Networks: Dynamical Behavior and Applications*, Kluwer Academic Publisher.

GREFENSTETTE, J.J. (1986), Optimization of Control Parameters for Genetic Algorithms, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-16,

Num. 1, Jan.-Feb., pag. 122-128.

GROSSBERG, S. (1976), Adaptive Pattern Classification and Universal Recording, *Biol. Cybernet.*, 23, pag. 121-134.

GROSSBERG, S. (1987), Competitive Learning: From Interactive Activation to Adaptive Resonance, *Cognitive Science*, 11, pag. 23-63.

GROSSBERG, S. & M. KUPERSTEIN (1986), *Neural Dynamics of Adaptive Sensory-Motor Control: Ballistic Eye Movements*, Elsevier/North Holland.

GUEZ, A.; J.L. EILBERT & M. KAM (1988), Neural Networks Architecture for Control, *IEEE Control Systems Magazine*, April, Vol. 8, num. 2, 22-25.

HAYES, E.F. (1984), Access to Supercomputers: an NSF Perspective, *Com. of the ACM*, 27, num 4, 299-303, April.

HAYES-ROTH, F. & J. McDERMOTT (1978), An Interference Matching Technique for Inducing Abstractions, *Com. of the ACM*, Vol. 21, Num. 5, May, pag. 401-411.

HEBB, D.O. (1949), *The Organization of Behavior*, Wiley, New York.

HECHT-NIELSEN, R. (1987), Neurocomputer Applications, *National Computer Conference*, pag.239-244.

HECHT-NIELSEN, R. (1990), *Neurocomputing*, Addison-Wesley Pu. Co.

HEISTERMANN, J. (1991), Learning in Neural Nets by Genetic Algorithms, in: R. ECKMILLER, G. HARTMANN & G. HAUSKE (Ed.), *Parallel Proc. in Neural Systems and Computers*, North-Holland, Amsterdam, pag. 165-168.

HILLIS, W.D. (1986), *The Connection Machine*, Cambridge, Mass.: MIT Press.

HILLIS, W.D. & G.L. STEELE Jr. (1986), Data Parallel Algorithms, *Com. of the ACM*, 29, num 12, pag. 1170-1184, December.

HILLMAN, D.V. (1990), Integrating Neural Nets and Expert Systems, *AI Expert*, June, pag. 54-59.

HINTON, G.E. (1985), Learning in Parallel Networks, *BYTE*, April, pag. 265-273.

HINTON, G.E.; J.L. McCLELLAND & D.E. RUMELHART (1986), Distributed Representations, in: D.E. RUMELHART & J.L. McCLELLAND (Eds), *Parallel Distributed Processing, I: Foundations*, (MIT Press, Cambridge).

HINTON, G.E. (1989), Connectionist Learning Procedures, *Artificial Intelligence*, Vol. 40, September, pag. 185-234.

HINTON, G.E.; T.J. SEJNOWSKI & D.H. ACKLEY (1984), Boltzmann Machines: Constraint Satisfaction Networks that Learn, *Cognitive Science*, 9, 147-169.

HIRAMATSU, A. (1989), ATM Communications Network Control by Neural Network, *Proc. of the IEEE Third Int. Conf. on Neural Networks*, Vol. 1, San Diego, California, pag. 259-266.

HOCKNEY, R.W. & C.R. JESSHOPE (1988), *Parallel Computers 2, Architecture,*

Programming and Algorithms, Adam Hilger, Bristol and Philadelphia.

HOLLAND, J.H. (1986), Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms applied to Parallel Ruled-Based Systems, in: R.S. MICHALSKI; J.G. CARBONELL & T.M. MITCHELL (Eds), *Machine Learning: An Artificial Intelligence Approach. Volume II*, (Morgan Kaufmann, Los Altos, CA), pag. 593-623.

HOPFIELD, J.J. (1982), Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proc. of the Nat. Academy of Science USA*, Vol. 79, pag. 2554-2558.

HORD, M. (1990), *Parallel Supercomputing in SIMD Arquitecture*, Fleet Publications.

HWANG, K. & F.A. BRIGGS (1984), *Computer Architecture and Parallel Processing*, New York, McGraw-Hill.

HWANG, K. (1989), *Parallel Processing for Supercomputing and Artificial Intelligence*, Fleet Publications.

INMOS Limited (1987), *IMS T800 Transputer*.

INMOS Limited (1988a), *Occam 2 Reference Manual*, Prentice Hall.

INMOS Limited (1988b) *Transputer Development System*, Prentice Hall.

INMOS Limited (1988c) *Transputer Reference Manual*, Prentice Hall.

INTEL CORPORATION (1988) *iPSC Simulator Manual*, Intel Scientific Computers.

INTEL CORPORATION (1988) *iPSC/2 C Programmer's Reference Manual*, Intel Scientific Computers.

INTEL CORPORATION (1988) *iPSC/2 User's Guide*, Intel Scientific Computers.

KARTACHEV, S. (1990), *Supercomputing Systems*, Fleet Publications.

KAWATO, M.; Y. UNO; M. ISOBE & R. SUZUKI (1988), Hierarchical Neural Networks Model for Voluntary Movement with Application to Robotics, *IEEE Control Systems Magazine*, April, pag. 8-15.

KERNIGHAN, B.W. & D.M. RITCHIE (1978), *The C Programming Language*, Prentice-Hall, Inc.

KHANNA, T. (1990), *Foundations of Neural Networks*, Addison-Wesley Pub. Co.

KIRCKPATRICK, S.; C.D. GELLAT Jr. & M.P. VECCHI (1983), Optimization by Simulated Annealing, *Science*, 220, pag. 671-680.

KEATING, C. (1990), A Fearful Symmetry, *BYTE*, May, pag. 221-230.

KIRNER, C. & S.B.T. MENDES (1988), *Sistemas Operacionais Distribuídos, Aspectos Gerais e Análise de sua Estrutura*, Editora Campus, Brasil.

KIRNER, C (1989), *Sistemas Operacionais para Ambientes Paralelos*, IX Congresso da Sociedade Brasileira de Computação, Uberlândia, MG, Brasil.

KOHONEN, T. (1982), Clustering, Taxonomy and Topological Maps of Patterns. In M. LANG (ed.), *Pattern Recognition. IEEE Computer Society Press*, Silver

Spring, MD.

KOHONEN, T. (1984), *Self-Organization and Associative Memories*, Springer-Verlag, Berlin.

KOSKO, B. (1988), Bidirectional Associative Memories, *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-18, Jan./Feb., pag. 49-60.

KUPERSTEIN, M. & J. RUBINSTEIN (1988), Implementation of an Adaptative Neural Controller for Sensory-Motor Coordination, *IEEE Control Systems Magazine*, April, pag. 25-30.

LAIRD, J.E.; A. NEWELL & P.S. ROSENBLOOM (1987), SOAR: An Architecture for General Intelligence, *Artificial Intelligence*, Vol. 33, September, pag. 1-64.

LAIRD, J.E. (1988), *Proc. of the Fifth Int. Conf. on Machine Learning*, Univ. of Michigan, Ann Arbor.

LEE, D.T.; D.J. SOBAJIC; Y.H. PAO & J.L. DOLCE (1992), Adaptive Inverse Control of a Synchronous Machine using Neural Networks, *submitted to IEEE Power Engineering Society*, Winter General Meeting.

LEE, D.T.; D.J. SOBAJIC; Y.H. PAO & J.L. DOLCE (1990), Neural Net based Megawatt-Frequency Control, *Proc. 5th IEEE Int. Symposium on Intelligent Control*, Sep., Philadelphia, Pennsylvania.

LINSKER, R. (1988), Self-Organization in a Perceptual Network, *Computer*, March.

MACHADO, R.J. & A.F. ROCHA (1989), Redes Neurais Combinatórias - Um Modelo Conexionista para Sistemas Baseados em Conhecimento, *6º SBIA*, Rio, Brasil.

MACHADO, R.J.; V.H.A. DUARTE; F.A.R.M. DENIS & A.F. ROCHA (1991), NEXT - The Neuro EXpert Tool - Program Description and Operation Manual, *Relatório Técnico CCR-120 IBM Centro Científico Rio*, Rio, Brasil.

MACHADO, R.J.; C. FERLIN & A.F. ROCHA (1992), Combining Semantic and Neural Networks in Expert Systems, *Relatório Técnico CCR-140 IBM Centro Científico Rio*, Rio de Janeiro, Brasil.

MARGULIS, N. (1989), The Intel 80860, *BYTE*, December.

MARSHALL, T. (1990), A Calculating RISC, *BYTE*, May, pag. 251-257.

MATHEUS, C.J. & W.E. HOHENSEE (1987), Learning in Artificial Neural Systems, *Comput. Intell.*, Vol. 3, pag. 283-294.

MAY, D. & R. SHEPHERD (1984), The Transputer Implementation of OCCAM, *Proc. of the Int. Conf. on Fifth Generation Computer Systems*.

McCULLOCH, W.S. & W.H. PITTS (1943), A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bull. Math. Biophys.*, 5, pag. 115-133.

McGREGOR, J.D. & A.M. RIEHL (1989), Using an Architectural Knowledge Base to Generate Code for Parallel Computers, *Com. of the ACM*, 32, num 9, 1065-1072, Sept.

McGREGOR, J.D. & A.M. RIEHL (1989), The Future of High Performance Computers in Science and Engineering, *C. ACM*, 32, num 9, 1091-1101, September.

MICHALSKI, R.S.; J.G. CARBONELL & T.M. MITCHELL (1986), *Machine Learning: An Artificial Intelligence Approach. Volume II*, Morgan Kaufmann, Los Altos, CA.

MILLER III, W.T. (1989), Real-Time Application of Neural Network for Sensor-Based Control of Robots with Vision, *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, num. 4, July/August.

MINSKY, M. (1975), *A framework for representing knowledge*. In The Psychology of Computer Vision. P.Winston, Mc Graw-Hill.

MINSKY, M. & S. PAPERT (1969), *Perceptrons: an Introduction to Computational Geometry*, MIT Press, Cambridge, MA.

MIYAMOTO, H.; M. KAWATO; T. SETOYAMA & R. SUZUKI (1988), Feedback-error-learning Neural Network for Trayectory Control of a Robotic Manipulator, *Neural Networks*, Vol. 1, pag. 251-265.

MONTANA, D.J. & L. DAVIS (1989), Training Feedforward Neural Networks using Genetic Algorithms, *Proc. of IJCAI-89*, Detroit, pag. 762-767.

MÜHLENBEIN, H. (1990), Limitation of Multi-Layer Perceptrons Networks - Step towards Genetic Neural Networks, *Parallel Computing* 14, pag. 249-260.

MÜHLENBEIN, H. (1992), Parallel Genetic Algorithms and Neural Networks as Learning Machines, in D. EVANS, G. JOUBERT & H. LIDDELL: *Parallel Computing '91: Advances in Parallel Computing*, North-Holland, Amsterdam, pag. 91-103.

