LEONARDO WELLISCH DE BARROS BARRETO

SEQUENCIAMENTO DA PRODUÇÃO ATRAVÉS DE REDES NEURAIS "SELF-ORGANIZING"

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia.

São Paulo

2000

LEONARDO WELLISCH DE BARROS BARRETO

SEQUENCIAMENTO DA PRODUÇÃO ATRAVÉS DE REDES NEURAIS "SELF-ORGANIZING"

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia.

Área de Concentração:
Engenharia Mecânica

Orientador:
Prof. Dr. Marcos R. P. Barretto

São Paulo

2000

página reservada para a ficha catalográfica (VERSO)

"Há um momento para tudo e um tempo para todo propósito debaixo do céu."
(Ecl. 3,1)

Aos meus pais, porque sempre colocaram nossa educação acima de seus objetivos pessoais.

À Georgeana, Alessandra, Rafaela (e a alguém que está chegando), porque compreenderam a importância deste trabalho para mim.

Amo vocês.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Marcos Ribeiro Pereira Barretto, pela sua confiança no meu trabalho e estímulo durante o processo de preparação desta dissertação. Agradeço também ao meu colega Nerone Marques de Brito Júnior, por todo o incentivo e cooperação ao longo do curso de mestrado.

Aos amigos da Sonopress-Rimo Ind. e Com. Fonográfica Ltda., por entenderem meus objetivos e me apoiarem incondicionalmente quando os deveres da vida acadêmica se sobrepuseram as minhas responsabilidades profissionais.

A todos os meus professores ao longo da minha vida acadêmica, por nutrirem em mim o desejo do aprendizado e o respeito por aqueles que se propõem à dedicar sua vida ao ensino.

À Universidade de São Paulo que, apesar das dificuldades existentes em nosso país, oferece aos cidadãos a possibilidade de acesso à cultura em seus graus mais elevados, dignificando o ensino público.

À minha família e a todos os meus amigos, que ao longo destes trabalho tantas vezes foram colocados em segundo plano.

À Deus, de Quem tudo recebemos, porque sem Ele nada podemos realizar.

SUMÁRIO

RESUMO

O problema de seqüenciamento de tarefas está presente nas mais diversas disciplinas do conhecimento e situações da vida prática. Notadamente, no ambiente industrial este problema se apresenta em inúmeras ocasiões onde sua resolução se torna crítica para que a corporação atinja seus objetivos e mantenha seus clientes satisfeitos.

Nos últimos anos, as redes neurais tem sido empregadas na resolução de diversos problemas considerados "difíceis", despertando o interesse tanto da comunidade acadêmica como do meio empresarial. Neste trabalho foi analisada a aplicação das redes neurais artificiais do tipo "Self-Organizing" para a resolução do problema de seqüenciamento de tarefas.

Este trabalho é iniciado com um exame do problema do seqüenciamento da produção, destacando-se as ferramentas comumente utilizadas em sua resolução. Após isso, é feita uma revisão sobre redes neurais, enfatizando-se as redes neurais do tipo "Self-Organizing". Em seguida é feita uma análise da aplicação destas redes ao problema de seqüenciamento e apresentado um modelo que possibilita esta aplicação. Finalmente, são apresentados obtidos através da implementação deste modelo em um computador do tipo PC através da linguagem de programação Visual Basic.

ABSTRACT

The problem of sequencing tasks is present in a multitude of knowledge disciplines and common situations in our everyday lives. Particularly in an industrial environment, this problem arises in various occasions when its solution is critical so that the corporation manages to reach its goals while keeping its customers satisfied.

Over the last years, the artificial neural networks have been employed in the resolution of many problems considered "hard", catching the attention of both academic and business communities. In this work, the application of "Self-Organizing" artificial neural networks to the task-sequencing problem is analyzed.

The present work begins examining the problem of sequencing tasks in a production environment, highlighting the tools commonly used on its resolution. After that, there is a revision on the subject of neural networks, emphasizing the "Self-Organizing" architecture.

Following that, the application of "Self-Organizing" neural networks to the sequencing problem is studied and a model is presented. Finally, the results obtained with an implementation of this model in a PC computer using the Visual Basic programming language are presented.