NAVAUX, P. (1990), *Processadores Pipeline e Processamento Vetorial*, VII Escola de Computação Brasileira, São Paulo, Brasil.

NORRIE, C. (1984), Supercomputers for Superproblems: An Architectural Introduction, *IEEE Computer*, March, 62-74.

OOSTHUIZEN, G.D. (1987), SUPERGRAN: A Connectionist Approach to Learning, Integrating Genetic Algorithms and Graph Induction, Dept. of Computer Science, University of Strathclyde, Scotland, pag. 132-139.

PALAGI, P.M. (1991), Algoritmos Genéticos no Aprendizado de Redes Neuronais, *Tese M.Sc. do Instituto de Ciências Exatas, Universidade de Brasilia*, Julho, Brasilia-DF, Brasil.

PASSINO, K.M.; M.A. SARTORI & P.J. ANTSAKLIS (1988), Neural Computing for

for Numeric-to-Symbolic Conversion in Control Systems, *IEEE Control Systems Magazine*, April, pag. 44-51.

PANCAKE, C.M. & D. BERGMARK (1990), Do Parallel Languages Respond to the Needs of Scientific Programmers?, *IEEE Computer*, vol. 23, num. 12, December.

Parallel C User Guide (1988), 3L Ltd.

3L Pascal Delivery Manual (1988), 3L Ltd.

PEASE, M.C. III (1977), The Indirect Binary n-Cube Microprocessor Array, *IEEE Transactions on Computers*, C-36, num 5, May, 458-473.

PENG, Y. & J.A. REGGIA (1989), A Connectionist Model For Diagnostic Problem Solving, *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, num. 2, March/April.

PERETTO, P. & J.J. NIEZ (1986), Stochastic Dynamics of Neural Networks, *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-16, num. 1, January/February, pag. 73-83.

PERRY, T.S. (1989), Intel's Secrets is Out, *IEEE Spectrum*, vol. 26, num. 4, April.

PETTEY, C.B.; M.R. LEUZE & J.J. GREFENSTETTE (1987), A Parallel Genetic Algorithm, *Vanderblit University*, Nashville, TN 37235, pag. 155-161.

PAO, Y.H. & D.J. SOBAJIC (1987), A Perspective on the Role of Parallel Distributed Processing of the Neural-Net Type in Implementing Automation, *Proc. of the IEEE Workshop on Languages for Automation*, Vienna, Austria.

PAO, Y.H. (1989), *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA.

PAO, Y.H.; V.C. CHEN & D.J. SOBAJIC (1989), A Perspective on Research Aimed at Understanding the Systems Nature of Neural Controllers, *Proc. of the Inte. Joint Conf. on Neural Networks*, Washington, D.C.

POUNTAIN, D. (1985), *A Tutorial Introduction to OCCAM Programming*, INMOS Limited.

PREPARATA, F.P. & J. VUILLEMIN (1981), The Cube-Connected Cycles: A Versatile Network for Parallel Computation, *Com. of the ACM*, 24, num. 5, May, 300-309.

PSALTIS, D.; A. SIDERIS & A.A. YAMAMURA (1988), A Multilayered Neural Networks Controller, *IEEE Control Systems Magazine*, April, Vol. 8, num. 2, pag. 17-21.

QUILAN, J.R. (1988), An Empirical Comparison of Genetic and Decision-Tree Classifiers, in J.E. LAIRD (Ed.), *Proc. of the Fifth Int. Conf. on Machine Learning*, Univ. of Michigan, Ann Arbor, 135-141.

QUINN, M.J. (1987), *Designing Efficient Algorithms for Parallel Computers*, McGraw-

Hill.

RANGWALA, S.S. & D.A. DORNFELD (1989), Learning and Optimization of Machining Operations Using Computing Abilities of Neural Networks, *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, num. 2, March/April.

RAUCH, H.E. & T. WINARSKE (1988), Neural Networks for Routing Communication Traffic, *IEEE Control Systems Magazine*, April, pag. 26-30.

REGGIA, J.A. & G.G. SUTTON III (1988), Self-Processing Networks and Their Biomedical Implications, *Proc. of the IEEE*, Vol. 76, Num. 6, June, pag. 680-692.

ROSENBLATT, F. (1962), *Principles of Neurodynamics*, Spartan, Washington, DC.

ROBERTSON, G. (1987), Parallel Implementation of Genetic Algorithms in a Classifier System, in: L. DAVIS (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, pag. 129-140.

ROBERTSON, G. (1988), Population Size in Classifier Systems, in J. LAIRD (Ed.), *Proc. of the Fifth Inte. Conf. on Machine Learning*, Univ. of Michigan, pag. 142-152.

ROBINSON, P. (1985), The SUM: An AI coprocessor, *BYTE*, Vol. 10, June, 169-180.

RUMELHART, D.E.; J.L. McCLELLAND & the PDP Group Research (1986), *Parallel Distributed Processing, I: Foundations*, MIT Press, Cambridge, MA.

RUMELHART, D.E. & D. ZIPSER (1985), Feature Discovery by Competitive Learning, *Cognitive Science*, 9, pag. 75-112.

RYAN, B. (1990), Separated at Birth, *BYTE*, May, pag. 207-212.

SANNIER II, A.V. & E.D. GOODMAN (1988), Midgard: A Genetic Approach to Adaptative Load Balancing for Distributed Systems, in J.E. LAIRD (Ed.), *Proc. of the Fifth Int. Conf. on Machine Learning*, Univ. of Michigan, pag. 174-180.

SAMAD, T. (1988), Towards Connectionist Rule-Based Systems, *Proc. of the IEEE Second Int. Confe. on Neural Networks*, Vol. 2, San Diego, California, pag. 525-532.

SAMUEL, A.L. (1959), Some Studies in Machine Learning Using the Game of Checkers, *IBM Journal of Research and Development*, 3, pag. 211-232.

SHIFFMANN, W. e K. MECKLENBURG (1991), Genetic Generation of Backpropagation Trained Neural Networks, in: R. ECKMILLER, G. HARTMANN & G. HAUSKE (Ed.), *Parallel Processing in Neural Systems and Computers*, North-Holland, Amsterdam, pag. 205-208.

SCHRODT, P.A. (1986), Predicting International Events, *BYTE*, November, 177-192.

SCIENTIFIC SUPERCOMPUTER SUBCOMMITTEE OF THE IEEE COMMITTEE ON COMMUNICATIONS AND INFORMATION POLICY (1988), Software for Supercomputers, *IEEE Computer*, December.

SEJNOWSKI, T.J. & C.R. ROSENBERG (1987), Parallel Networks that Learn to Pronounce English Text, *Complex Systems*, 1, pag. 145-168.

SKILLICORN, D.B. (1990), Architecture-Independent Parallel Computation, *IEEE Computer*, vol. 23, num. 12, December.

SOBAJIC, D.J.; D.T. LEE & Y.H. PAO (1988), Increased Effectiveness of Learning by Local Neural Feedback, *Int. Neural Networks Society First Annual Meeting*, Sep., Boston, MA.

SOBAJIC, D.J. & Y.H. PAO (1990), Neural-Net Control in Power System, *NSF Workshop on Artificial Neural-Net Methodology in Power Systems Engineering*, April, Clemson University, Clemson, SC.

SOLAR, M. (1991), Algoritmos Genéticos Paralelos e Conexionistas, *Exame de Qualificação para Candidato a D.Sc. da COPPE/UFRJ*, Rio, Brasil.

SOLAR, M. & S.B.T. MENDES (1991), Algoritmos Genéticos no Aprendizado das Redes Neurais, *Simposium Nacional de Computación*, Ciudad de México-DF, México, Novembro.

SOLAR. M. & S.B.T. MENDES (1992a), "Um Modelo Paralelo para o Aprendizado em Redes Neuronais baseado em Algoritmos Genéticos", XVIII Conferencia Latinoamericana en Informática - CLEI, PANEL'92, Setembro, Las Palmas de Gran Canaria, Espanha.

SOLAR, M. & S.B.T. MENDES (1992b), "Parallel Model for a Conexionist Genetic Algorithm", 12th World Computer Congress - IFIP Congress '92, Setembro, Madrid, Espanha.

SOLAR, M. & S.B.T. MENDES (1992c), "Un Modelo Conexionista para Control de Procesos: Beneficios y Problemas", I Congreso Interamericano de Computación Aplicada a la Industria de Procesos, CAIP 92, Centro de Información Tecnológica, La Serena, Chile.

SOUCEK, B. & M. SOUCEK (1988), *Neural and Massively Parallel Computers. The Sixth Generation*, Fleet Publications.

SPIESSENS, P. & B. MANDERICK (1991), A Genetic Algorithm for Massively Parallel Computers, in: R. ECKMILLER, G. HARTMANN & G. HAUSKE (Ed.), *Parallel Processing in Neural Systems and Computers*, North-Holland, Amsterdam, pag. 31-36.

STAUDHAMMER, J. (1987), Supercomputers and Graphics, *IEEE Computer Graphics and Applications*, July, 24-25.

STEIN, R.M. (1988), T800 and Counting, *BYTE*, November.

STONE, H. (1987), *High-Performance Computer Architecture*, Addison-Wesley Publishing Company.

SUDDARTH, S.C.; S.A. SUTTON & A.D.C. HOLDEN (1988), A Symbolic-Neural Method for Solving Control Problems, *Proc. of the IEEE Second Int. Conf. on Neural Networks*, Vol. 1, San Diego, California, pag. 515-523.

TALIA, D. (1990), Notes on Termination of OCCAM Processes, *SIGPLAN NOTICES*, vol. 25, num. 9, September.

TAZELAAR, J.M. (1990), Desktop Supercomputing, *BYTE*, May.

TERADA, R. (1990), *Introdução à Complexidade de Algoritmos Paralelos*, VII Escola de Computação, São Paulo, Brasil.

TOURETZKY, D.S. & D.A. POMERLEAU (1989), What's Hidden in the Hidden Layers?, *BYTE*, August, pag. 227-233.

UHR, L. (1987), *Multi-Computer Architectures for Artificial Intelligence*, Fleet Publications.

VON DER MARLSBURG, C. (1973), Self-Organizing of Orientation Sensitive Cells in the Striated Cortex, *Kybernetic*, 14, pag. 85-100.

WERBOS, P.J. (1989a), Maximizing Long-Term Gas Industry Profits in Two Minutes in Lotus Using Neural Network Methods, *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, num. 2, March/April.

WERBOS, P.J. (1989b), Backpropagation and Neurocontrol: A Review and Prospectus, *Proc. of the IEEE Third Int. Conf. on Neural Networks*, Vol. 1, San Diego, California, pag. 209-216.

WHITLEY, D.; T. STARKWEATHER & C. BOGART (1990), Genetic Algorithms and Neural Networks, *Parallel Computing 14*, pag. 347-361.

WIDROW, B. (1987), The Original Adaptive Neural-Nets Broom-Balancer, *IEEE Int. Symposium on Circuits and Systems*, pag. 351-357.

WIDROW, B. & R. WINTER (1988), Neural Nets for Adaptative Filtering and Adaptative Pattern Recognition, *Computer*, March.

WILLIAMS, R.J. (1988), On the Use of Backpropagation in Associative Reinforcement Learning, *Proc. of the IEEE Second Int. Conf. on Neural Networks*, Vol. 1, San Diego, California, pag. 263-270.

WILLIAMSON, M. (1985), *Artificial Intelligence for Microcomputers*, Brady Communications, New York.

WILSON, P. (1988), Parallel Processing Comes to PC's, *BYTE*, November, pag. 213-218.

ANEXO A

IMPLEMENTAÇÃO PARALELA DO MODELO AGC

```
/*

Host.c é o programa que controla os servidores Node.c.

Inicializa os programas de cada Nó e coleta os resultados.

*/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#include <time.h>

/* CONST */
#define  MAXPOP      40
#define  MAXSTRING  100
#define  TAMSIN       8
#define  MAXCAM       6
#define  FALSE        0
#define  TRUE         1
#define  TETA      0.02
#define  GAMA       1.0
#define  BRANCO "                                    "
#define  SIZE         8

#define  HOST_PID   100   /* id. do processo Host */
#define  APPL_PID     0   /* id. para as aplicações AG */
#define  INIT0_TYPE   0   /* tipo da msg inicial no cubo */
#define  INIT1_TYPE   1   /* tipo da msg inicial no cubo */
#define  INIT2_TYPE   2   /* tipo da msg inicial no cubo */
#define  INIT3_TYPE   3   /* tipo da msg inicial no cubo */
#define  SIZE_TYPE    4   /* tipo da msg Size */
#define  AG_TYPE      5   /* tipo de msg do AG */
#define  ALL_NODES   -1
#define  ALL_PIDS    -1 /*símbolo para todos os processos*/
#define  TIME_FACTOR 50.0 /* fator de conversão pata obter
                          tempo */

/* TYPE */
struct individuo {
         char   chrom[MAXSTRING];
         float  adapt, fenot;
         int    pai1, pai2, xsite;
         int    fim;
       } novapop[MAXPOP], velhapop[MAXPOP], global;


struct input {
         float caracter[MAXCAM];
        } padrao[15], target[15], output[15];
```

```
/* VAR  */
FILE   *f, *cr;
int    poptam, lchrom, gen, maxgen, numpad, numsin;
float  pcross, pmutacao, somadapt, fittotal, erromed;
int    nmutacao, ncross, jcross;
float  med, max, min;
int    maximo;
float  erromin, erant;
int    genint, cont, cte;
int    cam1, cam2, cam3;
float  a2[MAXCAM];
float  u1[MAXCAM];
float  u2[MAXCAM];
float  W12[MAXCAM][MAXCAM];
float  W23[MAXCAM][MAXCAM];

int    size;  /* tamanho do cubo consultado */
int    fim_processo=FALSE;
char   aleat[3];

double result;


long   tms, ms, tsec, sec, min;  /* para obter tempo */



OrdenaPopV()
{
 struct individuo indiv;
 int    i, j;

 for (i=0;i<=(poptam-1);i++)
   for (j=i+1;j<=poptam;j++)
    if (velhapop[j].adapt>velhapop[i].adapt)  {
       indiv  = velhapop[i];
       velhapop[i] = velhapop[j];
       velhapop[j] = indiv;
     }
}



OrdenaPopN()
{
 struct individuo indiv;
 int    i, j;

 for (i=0;i<=(poptam-1);i++)
   for (j=i+1;j<=poptam;j++)
    if (novapop[j].adapt>novapop[i].adapt)  {
       indiv      = novapop[i];
       novapop[i] = novapop[j];
       novapop[j] = indiv;
     }
}
```

```
Envia_Dados()
{
  csend(INIT0_TYPE,&cam1,sizeof(cam1), ALL_NODES,APPL_PID);
  csend(INIT0_TYPE,&cam2,sizeof(cam2),ALL_NODES, APPL_PID);
  csend(INIT0_TYPE,&cam3,sizeof(cam3),ALL_NODES, APPL_PID);
  csend(INIT0_TYPE,&numpad,sizeof(numpad),
        ALL_NODES,APPL_PID);
  csend(INIT0_TYPE,&cte,sizeof(cte),ALL_NODES, APPL_PID);
  csend(INIT1_TYPE,&aleat[0],sizeof(aleat[0]),
        ALL_NODES,APPL_PID);
  if ((aleat[0]=='n')||(aleat[0]=='N')) {
     csend(INIT2_TYPE,&padrao,sizeof(padrao),
           ALL_NODES,APPL_PID);
     csend(INIT3_TYPE,&target,sizeof(target),
           ALL_NODES, APPL_PID);
  }
}


Recebe_Melhor()
{
 int i;

 for (i=0; i<size; i++) {
  crecv(AG_TYPE,&velhapop[i],sizeof(velhapop[i]));
  if (velhapop[i].fim == TRUE) {
     fim_processo=TRUE;
     global = velhapop[i]
  }
 }
}




Salg()
{
 int  i, j, wg;
 char conf[3];
 char arquiv1[20];
 char arq[20];

 fim_processo = FALSE;
 randomize();
 gotoxy(1,25);
 printf("Entre com nome do arquivo de dados de entrada!");
 gotoxy(35,5); scanf("%s", &arq);
 ClrMens();
 if ((arq[0]==' ')||(arq[0]==''))  return(0);
 strcpy(arquiv1, arq);
 strcat(arq,".mat");
 strcat(arquiv1,".txt");
 if (((cr=fopen(arq,"w"))==NULL)||
     ((f=fopen(arquiv1,"w"))==NULL)) {
   gotoxy(1,25);
   printf("Arquivo existente. Sobrepoe (S/N)? S/N");
   scanf("%s",conf);
```

```
  ClrMens();
  if ((conf[0]=='N')||(conf[0]=='n'))  return(0);
 }
 Le_Dados();
 Envia_Dados();
 Tela2();
 fprintf(f, "Geracao n. med erromin   genint  nmutacao\n");
 do {
    if (fmod(++gen, 5)==0) fprintf(f,"    %d  ",gen);
    gotoxy(66,10); printf("%d",gen);
    Recebe_Melhor();
    OrdenaPopV();
    csend(AG_TYPE,&velhapop[0],sizeof(velhapop[0]),
          ALL_NODES, APPL_PID);
    Estatisticas();
 } while (fim_processo != TRUE);
 fprintf(f, " Geracao n.%d    ",gen);
 gotoxy(66,10); printf("%d",gen);
 velhapop[0] = global;
 Escreve_2();
 Descricao();
 fclose(f);
 fclose(cr);
}


InitReport()
{
 fprintf(f,"\n");
 fprintf(f," Parametros do ALG.GEN.\n");
 fprintf(f," ----------------------\n\n");
 fprintf(f,"Tamanho da Pop. (poptam)      = %d\n", poptam);
 fprintf(f,"Tamanho do Cromossomo (lchrom)= %d\n", lchrom);
 fprintf(f,"Prob. de Crossover (pcross)= %f\n", pcross);
 fprintf(f,"Prob. de mutacao (pmutacao)= %f\n", pmutacao);
 fprintf(f,"Constante Funcao Objetivo (cte) = %d\n", cte);
 fprintf(f,"Pop. inicial adapt max.         = %f\n", max);
 fprintf(f,"Pop. inicial adapt medio        = %f\n", med);
 fprintf(f,"Pop. inicial adapt min.         = %f\n", min);
 fprintf(f,"Pop. inicial adapt soma     = %f\n", somadapt);
 fprintf(f,"\n");
}


Tela1()
{
 LimpaTela();
 gotoxy(20, 5); printf("Nome Arquivo : ");
 gotoxy(20, 7); printf("Numero de Neuronios - ");
 gotoxy(20, 9); printf("Camada 1 : ");
 gotoxy(20,10); printf("Camada 2 : ");
 gotoxy(20,11); printf("Camada 3 : ");
 gotoxy(20,13); printf("Padroes Aleatorios (S/N)? ");
 gotoxy(20,14); printf("Numero de Padroes : ");
}
```

```
Tela2()
{
 LimpaTela();
 gotoxy(20, 7); printf("Erro Inicial: ");
 gotoxy(20,10); printf("Erro Medio Atual: 0.0000  ");
 gotoxy(51,10); printf("Geracao Atual: ");
 gotoxy(20,13); printf("Erro Medio Minimo: ");
 gotoxy(51,13); printf("Geracao E.minimo: ");
 gotoxy(20,17); printf("Numero mutacoes: ");
}


Tela3()
{
 LimpaTela();
 gotoxy(20, 5); printf("Nome Arquivo : ");
 gotoxy(20, 9); printf("Entrada - ");
 gotoxy(20,14); printf("Saida   - ");
}


ClrMens()
{ gotoxy(1,25); printf(BRANCO); }


LimpaTela()
{
 int i;
 for (i=1;i<=25;i++) { gotoxy(1,i); printf(BRANCO); }
}


int Menu()
{
 char selec[10];

  LimpaTela();
  gotoxy(20,4);
  printf("\n** Selecionar: A(prende)\n     S(imula) \n");
  printf("      F(im)      ** \n");
  do {
     scanf("%s",&selec);
     switch(selec[0])  {
      case 'a':
      case 'A': return(1);
      case 's':
      case 'S': return(2);
      case 'f':
      case 'F': return(3);
      default : printf("\n Responder A,S ou F\n");
                break;
     }
  } while ((selec[0]!='a')&&(selec[0]!='A')&&
              (selec[0]!='s')&&(selec[0]!='S')&&
              (selec[0]!='f')&&(selec[0]!='F'));
}
```

```
Rede_Feedf(a,o)
int a,o;
{
 int   i,j;
 float x;

 for (j=0;j<cam2;j++)  {
   u1[j] = 0.0;
   for (i=0;i<cam1;i++)
      u1[j] += (padrao[a].caracter[i] * (W12[j][i]));
   x = (-1.0 * GAMA)*(u1[j]-TETA);
   if (x >  88.0) x =  88.0;
   if (x < -88.0) x = -88.0;
   a2[j] = 1.0/(1.0 + exp(x));
  }
 for (j=0;j<cam3;j++)  {
   u2[j] = 0.0;
   for (i=0;i<cam2;i++) u2[j] += a2[i] * W23[j][i];
   x = (-1.0 * GAMA)*(u2[j]-TETA);
   if (x >  88.0)  x =  88.0;
   if (x < -88.0)  x = -88.0;