INTRODUÇÃO

1 Objetivos

Este trabalho tem como objetivo avaliar a aplicação de redes neurais do tipo "Self-Organizing" à solução do problema de seqüenciamento da produção. Para tanto, é utilizado um dos modelos de rede "Self-Organizing" apresentados pela literatura. Este modelo é utilizado para o seqüenciamento de conjuntos de ordens de produção e os resultados são comparados à outros métodos utilizados para a resolução do problema de seqüenciamento.

O ambiente de produção utilizado para avaliação da abordagem proposta é baseado numa situação real existente em uma determinada fábrica de CDs (Compact-Discs).

O ambiente de produção estudado é composto por uma máquina capaz de realizar uma operação específica em um determinado tipo de CD. As ordens a serem processadas possuem tempo de processamento e datas de entrega conhecidos. Estas ordens estão todas disponíveis para processamento no início do intervalo de seqüenciamento, caracterizando um ambiente estático. O principal objetivo de produção é o atendimento dos prazos de entrega, o que torna interessante a busca por uma redução no Número de Ordens em Atraso. De forma a melhor avaliar o modelo estudado, foi implementado um simulador na linguagem Visual Basic.

O modelo proposto objetiva oferecer soluções relativamente rápidas e com qualidade aceitável quando comparado à regras clássicas de seqüenciamento. Para tanto, é utilizado um modelo de rede neural do tipo "Self-Organizing", demonstrando a aplicabilidade deste método aos problemas combinatórios.

Assim como em outras áreas do conhecimento técnico, grande parte da literatura disponível está em língua estrangeira (predominantemente em inglês). Isto cria uma certa dificuldade na interpretação e tradução de termos utilizados no estudo da produção e de inteligência artificial. A nomenclatura utilizada aqui se baseia em grande parte em trabalhos como CALFAT (1990), CYTED (1994) e LUCENA (1987). Também foram essenciais as notas de aula das disciplinas cursadas na USP durante o curso de mestrado, em especial:

- "Fundamentos sobre manufatura automatizada – PMC815", 1996;
- "Percepção Robótica – PMC888", 1995;
- "Técnicas de apoio para projeto e desenvolvimento de sistemas de produção e de engenharia: Modelos de simulação, métodos heurísticos e sistemas especialistas", 1996 – PRO798;
- "CAD Inteligente – PMC867", 1996.

2 Justificativas do Trabalho

A motivação para este trabalho surgiu da experiência com a indústria de fabricação de "Compact-Discs" (CDs). Embora o processo de fabricação de CDs compreenda várias etapas (p/ Ex.: Masterização, Replicação, Acabamento, etc...), a área onde a programação se torna crítica é a fase de Replicação dos CDs, notadamente para alguns CDs específicos que requerem uma etapa extra de fabricação realizada em uma máquina especial. Algumas das características deste ambiente produtivo são:

- Existência de uma única máquina capaz de realizar esta operação especial;
- Tempo de processamento diretamente proporcional ao tamanho da ordem (quantidade de unidades à serem produzidas);

- Ordens de produção com tamanho variável;
- Sazonalidade da produção, com uma média anual de utilização da capacidade instalada em torno de 70%;
- Preocupação com o atendimento das datas de entrega.

Hoje a programação das máquinas é feita de forma manual cobrindo um horizonte de alguns dias. O sistema de controle utilizado mantém um histórico atualizado da situação das ordens e informa sobre os limites de capacidade globais diários. Torna-se interessante buscar um sistema capaz de auxiliar os programadores de forma rápida mesmo que utilizando critérios simples.

A aplicação das redes neurais se justifica na medida em que diversos trabalhos vem demonstrando a aplicabilidade deste método aos problemas de natureza combinatória como pode ser visto em BURKE et al. (1992). Entre as diversas arquiteturas de rede neural existentes, as redes do tipo "Self-Organizing", baseadas nos trabalho desenvolvidos por KOHONEN (1982, 1989 e 1997), apresentam interesse particular pois se baseiam em paradigmas semelhantes aos existentes no cérebro humano como pode ser visto com mais detalhes em DAMASIO (1994).

3 Organização do Trabalho

A organização deste trabalho é a seguinte:

No capítulo 1 são apresentados o objetivo do trabalho e os fatos que o motivaram.