   output[o].caracter[j] = 1.0/(1.0 + exp(x));
  }
}


Le_Dados()
{
 int i,j;
 char aleat[3];

 gotoxy(31, 9); scanf("%d",&cam1);
 gotoxy(31,10); scanf("%d",&cam2);
 gotoxy(31,11); scanf("%d",&cam3);
 aleat[0]= ' ';
 while ((aleat[0]!='S')&&(aleat[0]!='s')&&
        (aleat[0]!='N')&&(aleat[0]!='n')) {
  gotoxy(45,13); scanf("%s",&aleat);
 }
 gotoxy(45,14);  scanf("%d",&numpad);
 if ((aleat[0]=='S')||(aleat[0]=='s'))
   for (j=0;j<numpad;j++) {
     for (i=0;i<cam1;i++)
        padrao[j].caracter[i] = random(256);
     for (i=0;i<cam3;i++)
        target[j].caracter[i] = random(256);
    }
  else  {
    LimpaTela();
    for (i=0;i<numpad;i++)  {
      gotoxy(20,16); printf("Padrao %d :",i);
      gotoxy(20,17); printf("Entrada : ");
      for (j=0;j<cam1;j++)
         scanf("%f",&padrao[i].caracter[j]);
      gotoxy(20,19); printf("Saida   : ");
      for (j=0;j<cam3;j++)
          scanf("%f",&target[i].caracter[j]);
```

```
    }
  }
 gotoxy(20,21); printf("Constante da Funcao Objetivo : ");
 gotoxy(55,21); scanf("%d",&cte);
 fprintf(f,"\n");
 fprintf(f,"Numero de padroes: %d\n",numpad);
 fprintf(f,"\n");
 fprintf(f,"Neuronios por camada: %d %d  %d\n",
          cam1,cam2,cam3);
 fprintf(cr," %d %d %d ", cam1, cam2, cam3);
 for (j=0;j<numpad;j++)  {
    fprintf(f,"Padrao %d - input  :",j);
    for (i=0;i<cam1;i++)
       fprintf(f," %f ", padrao[j].caracter[i]);
    fprintf(f, "  -  target :");
    for (i=0;i<cam3;i++)
       fprintf(f," %f ", target[j].caracter[i]);
    fprintf(f,"\n");
  }
 fprintf(f,"\n\n");
 LimpaMatrizes();
}


float Errof(k,l)
int  k,l;
{
 int i;
 float  acum=0.0;

 for (i=0;i<cam3;i++)
   acum += (( (target[k].caracter[i]-output[l].caracter[i])
         *(target[k].caracter[i]-output[l].caracter[i]))/2.0);
 return(acum);
}




LimpaMatrizes()
{
 int i,j;

 for (i=0;i<cam2;i++) for (j=0;j<cam1;j++) W12[i][j]= 0.0;
 for (i=0;i<cam3;i++) for (j=0;j<cam2;j++) W23[i][j]= 0.0;
}


float ObjFunc(x)
float x;
{
 float k = 1.0;
 int   i;

 for (i=1;i<=cte;i++) k=k*x;
 return(1.0/k);
}
```

```
Simulacao()
{
 int i;
 float  fit=0.0;
 float normaliza;

 for (i=0;i<numpad;i++)  {
    Rede_Feedf(i,i);
    fit += Errof(i,i);
  }
 normaliza = numpad*cam3;
 fittotal = ObjFunc((fit*2.0)/normaliza);
}


Transf_em_Matriz(cromos)
char cromos[MAXSTRING];
{
 int i,j,k,allelo=0;

 LimpaMatrizes();
 for (k=0;k<cam3;k++)
  for (j=0;j<cam2;j++)  {
     W23[k][j] = (cromos[allelo++])/10.0;
     for (i=0;i<cam1;i++)  W12[j][i] += (cromos[allelo++]);
   }
 for (j=0;j<cam2;j++)
    for (i=0;i<cam1;i++) W12[j][i] = W12[j][i]/(cam3*10.0);
}


Simula_Rede()
{
 int  i, nums;
 char arquiv[20];
 gotoxy(35,5);
 printf("\nNome do arquivo de dados de entrada: ");
 scanf("%s", &arquiv);
 if ((arquiv[0]==' ')||(arquiv[0]==''))  return(0);
 strcat(arquiv, ".mat");
 if ((cr=fopen(arquiv,"r")) == NULL) {
  printf("\nArquivo inexistente... Tecle ENTER\n");
  scanf("%s");
  return(0);
 }
 fscanf(cr,"%d %d %d", &cam1, &cam2, &cam3);
 nums = cam3*cam2*(cam1+1);
 gotoxy(20,10);
 for (i=0;i<cam1;i++) scanf("%f",&padrao[0].caracter[i]);
 for (i=0;i<nums;i++)
    fscanf(cr,"%d", &velhapop[0].chrom[i]);
 Transf_em_Matriz(velhapop[0].chrom);
 Rede_Feedf(0,0);
 gotoxy(20,15);
 for (i=0;i<cam3;i++) printf(" %f ",output[0].caracter[i]);
 fclose(cr);
}
```

```
Descricao()
{int   j;
 fprintf(f,"\n");
 fprintf(f," Cromossoma n.%d\n",maximo);
 fprintf(f,"  adapt : %f\n", velhapop[maximo].adapt);
 fprintf(f,"  erro  = %f\n", velhapop[maximo].fenot);
 fprintf(f," DESCRICAO :\n ");
 for (j=0;j<numsin;j++)  {
   fprintf( f, " %d ", velhapop[maximo].chrom[j]);
   fprintf(cr, " %d ", velhapop[maximo].chrom[j]);
 }
}

Escreve_1()
{if (gen==0)  {
   gotoxy(34,7); printf(" %f ",erromed);
   fprintf(f," erro medio inicial = %f\n", erromed);
   fprintf(f,"\n");
 }
 else {
   if (fmod(gen, 5)==0)
    fprintf(f," %f %f %d %d\n",
              erromed,erromin,genint,nmutacao);
   gotoxy(38,10); printf("%f", erromed);
   gotoxy(39,13); printf("%f", erromin);
   gotoxy(69,13); printf("%d", genint);
   gotoxy(37,17); printf("%d", nmutacao);
 }
}

Escreve_2()
{ fprintf(f," med = %f ; ",erromed);
  fprintf(f," erromin = %f ; ",erromin);
  fprintf(f,"  genint = %d ; \n",genint);
}

Estatisticas()
{int j;
 float cte1;
 min = max = somadapt = velhapop[0].adapt;
 maximo = 0;
 for (j=1;j<poptam;j++) {
    somadapt += velhapop[j].adapt;
    if (velhapop[j].adapt > max)  {
      max = velhapop[j].adapt; maximo = j;
    }
    if (velhapop[j].adapt<min) min= velhapop[j].adapt;
 }
 med = somadapt/poptam;
 cte1 = cte;
 erromed = exp(log(1.0/med)*(1.0/cte1));
 if (erromed<erromin)  {
     erromin = erromed;
     genint = gen;
 }
 Escreve_1();
}
```

```
Estatistica()
{
 int j;
 float cte1;
 min = max = somadapt = novapop[0].adapt;
 maximo = 0;
 for (j=1;j<poptam;j++)  {
    somadapt += novapop[j].adapt;
    if (novapop[j].adapt > max) {
      max = novapop[j].adapt;
      maximo = j;
    }
    if (novapop[j].adapt<min) min= novapop[j].adapt;
   }
  med = somadapt/poptam;
  cte1 = cte;
  erromed = exp((log(1.0/med))*(1.0/cte1));
  if (erromed<erromin) {
     erromin = erromed;
     genint = gen;
  }
  Escreve_1();
}


Estavel()
{
 float x;
 x = erant-erromed;
 if ((-0.002<=x)&&(x<=0.002)) cont++; else cont=0;
}


main()
{
 setpid(HOST_PID);
 load("nodo",ALL_NODES,APPL_PID);
 csend(PART_TYPE, &SIZE,sizeof(SIZE),ALL_NODES, APPL_PID);
 while (TRUE==1) {
   switch(Menu())  {
   case 1:  Tela1();  Salg();   break;
   case 2:  Tela3();  Simula_Rede();  break;
   case 3:  exit(0);
   default: printf("\nError na Seleção\n"); break;
   }
 }
   tms = msg.geracao;
   ms  = tms%1000;
   tsec = (tms-ms)/100;
   sec = tsec%60;
   min = (tsec-sec)/60;
 killcube(ALL_NODES, ALL_PIDS);
}
```

```
/*

   Nodo.c é o programa de cada Nó.
   Procura o ótimo na população local .

*/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#include <time.h>

/* CONST */
#define  MAXPOP      40
#define  MAXSTRING  100
#define  TAMSIN       8
#define  MAXCAM       6
#define  FALSE        0
#define  TRUE         1
#define  TETA      0.02
#define  GAMA       1.0
#define  BRANCO     "                              "
#define  NOS          8

#define  HOST_PID   100   /* id. do processo Host */
#define  APPL_PID     0   /* id. para as aplicacoes AG */
#define  INIT0_TYPE   0   /* tipo da msg inicial no cubo */
#define  INIT1_TYPE   1   /* tipo da msg inicial no cubo */
#define  INIT2_TYPE   2   /* tipo da msg inicial no cubo */
#define  INIT3_TYPE   3   /* tipo da msg inicial no cubo */
#define  SIZE_TYPE    4   /* tipo da msg Size */
#define  AG_TYPE      5   /* tipo de msg do AG */
#define  ALL_NODES   -1
#define  ALL_PIDS    -1   /* simbolo para todos os
                          processos */
#define  TIME_FACTOR 50.0 /* fator de conversão para obter
                          tempo */

/* TYPE */
struct individuo {
         char   chrom[MAXSTRING];
         float  adapt, fenot;
         int    pai1, pai2, xsite;
         int    fim;
       } novapop[MAXPOP], velhapop[MAXPOP];

struct input {
         float caracter[MAXCAM];
        } padrao[15], target[15], output[15];

/* VAR  */
int    poptam, lchrom, gen, maxgen, numpad, numsin;
float  pcross, pmutacao, somadapt, fittotal, erromed;
int    nmutacao, ncross, jcross;
float  med, max, min;
int    maximo;
```

```
float  erromin, erant;
int    genint, cont, cte;
int    cam1, cam2, cam3;
float  a2[MAXCAM];
float  u1[MAXCAM];
float  u2[MAXCAM];
float  W12[MAXCAM][MAXCAM];
float  W23[MAXCAM][MAXCAM];

char   aleat[3];

int    size;  /* tamanho do cubo consultado */
int    work_nodes;

long   tms, ms, tsec, sec, min;  /* para obter tempo */



int Flip(prob)
float prob;
{
 if (prob >= 0.9999) return(TRUE);
 else return( (random(100)/98.9) < prob);
}



int  Rnd(low, high)
int low,high;
{
 int e;

 if (low>=high) return(low);
 else {
       e = floor((random(10)/9.0) * (high-low+1)+low);
       if (e>high) return(high); else return(e);
  }
}


OrdenaPopV()
{
 struct individuo indiv;
 int    i, j;

 for (i=0;i<=(poptam-1);i++)
   for (j=i+1;j<=poptam;j++)
    if (velhapop[j].adapt>velhapop[i].adapt)  {
       indiv  = velhapop[i];
       velhapop[i] = velhapop[j];
       velhapop[j] = indiv;
     }
}
```

```
OrdenaPopN()
{
 struct individuo indiv;
 int    i, j;

 for (i=0;i<=(poptam-1);i++)
   for (j=i+1;j<=poptam;j++)
    if (novapop[j].adapt>novapop[i].adapt)  {
       indiv      = novapop[i];
       novapop[i] = novapop[j];
       novapop[j] = indiv;
     }
}



int Selecao(tipo)
int tipo;
{
 if ((tipo==0)||(tipo==1)) tipo=2;
 if (tipo>div(poptam,2).quot) tipo=10;
 return(random(tipo));
}



SetBit(n, ind, j)
int n, ind, j;
{
 float t;
 char des;
 float n1;

 n1 = n;
 t = exp(n1 * log(2.0));
 des = velhapop[ind].chrom[j];
 if ((des >> n) & ~(~0<<1))  velhapop[ind].chrom[j] -= t;
 else velhapop[ind].chrom[j] += t;
}



char Mutacao(pai, j)
int pai,j;
{
 int x;

 if (Flip(pmutacao)==TRUE)  {
   x = 7-random(8);
   nmutacao++;
   SetBit(x,pai,j);
   return(velhapop[pai].chrom[j]);
  }
 else return(velhapop[pai].chrom[j]);
}
```

```
Cross(f1, p1, f2, p2, j)
int f1, p1, f2, p2, j;
{ novapop[f1].chrom[j] = Mutacao(p1,j);
  novapop[f2].chrom[j] = Mutacao(p2,j);
}


Crossover(p1, p2, f1, f2)
int p1, p2, f1, f2;
{int  divisao, resto, j;

 if (Flip(pcross)==TRUE)  {
   divisao = random(numsin);  ncross++;
   for (j=0;j<divisao;j++)  Cross(f1, p1, f2, p2, j);
   for (j=divisao;j<numsin; j++) Cross(f1, p2, f2, p1, j);
  }
 else {
       divisao = div(lchrom, 8).quot;
       for (j=0;j<divisao;j++)  Cross(f1, p1, f2, p2, j);
  }
}


Genesis()
{
 int j=0, conj1, conj2;
 float cte1;

 OrdenaPopV();
 do {
   conj1 = Selecao(j); conj2 = Selecao(j);
   Crossover(conj1, conj2, j, j+1);
   Transf_em_Matriz(novapop[j].chrom);
   Simulacao();
   cte1 = cte;
   novapop[j].adapt = fittotal;
   novapop[j].fenot =
             exp(log(1.0/novapop[j].adapt)*(1.0/cte1));
   novapop[j].pai1 = conj1;
   novapop[j].pai2 = conj2;
   novapop[j].xsite = jcross;
   Transf_em_Matriz(novapop[j+1].chrom);
   Simulacao();
   novapop[j+1].adapt = fittotal;
   novapop[j+1].fenot=
             exp(log(1.0/novapop[j+1].adapt)*(1.0/cte1));
   novapop[j+1].pai1 = conj1;
   novapop[j+1].pai2 = conj2;
   novapop[j+1].xsite = jcross;
   j++;
 } while (++j<poptam);
 OrdenaPopN();
}
```

```
Salg()
{int  i, j, wg;

 randomize(); Recebe_Dados(); Inicializa();
 do {
     Genesis();
     wg = div(poptam,2).quot;
     for (i=0;(i<div(poptam,2).quot);i++) {
      for (j=0;j<numsin;j++)
         velhapop[wg].chrom[j] = novapop[i].chrom[j];
      velhapop[wg].adapt = novapop[i].adapt;
      velhapop[wg].fenot = novapop[i].fenot;
      velhapop[wg].pai1  = novapop[i].pai1;
      velhapop[wg].pai2  = novapop[i].pai2;
      velhapop[wg].fim   = FALSE;
      velhapop[wg++].xsite = novapop[i].xsite;
     }
     Estatisticas();  Estavel();
     csend(AG_TYPE,&velhapop[0], sizeof(velhapop[0]),
           myhost(), HOST_PID);
     crecv(AG_TYPE,&velhapop[poptam-1],
           sizeof(velhapop[poptam-1]));
     erant= erromed;
 } while ((cont<6)&&(gen<3500));
 velhapop[0].fim = TRUE;
 csend(AG_TYPE,&velhapop[0], sizeof(velhapop[0]),
       myhost(), HOST_PID);
 Escreve_2(); Descricao();
}


InitData()
{poptam  = 20; lchrom  = (cam1+1)*cam2*cam3*TAMSIN;
 pcross  = 0.90; pmutacao= 0.02;
 numsin  = cam3*cam2*(cam1+1);
 gen=genint=erromin=cont=nmutacao=ncross=jcross=genint=0;
}


InitPop()
{int j, j1;
 float cte1;
 randomize();
 for (j =0;j<poptam;j++) {
    for (j1=0;j1<numsin;j1++)
       velhapop[j].chrom[j1] = random(256);
    Transf_em_Matriz(velhapop[j].chrom);
    Simulacao();
    cte1 = cte;
    velhapop[j].adapt = fittotal;
    velhapop[j].fenot =
                exp((log(1.0/fittotal))*(1.0/cte1));
    velhapop[j].pai1=velhapop[j].pai2=velhapop[j].xsite=0;
    velhapop[j].fim  = FALSE;
   }
}
```

```
Inicializa()
{
 InitData(); InitPop(); Estatisticas();
 erant = erromin = erromed;
}


Estatisticas()
{
 int j;
 float cte1;

 min = max = somadapt = velhapop[0].adapt;
 maximo = 0;
 for (j=1;j<poptam;j++) {
    somadapt += velhapop[j].adapt;
    if (velhapop[j].adapt > max)  {
      max = velhapop[j].adapt;
      maximo = j;
    }
    if (velhapop[j].adapt<min) min= velhapop[j].adapt;
 }
 med = somadapt/poptam;
 cte1 = cte;
 erromed = exp(log(1.0/med)*(1.0/cte1));
 if (erromed<erromin)  {
     erromin = erromed;
     genint = gen;
 }
 Escreve_1();
}


Estatistica()
{
 int j;
 float cte1;

 min = max = somadapt = novapop[0].adapt;
 maximo = 0;
 for (j=1;j<poptam;j++)  {
    somadapt += novapop[j].adapt;
    if (novapop[j].adapt > max) {
      max = novapop[j].adapt;
      maximo = j;
    }
    if (novapop[j].adapt<min) min= novapop[j].adapt;
  }
  med = somadapt/poptam;
  cte1 = cte;
  erromed = exp((log(1.0/med))*(1.0/cte1));
  if (erromed<erromin) {
     erromin = erromed;
     genint = gen;
  }
  Escreve_1();
}
```

```
Estavel()
{float x;

 x = erant-erromed;
 if ((-0.002<=x)&&(x<=0.002)) cont++; else cont=0;
}


Rede_Feedf(a,o)
int a,o;
{
 int   i,j;
 float x;

 for (j=0;j<cam2;j++)  {
   u1[j] = 0.0;
   for (i=0;i<cam1;i++)
      u1[j] += (padrao[a].caracter[i] * (W12[j][i]));
   x = (-1.0 * GAMA)*(u1[j]-TETA);
   if (x >  88.0) x =  88.0;
   if (x < -88.0) x = -88.0;
   a2[j] = 1.0/(1.0 + exp(x));
  }
 for (j=0;j<cam3;j++)  {
   u2[j] = 0.0;
   for (i=0;i<cam2;i++) u2[j] += a2[i] * W23[j][i];
   x = (-1.0 * GAMA)*(u2[j]-TETA);
   if (x >  88.0)  x =  88.0;
   if (x < -88.0)  x = -88.0;
   output[o].caracter[j] = 1.0/(1.0 + exp(x));
  }
}



Recebe_Dados()
{
  crecv(INIT0_TYPE,&cam1,    sizeof(cam1));
  crecv(INIT0_TYPE,&cam2,    sizeof(cam2));
  crecv(INIT0_TYPE,&cam3,    sizeof(cam3));
  crecv(INIT0_TYPE,&numpad,  sizeof(numpad));
  crecv(INIT0_TYPE,&cte,     sizeof(cte));
  crecv(INIT1_TYPE,&aleat[0],sizeof(aleat[0]));
  if ((aleat[0]=='S')||(aleat[0]=='s'))
    for (j=0;j<numpad;j++) {
      for (i=0;i<cam1;i++)
         padrao[j].caracter[i] = random(256);
      for (i=0;i<cam3;i++)
         target[j].caracter[i] = random(256);
    }
  else  {
     crecv(INIT2_TYPE,&padrao,sizeof(padrao));
     crecv(INIT3_TYPE,&target,sizeof(target));
   }
 LimpaMatrizes();
}
```

```
float Errof(k,l)
int  k,l;
{int i; float  acum=0.0;

 for (i=0;i<cam3;i++)
   acum += (( (target[k].caracter[i]-output[l].caracter[i])
      *(target[k].caracter[i]-output[l].caracter[i]))/2.0);
 return(acum);
}


LimpaMatrizes()
{int i,j;

 for (i=0;i<cam2;i++) for (j=0;j<cam1;j++) W12[i][j]=0.0;
 for (i=0;i<cam3;i++) for (j=0;j<cam2;j++) W23[i][j]=0.0;
}


float ObjFunc(x)
float x;
{float k = 1.0; int   i;

 for (i=1;i<=cte;i++) k=k*x;
 return(1.0/k);
}


Simulacao()
{int i;
 float  fit=0.0;
 float normaliza;

 for (i=0;i<numpad;i++)  {
    Rede_Feedf(i,i); flt += Errof(i,i);
  }
 normaliza = numpad*cam3;
 fittotal = ObjFunc((fit*2.0)/normaliza);
}


Transf_em_Matriz(cromos)
char cromos[MAXSTRING];
{int i,j,k,allelo=0;

 LimpaMatrizes();
 for (k=0;k<cam3;k++)
  for (j=0;j<cam2;j++)  {
     W23[k][j] = (cromos[allelo++])/10.0;
     for (i=0;i<cam1;i++)  W12[j][i] += (cromos[allelo++]);
   }
 for (j=0;j<cam2;j++)
    for (i=0;i<cam1;i++) W12[j][i] = W12[j][i]/(cam3*10.0);
}
```

```
main()
{
 my_pid = mypid();
 my_node = mynode();
 for (;;) {
   crecv(SIZE_TYPE,&work_nodes,sizeof(work_nodes));
   if (my_node < work_nodes) {
        starttime = mclock();
        Salg();
   }
 }
}
```