No capítulo 2 é apresentada uma definição do problema de seqüenciamento da produção. Em seguida são descritas as diversas formas de classificação dos ambientes produtivos e as metodologias para avaliação da qualidade desta programação.

No capítulo 3 é realizada uma revisão da literatura sobre redes neurais, destacando-se o problema de seqüenciamento e as diversas propostas para a aplicação das redes na solução deste problema. Em seguida é apresentada a abordagem proposta para a resolução do problema de seqüenciamento.

No capítulo 4 são apresentados exemplos de simulações e resultados obtidos.

No Anexo A encontram-se as tabelas com os dados de entrada utilizados para avaliação do modelo.

No Anexo B encontram-se as seqüências obtidas com os dados de entrada.

No Anexo C encontram-se uma listagem do programa em Visual Basic utilizado.

No Apêndices encontram-se as resultados adicionais ilustrativos obtidos durante os testes.

CAPÍTULO 1 - SEQÜENCIAMENTO DA PRODUÇÃO

1.1 Definições e Suposições Básicas

De uma forma bastante sintética, MORTON (1993) define o conceito de "Scheduling" como sendo "a realização de um conjunto de tarefas que ocupam diversos recursos por um período de tempo".

Com base na definição apresentada por MORTON (1993), um sistema de "scheduling" deve ser capaz de tomar decisões sobre a alocação dos recursos disponíveis de maneira a completar as tarefas existentes dentro das restrições de tempo e qualidade especificadas. Paralelamente, o sistema deve maximizar a utilização dos recursos e reduzir os custos operacionais. As decisões básicas tomadas por um sistema deste tipo podem ser agrupadas da seguinte forma:

- Sequenciamento
- Alocação / Liberação
- Definição de roteiro

Ao longo deste trabalho, será focalizado o seqüenciamento onde, basicamente, o problema é definir a seqüência de processamento de um determinado conjunto de tarefas em uma determinada máquina. Este problema está formalizado nos textos de BAKER (1997) e CONWAY (1967).

A literatura utiliza o termo "Job" para indicar a unidade de produto ou serviço que será entregue ao cliente no final do processamento. Para efeito de analogia com o caso abordado neste trabalho, o termo Ordem de Produção (OP) será usado como sinônimo de "Job".

O mesmo raciocínio se aplica ao termo "Machine" que neste trabalho será definido como Recurso ou Máquina.

Diversas suposições e/ou restrições são empregadas na literatura como pode ser visto em CONWAY (1967), objetivando reduzir-se a complexidade do problema sem uma perda excessiva de generalidade. As suposições que serão utilizadas neste trabalho são:

- Cada máquina está continuamente disponível, não sofrendo paradas por quebra ou manutenção;
- As ordens são processadas de maneira ordenada e linear, não existindo bifurcações nos roteiros;
- Cada operação pode ser realizada por apenas um tipo de máquina, o que não impede a existência de máquinas em paralelo;
- Não é permitido que haja "preemption", ou seja, uma operação já iniciada deve ser completada antes que outra operação possa ser realizada por uma determinada máquina;
- Os tempos de processamento de operações sucessivas em uma mesma máquina não podem ser sobrepostos;
- Cada máquina pode processar apenas uma operação de cada vez.

1.2 Níveis de Programação

O problema de "scheduling" está presente em diversos níveis do ambiente industrial. Uma das propostas de classificação destes níveis é apresentada por MORTON (1993) conforme mostra a tabela à seguir:

Níveis do Problema de "Scheduling"		
Nível	**Exemplo de problema**	**Horizonte**
1 – Planejamento de Longo Prazo	Projeto da fábrica, expansão da fábrica	2 à 5 anos
2 – Planejamento de Médio Prazo	Regularização da produção	1 à 2 anos
3 – Planejamento de Curto-Prazo	Planejamento de materiais	3 à 6 meses
4 – "Scheduling"	Roteamento de processos, balanceamento de linhas, ajuste de tamanho de lotes	2 à 6 semanas
5 – "Scheduling" reativo / controle de chão-de-fábrica	Mudanças de prioridade, defeitos nas máquinas, materiais atrasados.	1 à 3 dias

Este trabalho se concentrará no Nível 4 onde se possui um razoável conhecimento *a priori* das tarefas à serem realizadas durante o horizonte a ser programado.