ANEXO B: DADOS OBTIDOS

Os dados das Tabelas B.1, B.2, B.3 foram obtidos de uma RNA com l neurônios de entrada (l=2 na Tabela B.1; l=3 na Tabela B.2; l=4 na Tabela B.3.). No caso do OU-Exclusivo o n=1 (camada de saída). O L calculou-se com m = [l..2·l+2]. Portanto L=m·n·(l+1)·8 é da forma: $L = 8 \cdot m \cdot (l+1)$

TABELA B.1: L ótimo para OU-Exclusivo. (l=2)

$L = 24 \cdot m$ (m= 2,3,4,5,6,7)

L	1	2	3	4	5	6	7	8	9	10	x_m
48	23	22	24	20	26	21	24	22	25	23	**23**
72	16	18	14	15	17	13	19	14	18	16	**16**
96	16	18	15	19	16	18	19	15	17	16	**17**
120	**11**	**9**	**12**	**8**	**9**	**12**	**11**	**10**	**9**	**10**	**10**
144	32	34	31	35	30	36	32	35	31	34	**33**
168	35	34	32	35	30	36	33	35	32	36	**34**

TABELA B.2: L ótimo para OU-Exclusivo. (l=3)

$L = 32 \cdot m$ (m= 3,4,5,6,7,8)

L	1	2	3	4	5	6	7	8	9	10	x_m
96	36	38	35	34	39	33	37	36	34	38	**36**
128	22	25	20	26	23	25	22	26	25	24	**24**
160	15	18	16	23	20	21	19	22	15	18	**19**
192	19	20	18	21	14	17	14	17	15	22	**18**
224	**15**	**12**	**10**	**13**	**16**	**14**	**15**	**16**	**15**	**16**	**14**
256	28	24	26	27	23	29	24	25	28	26	**26**

TABELA B.3: L ótimo para OU-Exclusivo. (l=4)

$L = 40 \cdot m$ (m= 4,5,6,7,8,9)

L	1	2	3	4	5	6	7	8	9	10	x_m
160	65	68	68	66	70	69	65	67	66	66	**67**
200	47	48	45	49	46	48	49	45	47	46	**48**
240	41	39	42	38	39	42	41	40	39	40	**40**
280	46	45	48	48	46	50	49	45	47	46	**47**
320	49	50	44	47	45	52	48	51	44	47	**48**
360	**36**	**38**	**34**	**35**	**37**	**33**	**39**	**34**	**38**	**36**	**35**

Os dados das Tabelas B.4, B.5, B.6 foram obtidos de uma RNA com l neurônios de entrada (l=2 na Tabela B.4; l=3 na Tabela B.5; l=4 na Tabela B.6.)

No caso da Inversão o n=l (camada de saída). O L calculou-se com m = [l..2·l+2]. Portanto L=m·l·(l+1)·8 é da forma:

$$L = 8 \cdot m \cdot l \cdot (l+1)$$

TABELA B.4: L ótimo para Inversão. (l=2)

L = 48·m (m= 2,3,4,5,6)

L	1	2	3	4	5	6	7	8	9	10	x_m
96	22	18	21	14	19	20	17	15	17	14	**18**
144	15	14	15	16	16	16	15	12	10	13	**14**
192	11	10	9	12	8	9	12	10	11	9	**10**
240	**12**	**8**	**11**	**5**	**9**	**10**	**7**	**5**	**7**	**6**	**8**
288	13	11	14	12	15	12	14	10	16	13	**13**

TABELA B.5: L ótimo para Inversão. (l=3)

L = 96·m (m= 3,4,5,6,7)

L	1	2	3	4	5	6	7	8	9	10	x_m
288	26	23	25	22	25	20	22	26	25	24	**24**
384	16	18	19	15	18	15	19	16	17	16	**17**
480	15	18	14	17	17	14	16	18	17	15	**16**
576	15	13	11	14	12	10	12	14	16	13	**13**
672	**12**	**10**	**13**	**10**	**9**	**13**	**12**	**10**	**9**	**11**	**11**

TABELA B.6: L ótimo para Inversão. (l=4)

L = 160·m (m= 4,5)

L	1	2	3	4	5	6	7	8	9	10	x_m
640	22	20	22	20	23	19	21	20	19	23	**21**
800	**16**	**18**	**14**	**15**	**17**	**13**	**19**	**14**	**18**	**16**	**16**

Os dados das Tabelas B.7, B.8, B.9 foram obtidos de uma RNA com l neurônios de entrada (l=2 na Tabela B.7; l=3 na Tabela B.8; l=4 na Tabela B.9.)

No caso da Simetria o n=1 (camada de saída). O L calculou-se com m = [l..2·l+2]. Portanto L=m·n·(l+1)·8 é da forma:

$$L = 8 \cdot m \cdot (l+1)$$

TABELA B.7: L ótimo para Simetria. (l=2)

L = 24·m (m= 2,3,4,5,6,7)

L	1	2	3	4	5	6	7	8	9	10	x_m
48	22	22	20	20	23	19	21	23	20	19	**21**
72	14	19	20	17	15	22	18	21	17	14	**18**
96	16	15	14	15	10	16	16	15	12	13	**14**
120	**5**	**9**	**5**	**7**	**10**	**7**	**12**	**8**	**11**	**6**	**8**
144	11	10	9	12	8	9	12	10	11	9	**10**
168	13	13	11	11	14	12	11	14	10	10	**12**

TABELA B.8: L ótimo para Simetria. (l=3)

L = 32·m (m= 4,5,6,7,8)

L	1	2	3	4	5	6	7	8	9	10	x_m
128	46	48	45	44	49	43	47	46	44	48	**46**
160	30	36	32	32	34	31	35	34	35	31	**33**
192	25	25	23	23	26	26	23	24	23	22	**24**
224	**10**	**12**	**12**	**11**	**13**	**10**	**10**	**13**	**9**	**9**	**11**
256	19	15	16	18	15	19	16	18	17	16	**17**

TABELA B.9: L ótimo para Simetria. (l=4)

L = 40·m (m= 6,7,8,9)

L	1	2	3	4	5	6	7	8	9	10	x_m
240	36	38	34	35	37	33	39	34	38	36	**35**
280	30	32	32	31	33	30	30	33	29	29	**31**
320	21	23	23	22	24	21	21	24	20	20	**22**
360	**21**	**23**	**20**	**22**	**22**	**20**	**23**	**19**	**20**	**19**	**21**

Os dados das Tabelas B.10, B.11, B.12 foram obtidos de uma RNA com l neurônios de entrada (l=2 na Tabela B.10; l=3 na Tabela B.11; l=4 na Tabela B.12.)

No caso da Associatividade o n=l (camada de saída). O L calculou-se com m = [l..2·l+2]. Portanto L=m·l·(l+1)·8 é da forma:

$$L = 8 \cdot m \cdot l \cdot (l+1)$$

TABELA B.10: L ótimo para Associatividade. (l=2)

L = 48·m (m= 2,3,4,5,6)

L	1	2	3	4	5	6	7	8	9	10	x_m
96	13	13	11	11	14	12	11	14	10	10	**12**
144	6	10	6	8	11	8	13	9	12	7	**9**
192	6	5	5	5	6	7	7	7	6	7	**6**
240	**5**	**5**	**6**	**5**	**6**	**7**	**7**	**7**	**7**	**7**	**6**
288	12	8	5	9	10	7	5	11	7	6	**8**

TABELA B.11: L ótimo para Associatividade. (l=3)

L = 96·m (m= 3,4,5,6,7,8)

L	1	2	3	4	5	6	7	8	9	10	x_m
288	18	15	19	16	18	17	16	19	15	16	**17**
384	14	12	10	15	13	11	12	14	16	13	**13**
480	13	11	14	12	11	14	10	11	10	13	**12**
576	8	11	8	13	9	12	6	10	6	7	**9**
672	**5**	**12**	**8**	**9**	**7**	**6**	**10**	**7**	**5**	**11**	**8**
768	12	10	11	11	13	11	14	14	10	13	**12**

TABELA B.12: L ótimo para Associatividade. (l=4)

L = 160·m (m= 4,5,6)

L	1	2	3	4	5	6	7	8	9	10	x_m
640	23	20	22	20	22	23	19	20	19	21	**21**
800	20	22	19	21	21	18	22	19	18	19	**20**
960	**15**	**17**	**13**	**19**	**16**	**18**	**14**	**14**	**18**	**16**	**16**

Para a obtenção da relação entre o TP com o número de Gerações fixou-se o problema com l=3 para todos os casos, e o L usado foi o obtido como ótimo segundo os dados acima.

TABELA B.13: Relação TP com Geração para OU-Exclusivo.

TP	1	2	3	4	5	6	7	8	9	10	x_m
10	22	24	24	23	25	21	21	25	22	22	**23**
20	15	12	10	13	16	14	15	16	15	16	**14**
30	13	10	10	12	12	12	13	14	13	15	**12**
40	11	8	13	9	6	10	6	8	12	7	**9**
50	12	7	6	10	6	8	11	8	13	9	**9**
60	6	9	6	7	10	7	11	8	10	7	**8**

TABELA B.14: Relação TP com Geração para Inversão.

TP	1	2	3	4	5	6	7	8	9	10	x_m
10	37	39	36	35	38	34	40	35	39	37	**36**
20	12	10	13	10	9	13	12	10	9	11	**11**
30	15	12	10	13	21	14	20	16	15	16	**15**
40	16	13	15	21	20	16	13	14	18	16	**16**
50	18	15	19	16	18	17	16	19	15	16	**17**
60	11	10	9	12	8	9	12	10	11	9	**10**

TABELA B.15: Relação TP com Geração para Simetria.

TP	1	2	3	4	5	6	7	8	9	10	x_m
10	25	22	20	23	26	24	25	26	25	26	**24**
20	10	12	12	11	13	10	10	13	9	9	**11**
30	10	12	10	13	9	10	9	13	12	11	**11**
40	13	9	6	11	8	10	6	7	8	12	**9**
50	10	7	11	8	6	9	6	7	10	7	**8**
60	6	7	10	6	9	7	10	11	8	7	**8**

TABELA B.16: Relação TP com Geração para Associatividade.

TP	1	2	3	4	5	6	7	8	9	10	x_m
10	16	13	15	21	20	16	13	14	18	16	**16**
20	5	12	8	9	7	6	10	7	5	11	**8**
30	8	6	9	6	7	10	7	11	10	7	**8**
40	6	5	5	5	6	7	7	7	6	7	**6**
50	7	9	5	5	6	11	7	8	6	7	**7**
60	6	7	6	6	7	5	5	5	7	7	**6**

TABELA B.17: Relação TP com Tempo de Processamento Ou-Exclusivo.

TP	tp
20	7,22
30	7,07
40	6,78
50	6,81
60	6,86

TABELA B.18: Relação TP com Tempo de Processamento Simetria.

TP	tp
20	3,22
30	4,01
40	3,48
50	3,31
60	3,28

TABELA B.19: Relação TP com Tempo de Processamento Inversão.

TP	tp
20	6,52
30	6,42
40	6,48
50	6,21
60	6,39

TABELA B.20: Relação TP com Tempo de Processamento Autoassociatividade.

TP	tp
20	3,07
30	2,89
40	3,14
50	2,97
60	3,07

ANEXO C
METODOLOGIA DE DESENVOLVIMENTO PARA IMPLEMENTAR UMA RNA

Como no projeto e análise de qualquer sistema complexo, a computação neuronal requer uma metodologia que permita uma tarefa ser descomposta por toda parte em subtarefas, com passos adicionais de refinamento (e vários processos de iteração para cima ou para baixo nos níveis de complexidade) que são necessários antes de derivar as subtarefas que é um processo muito dirigido na implementação.

Na sequencia são mostradas as fases mais importantes no desenvolvimento de uma RNA, tais como a seleção dos paradígmas e dos parâmetros do algorítmo de aprendizado.

C.1.- FASE CONCEITUAL

A fase conceitual planeja uma aproximação para construir a aplicação, onde valida a proposta e seleciona os paradígmas conexionistas que podem ser adequados para atingir os requerimentos necessários.

Uma forma de validar a aplicação é comparando as características com outras aplicações. As aplicações das RNA mais bem sucedidas têm várias características comúns:

- a aplicação depende intensivamente dos dados com interações múltiplas dos parâmetros;
- a área do problema é rica em dados históricos ou exemplos;
- o conjunto de dados é incompleto, contêm erros, e descreve exemplos específicos;
- são desconhecidas as funções para determinar uma solução ou é dificil de descobrir.

Muitas classes de aplicações que envolvem reconhecimento de padrões ou mapeamento estatístico são interessantes de serem desenvolvidas com RNA. As aplicações mais comúns são reconhecimento de caracteres, previsão, processamento de informação, monitoração de processos, processamento de sinais, controle de robós e controle adaptativo.

Os pesquisadores (HECHT-NIELSEN, 1987; OBERMEIER e BARRON, 1989) indicam que existem várias heurísticas para selecionar as aplicações das RNA,

entre elas:

- a tecnologia dos computadores convencionais não é adequada;
- o problema requer de raciocínio qualitativo ou raciocínio quantitativo complexo;
- a solução é derivada de parâmetros altamente inter-dependentes que não possuem uma quantificação precisa;
- os dados estão imediatamente disponíveis mas são multivariados e intrinsicamente ruidoso;
- o tempo de desenvolvimento do projeto é curto, mas dispõe-se de tempo suficiente para treinar uma RNA.

Algumas propriedades gerais das RNA influenciam bastante na decisão de usá-las, tais como paralelismo, adaptabilidade, generalização, velocidade e flexibilidade.

Também são importante as necessidades específicas da aplicação, tais como: recursos (tempo, equipamento, dinheiro); fonte dos dados (tipo, custo, precisão, tempo); requerimentos da solução (tipo, precisão, consistência); tempo (projeto, treinamento, operação); técnicas atuais e algorítmos de aprendizado.

O segundo passo da fase conceitual é selecionar o conjunto inicial dos paradígmas conexionistas, sendo os mais interessantes: o tamanho, o tipo da saída requerida, a classificação de memória associativa, o método de treinamento e as necessidades do tempo.

C.1.1.- Tamanho da RNA

Uma vez que tem sido selecionada a aplicação, é essencial limitar o problema para garantir que o problema está apropriadamente isolado de efeitos exógenos e escalado no tamanho adequado para as capacidades computacionais neuronais. Como regra empírica, os paradígmas conexionistas não-estatísticos, tais como "back propagation" e "counter propagation" trabalham melhor com RNA com menos de 1000 nós. Os paradígmas estatísticos, incluindo a máquina de BOLTZMANN, geralmente requerem uma RNA significativamente menor (na ordem de 200 nós).

C.1.2.- Tipo da Saída Requerida

O tipo da saída requerida pela aplicação limita os paradígmas conexionistas

que podem ser selecionados. Existem quatro interpretações comúns para a saída das RNA, sendo: classificação, padrão, números reais e otimização.

As classificações mapeiam estatisticamente as entradas para categorias discretas. Tipicamente, existem dois ou mais nós de saída, com só um deles ativo em qualquer tempo. Por exemplo, uma aplicação de controle de processos pode ter três classificações: normal, congestionado e alarme. Um só destes três modos é apropriado em qualquer tempo. Outras aplicações que usam classificação incluem processamento de informação, processamento de sinais, e análise de cenários.

Os padrões são possivelmente a saída mais comúm nas diversas RNA. Neste tipo de saída, os nós múltiplos (todos potencialmente ativos) formam um padrão em resposta à entrada. O nível de ativação dos nós de saída podem ser valores de verdade (booleano) ou escalas de tonalidades ("gray"), dependendo da escolha do paradígma conexionista. As áreas de aplicações que contam com saídas de padrões são processamento de imagens, fusão de dados e identificação de símbolos.

As saídas de números reais traduzem em valores, tais como degraus, distância ou tensão. Os nós de saída que produzem números reais podem ser auto-suficientes ou compor vários conjuntos. As áreas de aplicações que produzem números reais incluem previsão financeira, mapeamento funcional, controle de processos e robótica.

A otimização é um padrão especial interpretado como um conjunto de decisões. As otimizações são o resultado de problemas específicos da pesquisa operacional que podem ser formulados em matrizes. Exemplos de otimização que têm sido resolvidos por RNAs são o problema do caixeiro viajante e problemas de alocação de tarefas.

C.1.3.- Classificação de Memórias Associativas

A memória associativa obtém a informação completa a partir de informação parcial, e tem um papel fundamental em reconhecimento de padrões e aplicações de processamento da informação.

Duas variantes de memórias associativas são de especial interesse nas aplicações de RNAs: memórias auto-associativas e hetero-associativas. A memória auto-

associativa mapeia parcelas de dados em si mesmos, memorizando informação específica. Os paradígmas conexionistas auto-associativo mais comúns são a RNA de HOPFIELD (1982) e a teoria de resonancia adaptativa (ART) (CARPENTER e GROSSBERG, 1988), embora muitos pesquisadores têm explorado paradígmas similares. Este tipo de memória é usado quando se quer reconstruir um padrão parcial ou um padrão contendo erros na sua forma original, ou para otimizar certos problemas da pesquisa operacional. Exemplos de aplicações auto-associativas são reconhecimento de caracteres, completar imagens, reconhecimento retinal e reconstrução de sinais.

A memória hetero-associativa mapeia um conjunto de padrões para um outro padrão. Os padrões, as classificações e os números reais são geralmente armazenados como saídas de redes hetero-associativas. Os paradígmas conexionistas que pertencem às memórias hetero-associativas incluem Memória Associativa Bidireccional (BAM) (KOSKO, 1988), o esquema de mapeamento do KOHONEN (1984), e alguns paradígmas estatísticos. Exemplos de aplicações hetero-associativas são classificação de padrões e monitoração e controle de processos.

C.1.4.- Método de Treinamento

O método apropriado para treinar uma RNA depende principalmente da disponibilidade dos dados e do tempo. As três classificações comúns dos métodos de treinamento são: supervisionado, não-supervisionado e aprendizado reforçado.

O aprendizado supervisionado requer pares de dados consistindo de um padrão de entrada e o resultado correto. Os dados de treinamento devem possuir a solução da RNA, o qual as vezes é dificil de conseguir. O tempo requerido para treinar uma RNA usando este aprendizado depende do paradígma específico usado e da natureza do problema, mas é moderadamente alto (usualmente muitas horas por alguns dias).

O treinamento não-supervisionado classifica os padrões de entrada internamente e não precisa de um resultado esperado. Os dados requeridos para este tipo de treinamento são portanto mais simples de obter e com menor custo. O tempo de treinamento para o aprendizado não-supervisionado, normalmente requer muitas horas de treinamento.

O aprendizado reforçado é um compromiso entre o treinamento

supervisionado e o não-supervisionado, o qual requer uma entrada e só uma recompensa ou punição como saída. O aprendizado reforçado é menos comúm pela sua complexidade e longo tempo de treinamento, mas tem um maior potencial para treinar sistemas modulares grandes de RNA interconectadas. Os dados requeridos são menos exigentes que para o supervisionado e geralmente pode ser alimentado dentro do sistema em forma dinámica via realimentação em vez de conexões estáticas.

C.1.5.- Limitações de Tempo

O tempo é um fator importante sempre pouco considerado no desenvolvimento de uma RNA. Existem duas categorias básicas: o tempo de treinamento e o tempo de execução. O tempo de treinamento começa quando os dados são apresentados para a RNA e termina quando as conexões estão ajustadas apropriadamente. Dado que o tempo de treinamento depende do tamanho e da qualidade do conjunto de treinamento, e é consideravelmente maior que o tempo de execução, o treinamento é geralmente efetuado em forma "off-line". O tempo de execução começa quando uma RNA treinada recebe informação de entrada e acaba quando a RNA produz um resultado. O tempo de operação depende principalmente do computador e da dinámica da RNA, e deve cumprir limitações "on-line" ou em tempo real.

A tabela C.1 mostra uma base para selecionar um conjunto viável de paradígmas conexionistas. Selecionar um pequeno conjunto de paradígmas para a experimentação é melhor, porque os fatores específicos da aplicação ocasionalmente elimina alguns deles. Por exemplo, dado um problema de classificação em monitoração de processos, os paradígmas mais apropriados são "back propagation", "counter propagation" e elementos adaptativos lineares (MADALINE) (BARTO, SUTTON e ANDERSON, 1983). Dada a natureza do problema, eventualmente "counter propagation" e MADALINE são incapaz de atingir a exatidão requerida (BAILEY e THOMPSON, 1990).

Depois que a aplicação proposta é validada e os paradígmas conexionistas foram selecionados, deve ser feito um estudo de practicabilidade. Se o projeto é julgado técnica e economicamente possível, então a fase conceitual está completada e o projeto está pronto para a fase de projeto.

Design Factor / Paradigm	Transfer Function	Number of Layers	Type of Input Accepted	Size of Hidden Layers	Connectivity	Learning Algorithms	Learning Parameters
Back Propagation	Sigmoid Hyperbolic Tangent	2 or More	Continuous	Small to Medium	Fully Inter-Connected, Random, Slabs	Generalized Delta Rule	Learning Constant, Momentum
Counter Propagation	Kohonen & Sigmoid	2 or 4	Continuous	Same as Kohonen	Fully Inter-Connected	Kohonen & Grossberg	Kohonen & Outstar
Madaline	Signum	2	Bipolar	Small to Medium	Fully Inter-Connected	Delta Rule & Max, Min, Majority	Learning Constant
Outstar	Sigmoid	1	Binary Continuous	N/A	Varies by Application	Outstar	Decay Time, Attrack Func.
Art-2	Sigmoid	1	Grey Scale	Increases by Data Catagory	Fully Inter-Connected	Art 2	Vigilance Gain
Kohonen Network	Competitive Learning	1	Continuous	Equals # of Data Categories	Fully Inter-Connected	Kohonen	Neighborhood Size, Alpha, Beta
Boltzmann Machine	Varies	2 or More	Binary Continuous	Small to Medium	Fully Inter-Connected Random	Boltzmann	Temperature
Hopfield Network	Hard Limiting	1	Binary Bipolar	N/A	Fully Inter-Connected	Hopfield	None
Adaline	Signum	1	Bipolar	N/A	Fully Inter-Connected Random	Delta Rule	Learning Constant
BAM	Clamped Linear	1	Bipolar	N/A	Fully Inter-Connceted	BAM	Retention, Gain
Perceptron	Perceptron	1	Continuous	N/A	Random	Perceptron Convergence, Delta Rule	Learning Constant

TABELA C.1: Paradígmas em RNAs.