1.3 Ambiente de Sequenciamento

Embora a atividade de "scheduling" esteja presente em diversos níveis como apresentado anteriormente, este trabalho irá focalizar o estudo de métodos para seqüenciamento em ambientes produtivos.

1.3.1 Classificação dos Ambientes de Sequenciamento

Diversos autores como MORTON (1993), PINEDO (1995) e SANTORO; PACHECO (1999) apresentam propostas para a definição e classificação do ambiente onde irá se configurar o problema de "scheduling". Dependendo do tipo de ambiente, o "scheduling" englobará diversas atividades, desde a escolha de roteiros até o seqüenciamento de tarefas.

Alguns autores como SANTORO; PACHECO (1999) apresentam uma divisão básica entre o ambiente mais geral conhecido como "Job Shop", onde se admite uma variação nos roteiros de produção de uma determinada ordem e o ambiente denominado "Flow Shop", onde as operações são estritamente ordenadas e seguem um fluxo definido. Esta divisão básica é comumente subdividida de várias formas, sendo uma delas a apresentada por MORTON (1993):

Ambientes de Resolução do Problema de Sequenciamento	
Tipo de Ambiente	**Características**
"Job Shop" Fechado	Produção discreta, fluxo complexo, produtos com características únicas, sem partes em comum
"Job Shop" Aberto	Produção discreta, fluxo complexo, alguns produtos similares e/ou com partes em comum
"Batch Shop"	Produção discreta ou contínua, fluxo menos complexo, produtos similares e com partes em comum
"Flow Shop"	Produção discreta ou contínua, fluxo linear, produtos bastante similares
"Batch Shop"	Produção discreta ou contínua, fluxo menos complexo, produtos similares e com partes em comum

Ambientes de Resolução do Problema de Sequenciamento (continuação)	
Tipo de Ambiente	**Características**
Célula de Manufatura	Produção discreta, é uma versão automatizada e agrupada de uma "Job shop" aberta ou uma "Batch shop"
"Assembly Shop"	É uma versão de montagem de uma "Job shop" aberta ou uma "Batch shop"
Linha de Montagem Tradicional	Alto volume, baixa variedade; é uma versão de linha de transferência de uma "Assembly shop"
Linha de Transferência	Volume muito alto e baixa variedade com produção linear e automatizada
Linha de Transferência Flexível	Tentativa moderna de se trazer as vantagens dos modelos de alto volume para o modelo de "Job shop"

Com o objetivo de melhor ilustrar este trabalho, são apresentados a seguir alguns detalhes relativos a estes ambientes.

1.3.2 "Job Shop"

Uma das formas de se classificar os problemas de "Job shop" é quanto à forma de atendimento da demanda, conforme apresentado por HAX (1984) e SANTORO; PACHECO (1999), classificando-se os "Job Shops" em fechados, abertos e mistos.

1.3.2.1 "Job Shop" Fechados

Neste modelo, o planejador da produção possui um certo conhecimento sobre o padrão de demanda dos produtos à serem fabricados. Isto possibilita que estoques de sejam formados para atender aos pedidos de forma mais rápida. Neste modelo, é muito

importante um acompanhamento constante dos estoques (não só de produtos acabados como também de materiais intermediários) para evitar riscos como obsolescência e perda de validade de determinados componentes perecíveis. Um exemplo pode ser uma linha de montagem de microcomputadores onde alguns modelos de maior demanda são montados em quantidades maiores do que as requeridas por um determinado pedido de um cliente para que o tempo de entrega seja reduzido em novos pedidos.

Uma variação sobre os "Job Shops fechados são os "Batch Shops" onde roteiros de produção simplificados aliados à uma demanda estável tornam atraente a construção de estoques razoavelmente elevados como ocorre em algumas indústrias químicas.

1.3.2.2 "Job Shop" Abertos

Ao contrário dos "Job Shops" fechados, neste ambiente quase todos os produtos são extremamente individualizados e construídos por encomenda, dificultando não só o planejamento como também qualquer tentativa de criação de estoques de produtos acabados. Na prática, "Job Shops" puramente abertos não são encontrados uma vez que os planejadores eventualmente encontram uma forma de detectar alguns padrões na demanda.

Finalmente, os "Job Shops" mistos apresentam características de ambos os sistemas.