C.2.- FASE DE PROJETO

A fase de projeto especifica os valores iniciais e condições para selecionar os paradígmas conexionistas dos nós, da RNA e níveis de treinamento. A fase de projeto inclui vários passos para determinar os tipo dos nós ou elementos de processamento, o tamanho e conectividade das camadas da RNA, e o algorítmo de aprendizado e parâmetros.

A Tabela C.1 mostra as decisões do projeto para alguns dos paradígmas conexionistas mais comunmente usados. Muitos destes paradígmas oferecem flexibilidade em termos de tamanho, conectividade ou parâmetros de aprendizado, enquanto outros são levemente restringidos pela natureza da arquitetura ou algorítmo de aprendizado.

Como definição muitos autores definem camada ("layer"), como o conjunto de nós cujos pesos são manipulados ativamente. Os nós que servem de entrada ou saída não são considerado como camada. O termo grupo ("slab" ou "cluster") é usado para denotar conjuntos de nós que podem ser diferentes em termos de especificação interna ou conectividade, mas compartilham a mesma camada. Portanto, uma única camada pode conter múltiplos grupos.

C.2.1.- Nível do Nó

O primeiro passo no projeto de uma RNA é determinar quais nós ou elementos de processamento usar. As decisões neste nível incluem o tipo de entradas, o meio de combinação e a função de transferencia.

Os nós complexos nas RNA incluem três funções: o meio de combinação, uma função de ativação, e uma função de saída. O meio de combinação é usualmente uma soma ponderada das entradas, mas outras funções podem ser usadas. O nível de excitação ou ativação do nó é determinado pela função de ativação, o qual vai depender do nível de ativação anterior somado como entrada corrente. A função de saída usa o nível de ativação para produzir uma saída. Muitas RNAs comúns projetadas simplificam seus nós pela combinação das funções de ativação e saída numa função de transferencia.

Os pesquisadores tem experimentado muitas funções de transferencia, tais como linear, gancho linear ("clamped"), signum, sigmoide e tangente hiperbólica. A escolha da função de transferencia está baseada nos tipos de entrada e saídas e o algorítmo de aprendizado a ser usado. Para os pares binários de entrada-saída pode-se usar uma função degrau. Para um par de entrada-saída bipolar (-1 e +1) é usada a função signum. Os nós que manipulam valores contínuos usam a função de transferencia linear ou sigmoidal.

Alguns algorítmos de aprendizado impõem restrições para a função de transferencia que se deseja usar. Por exemplo, "backpropagation" requer que a função seja diferenciável em todo ponto, e portanto usa a função de transferencia sigmoidal.

Os requerimentos computacionais para a função de transferencia sigmoidal são maiores que aqueles de natureza linear ou degrau. Estes requerimentos tem motivado alguns pesquisadores para criar tabelas de melhora ou busca ("loockup") de

valores para substituir a computação. Similarmente, os cálculos de probabilidade para os paradígmas estatísticos, tais como máquina de BOLTZMANN sempre envolvem tabelas de busca.

C.2.1.- Nível da RNA

O segundo passo na fase de projeto é determinar o número, o tamanho e a conectividade das diferentes camadas da RNA. As decisões que devem ser tomadas neste ponto são o número de camadas, o tamanho de cada camada, o tipo de entrada e saída esperado, e como ligar cada camada.

Muitos dos paradígmas conexionistas comunmente usados tem um número pre-determinado de camadas. Por exemplo, ADALINE, MADALINE, a RNA de HOPFIELD, ART-1, ART-2, o esquema de mapeamento auto-organizado de KOHONEN, e as memórias associativas bidirecionais (BAM) requerem de uma ou duas camadas para funcionar. Outros paradígmas, tais como "backpropagation", paradígmas estatísticos, e Neocognitron (FUKUSHIMA et alii, 1983), permitem um número variável de camadas escondidas na RNA.

As camadas escondidas agem como camadas de abstração, puxando esquemas da entrada. Incrementando o número de camadas escondidas sequenciais aumenta o poder do processamento da RNA, mas complica significativamente o treinamento e intensifica o efeito de "caixa preta" (os erros são mais difícil de detetar). Acrescentar mais camadas aumenta o tempo e o número de exemplos de treinamento necessários para treinar a RNA apropriadamente. Como resultado empírico, o melhor é começar com uma camada e acrescentar as outras camadas na medida que seja necessário.

Outro método para acrescentar o poder de processamento da RNA é somar múltiplos grupos dentro de uma única camada escondida. Os grupos múltiplos paralelos podem usar diferentes tipos e números de nós. Esta arquitetura tenta forçar os grupos a extrair diferentes esquemas simultaneamente. Muitas aplicações não requerem grupos múltiplos dentro das camadas escondidas, mas esta arquitetura provê uma alternativa interessante para uma única camada escondida.

Depois de selecionar o número de camadas para a RNA, o passo seguinte na fase de projeto é determinar o tamanho (em número de nós) de cada camada. A camada

de entrada apresenta os dados para a RNA. O número de nós de entrada é calculado a partir do número de dados fontes e os nós necessários para representar cada fonte. Geralmente a decisão mais difícil de projetar é acertar a fonte correta dos dados: o volume de dados não processados ou dados falsos impedem o treinamento.

São requeridos quatro passos para determinar o número apropriado de dados fontes. Primeiro, identificar todos os dados que em alguma forma estão relacionados com a área da aplicação. Segundo, eliminar os dados fontes considerados como irrealizáveis. Terceiro, filtrar os dados fontes que são impraticáveis por razões técnicas ou económicas. No fim, explorar métodos de combinar ou pre-processar dados para fazé-los mais significativas. Por exemplo, as razões são muitas vezes mais significativas que os números individuais, porque provem explicitamente uma relação relevante, em vez de forçar a RNA a descobri-la.

Depois que a fonte dos dados foi identificada, ela deve ser representada na camada de entrada da RNA. Os dados de valores contínuos para o tempo T podem ser representados como um nó simples na RNA que aceita números reais. Se a RNA aceitar só entradas binárias ou bipolares, pode ser usada uma série de nós representando escopos de valores para a fonte de dados. As entradas binárias ou bipolares são muitas vezes usadas para indicar a presença ou ausencia de uma categoria ou atributo. Os padrões são representados por uma série de nós binários ou escala de tonalidades.

Tendo suficientes nós na camada escondida, certas RNAs podem mapear estatisticamente dois conjuntos qualquer. É difícil determinar o número apropriado de nós na camada escondida, e muitas vezes é determinado pela experimentação. Dispor de poucos nós na camada escondida enfraquece a RNA e sempre a impede de mapear corretamente as entradas para as saídas. Ao invés, muitos nós força o uso de tabelas de busca e impede a generalização. Em resumo, muitos nós permitem à RNA memorizar os padrões apresentados a ela sem extrair as características salientes. Isto é, quando são apresentados novos padrões à RNA, esta está inabilitada para processar o padrão apropriadamente, porque não descobriu os princípios fundamentais do problema (TOURETZKI e POMERLEAU, 1989).

Se existir uma única camada escondida, o tamanho inicial desejado é de 75% do tamanho da camada de entrada. Para mais de uma camada escondida, deve-se reduzir o tamanho de cada camada e atingir no possível resultados similares (PSALTIS et alii, 1988).

Depois de treinar a RNA, é importante testá-la com o conjunto de treinamento e com exemplos que a RNA nunca antes tenha encontrado. Como uma regra geral, para melhorar a exatidão da RNA sobre o conjunto de treinamento, deve-se incrementar o tamanho da camada escondida. Alternativamente, para melhorar a capacidade de generalização, e portanto melhorar o rendimento sobre casos novos, deve-se reduzir o tamanho da camada escondida. Os requerimentos para limitar o tamanho da camada escondida são indicados pela rápida deteriorização do rendimento da RNA. Um tamanho ótimo para a RNA é um equilibrio entre os objetivos da exatidão e a generalização para cada aplicação particular.

O número de nós de saída da RNA é determinado pelo paradígma conexionista usado e o tipo de saída esperada. Os paradígmas auto-associativos tem o mesmo número de entradas e saídas. Os paradígmas hetero-associativos geralmente tem menor número de nós de saída que nós de entrada.

A conectividade da RNA descreve como os nós serão ligados. A mais comúm é a RNA "feed forward" encontrada em "backpropagation" e Perceptron. As RNAs "feed forward" ligam os nós de uma camada com aqueles nós da camada posterior sem realimentação e sem conexões entre os nós da camada. Outros paradígmas, tais como a RNA de HOPFIELD e o mapeamento de KOHONEN, contam com conexões intra-camadas. No fim, redes como ART-1, ART-2 e "backpropagation" recorrente tem ligações de realimentação.

Dada a natureza da conectividade, seja "feed forward", intra-camada ou de realimentação, como os outros nós devem ser ligados?. A resposta é específica do paradígma. Por exemplo, Perceptron liga aleatoriamente as entradas com as saídas, BAM é totalmente inter-conectada, e a RNA de HOPFIELD é completamente intra-conectada. Geralmente dentro de uma RNA multi-camadas é melhor ter as camadas adjacentes totalmente conectadas. Este esquema prove uma maior flexibilidade quando o algorítmo de treinamento está procurando ajustar o peso adequado. O algorítmo de treinamento pode anular ligações desnecessárias ajustando seus peso em zero.

Depois de determinar os tipos de nós, o tamanho da RNA e a conectividade das camadas, deve-se identificar o ajuste dos parâmetros para o algorítmo de aprendizado.

Uma vez pronta a fase de projeto é necessário passar para a fase de implementação da RNA. Nesta fase devem estudar-se as ferramentas disponíveis para

simular as RNAs, tais como o simulador de ROCHESTER (PDP), e também deve ser analisada a possibilidade de implementar a RNA em processadores paralelos, tais como o Transputer. A decisão depende basicamente dos recursos disponíveis e da importância do projeto para operar em tempo real, dado que um simulador em máquinas sequenciais é sempre menos eficiente, por razões obvias, que aquele implementado numa máquina paralela.

No fim, uma vez implementado o sistema deve-se passar para a fase de manutenção, onde é necessário dispor de um manual de operação do sistema e um manual técnico para superar possíveis problemas no uso do sistema.

Deste estudo feito para determinar uma metodologia de desenvolvimento na implementação de RNAs em controle de processos, se tem como principal conclusão que as fases de treinamento e consulta não permitem o processamento em tempo real.

Esta é a motivação de propor um modelo que combine ambas as fases, permitindo treinar a RNA na fase de operação.

A quantidade de processamento num modelo deste tipo é muito elevada, portanto é indispensável o uso de técnicas de computação paralela para permitir uma resposta adequada em tempo real.