1.3.3 "Flow Shop"

O "Flow-Shop" é semelhante a um "Batch-shop" com um fluxo linear. No caso mais simples, cada produto requer as mesma atividades, na mesma seqüência, no mesmo conjunto de máquinas. Numa "Flow-Shop" Composta, cada máquina na seqüência pode

ser substituída por um conjunto de máquinas em paralelo. Empresas engarrafadoras constituem um exemplo deste tipo de produção. O processo geral de replicação de CDs se encaixa neste modelo.

Existem modelos que combinam o "Batch Shop" e o "Flow Shop" como na indústria alimentícia, onde é comum a existência de processos industriais com fases distintas. Inicialmente se preparam grandes quantidades de alguns produtos intermediários que são combinados nos produtos finais e, numa etapa posterior, são embalados em equipamentos semelhantes. Se observado como um todo, o processo de replicação e posterior embalagem de CDs constitui um caso de "Batch/Flow-Shop".

1.3.4 Ambientes Híbridos

São aqueles ambientes que combinam características encontradas nos "Job Shops" e nos "Flow Shops" de forma a atender necessidades industriais específicas.

a) Célula de Manufatura: Este modelo tenta combinar a flexibilidade da "Job-Shop" com o baixo custo associado à uma "Flow-Shop". As máquinas costumam ser controladas por sistemas automáticos que possibilitam grande flexibilidade de roteiros. Um exemplo são os centros de usinagem com controle numérico.

b) "Assembly-Shop": É uma "Job-Shop" aberta ou uma "Batch-Shop" em que conjuntos intermediários são montados até se atingir o produto final.

c) Linha de Montagem Tradicional: É uma versão da "Assembly-Shop" com volumes menores e uma menor variedade de produtos. Os componentes em processo são movimentados através de sistemas como esteiras rolantes e os estoques intermediários são minimizados. Em geral, empregam grande quantidade de trabalho humano.

d) Linha de Transferência: É um modelo onde os volumes são bastante altos e a variedade bem pequena. Costumam ser extremamente automatizadas. Como exemplo temos as células de fabricação de CDs ("Monoliners") onde diversas máquinas (injetora, metalizadora, laqueadora, impressora e estação de testes) funcionam sincronizadas e o produto é transportado automaticamente de uma máquina à outra.

e) Linha de Transferência Flexível: É uma variação sobre as linhas de transferência tradicionais onde sistemas eletrônicos sofisticados possibilitam algumas variações de roteiro.

1.4 Terminologia Utilizada nos Problemas de Sequenciamento

A literatura apresenta algumas variações para a nomenclatura utilizada nos problemas de seqüenciamento. Neste trabalho, será utilizada uma convenção baseada na proposta de CONWAY (1967), a qual é utilizada em diversos trabalhos como MORTON (1993). Para tornar esta convenção mais atualizada e situá-la no contexto da nossa linguagem, foram utilizadas como referência os trabalhos de CALFAT (1990), SANTORO; PACHECO (1999) e TEDESCHI (1997). Dentro desta proposta, o problema é definido como:

- Máquina: é a unidade produtiva capaz de realizar operações;
- Operação: é uma tarefa à ser executada em uma máquina;
- Ordem: é um conjunto de operações necessárias a obtenção de um determinado produto, podendo ser composta por uma única operação;

Diversas variáveis são utilizadas nos problemas de seqüenciamento, entre as quais:

c_j → Data de término do processamento da j-ésima ordem. A representação é originária do inglês "Completion time";

C_{max} → Quantidade de tempo utilizada entre o início do seqüenciamento e o término da última ordem. Na literatura, este termo muitas vezes é tratado pela denominação em inglês "Makespan";

d_j → Data de entrega da j-ésima ordem. A representação é originária do inglês "Due date";

id_j → Identificador da j-ésima ordem;

N → Número total de ordens;

p_{ij} → Quantidade de tempo necessária para o processamento da j-ésima ordem. Para problemas onde todas as ordens possuem uma única operação, podemos adotar o sub-escrito *j* para indicar a ordem à qual se faz referência. Em sistemas onde o tempo de processo depende da máquina onde a atividade será realizada, utiliza-se a simbologia p_{ij}, onde *i* identifica a máquina onde a operação será realizada;

r_j → Data de liberação para produção da j-ésima ordem. A simbologia vem do inglês "Release date";

s_j → Data de início do processamento da j-ésima ordem, do inglês "Start Date";

z_j → Índice de prioridade da j-ésima ordem. Quanto maior o valor de z_j, maior a prioridade;

w_j → Peso da j-ésima ordem. Calculado de acordo com a prioridade z_j da ordem. Quanto maior o valor de w_j, maior a penalidade em caso de atraso. Em geral, tem-se:

$$w_j = \frac{z_j}{\sum_{j=1}^{N} z_j} \quad \text{levando-se em conta que:} \quad \sum_{j=1}^{N} w_j = 1$$

onde N representa a quantidade total de ordens.

1.5 Classificação dos Problemas de Sequenciamento

O caso mais simples dos problemas de seqüenciamento ocorre quando existe apenas uma máquina onde as tarefas serão executadas. Mesmo nesta situação onde a complexidade é potencialmente baixa, para que seja possível uma análise do problema, diversas hipóteses simplificadoras são aplicadas. A aplicação ou não destas hipóteses, permite classificar os problemas de seqüenciamento da seguinte forma como proposto por MORTON (1993):

- Quanto a natureza de chegada das ordens, podendo ser estáticos ou dinâmicos
- Quanto a permitirem ou não que ordens sejam interrompidas após iniciado seu processamento ("Preemption")
- Quanto ao tipo de objetivo, podendo ser regulares ou não-regulares.

1.5.1 Estáticos x Dinâmicos

Os problemas Estáticos são aqueles onde todas as atividades a serem programadas já são conhecidas no início do intervalo de programação. Ou seja, não surgirão e nem serão canceladas ordens de produção ao longo do intervalo de programação.

Já os problemas Dinâmicos são aqueles onde as atividades existentes no início do intervalo de programação poderão ser canceladas e novas atividades poderão ser liberadas para programação após o início do intervalo de sequenciamento.

1.5.2 "Preemptive" x "Non-Preemptive"

Os problemas do tipo "Preemptive" são aqueles onde as atividades podem ser interrompidas para que atividades com prioridades mais elevadas sejam processadas. Neste caso, torna-se importante definir a política pela qual o custo desta interrupção será incorporado à avaliação do objetivo que se deseja alcançar. Em alguns casos, adota-se que as atividades interrompidas podem ser reiniciadas sem que isto gere custos extras.

1.5.3 Objetivos Regulares x Não-Regulares

Esta é uma classificação que leva em conta o tipo de objetivo à ser alcançado através do seqüenciamento.

Problemas com objetivos Regulares são aqueles baseados nos tempos de término das tarefas à serem programadas. Ou seja, é sempre preferível terminar uma atividade o quanto antes. Como demonstrado por CONWAY (1967), estes objetivos podem ser escritos como uma função $M = f(C_1, C_2, \ldots, C_n)$ que aumenta se pelo menos um dos tempos de término C_j aumentar, ou seja:

$M' > M$ se e somente se $C'_j > C_j$ para pelo menos um j, $1 \leq j \leq n$

Já nos problemas com objetivos Não-Regulares, outras medidas de desempenho, como por exemplo a utilização do sistema, podem se tornar prioritárias.

1.6 Variáveis de Desempenho

A literatura apresenta diversas grandezas (também chamadas "variáveis de desempenho") para avaliação da situação de uma determinada tarefa alocada em uma

seqüência. Ao longo deste trabalho, serão utilizadas algumas das variáveis propostas por CONWAY (1967) e MORTON (1993) como pode ser visto na tabela à seguir:

Variáveis de Desempenho			
Variável	**Símbolo**	**Significado**	**Formulação**
Tempo de Execução	C_j	Neste trabalho, estaremos tratando de ordens compostas por uma única operação. Portanto será utilizado o subscrito *j* para identificar o tempo de execução da ordem *j*	---
Tempo de Fluxo	F_j	É a quantidade de tempo total durante o qual a ordem *j* permanece no sistema	$F_j = C_j - r_j$
Desvio	L_j	É a quantidade de tempo (positiva ou negativa) que o Tempo de Execução da ordem *j* se desviou da Data de Entrega	$L_j = C_j - d_j$
Atraso	T_j	É o atraso da ordem *j*	$T_j = \max\{0, L_j\}$
Adiantamento	E_j	É o adiantamento da ordem *j*	$E_j = \max\{0, -L_j\}$

1.7 Funções Objetivo

Como descrito no item anterior, as variáveis de desempenho servem para verificar a situação de tarefas específicas. Em geral, o problema de seqüenciamento envolve a avaliação de um conjunto de tarefas em uma determinada seqüência e não apenas de tarefas individuais. Para que se possa avaliar a qualidade de um determinado seqüenciamento, as variáveis acima podem ser combinadas em "funções objetivo" (ou "critérios de desempenho"). Neste trabalho, serão utilizadas algumas funções objetivo como as apresentadas por MORTON (1993) e TEDESCHI (1997). A tabela à seguir mostra uma descrição de diversas funções encontradas na literatura:

Funções Objetivo		
Função	**Símbolo**	**Formulação**
Tempo Máximo de Execução [1]	C_{max}	$C_{max} = \max\{C_j\}$
Tempo de Fluxo Ponderado	F_{wt}	$F_{wt} = \sum_j w_j F_j$
Desvio Médio	L_{avg}	$L_{avg} = \frac{\sum_j L_j}{N}$
Desvio Máximo	L_{max}	$L_{max} = \max\{\lvert L_j \rvert\}$
Desvio Ponderado	L_{wt}	$L_{wt} = \sum_j w_j L_j$
Atraso Máximo	T_{max}	$T_{max} = \max\{T_j\}$
Atraso Total	T_{tot}	$T_{tot} = \sum_j T_j$
Atraso Ponderado	T_{wt}	$T_{wt} = \sum_j w_j T_j$
Adiantamento Ponderado	E_{wt}	$E_{wt} = \sum_j w_j E_j$
Adiantamento e Atraso Ponderado	ET_{wt}	$ET_{wt} = \sum_j (w_{Tj} T_j + w_{Ej} E_j)$
Tempo de Fluxo Máximo	F_{max}	$F_{max} = \max\{F_j\}$
Número de Ordens em Atraso	N_{tot}	$N_{tot} = \sum_j \delta(T_j)$ Onde $\delta(x) = 1$ se $x > 0$ $\delta(x) = 0$ se $x \leq 0$
Número Ponderado de Ordens em Atraso	N_{wt}	$N_{wt} = \sum_j w_{Nj}\, \delta(T_j)$ Onde $\delta(x) = 1$ se $x > 0$ $\delta(x) = 0$ se $x \leq 0$

[1] ou Horizonte de Programação (do inglês, "Makespan")

1.7.1 Características de Algumas Funções Objetivo

1.7.1.1 Tempo Máximo de Execução (C_{max})

Em geral, o seqüenciamento com o menor tempo total de fabricação é também aquele com a maior utilização de recursos. Este objetivo pode ser empregado de forma bem abrangente para problemas onde existe uma única maquina mas é limitado na aplicação a sistemas com várias máquinas pois não é positivamente afetado por máquinas que terminam suas tarefas antes do previsto.

1.7.1.2 Tempo de Fluxo Ponderado (F_{wt}) e Desvio Ponderado (L_{wt})

São objetivos de aplicação simples e que produzem seqüências boas na maioria dos casos, mesmo naqueles onde existem outros objetivos mais importantes.

1.7.1.3 Atraso Ponderado (T_{wt})

Embora este objetivo seja representativo em muitas situações, os problemas que se baseiam nele são de difícil solução.

1.7.1.4 Tempo de Fluxo , Desvio e Atraso Máximos (F_{max} , L_{max} , T_{max})

São objetivos relativamente simples que fornecem uma solução inicial para diversos outros problemas. Existem trabalhos interessantes sobre T_{max} como LAWELER et al. (1981) e LENSTRA et al. (1977).

1.7.1.5 Número Ponderado de Ordens em Atraso (N_{wt})

Este é um objetivo importante quando se deseja limitar as penalidades num ambiente onde a existência de ordens em atraso é praticamente inevitável como no caso estudado neste trabalho. Problemas baseados neste tipo de objetivo são de difícil solução pois o esforço de resolução cresce de forma exponencial com o tamanho do problema.

1.7.2. Pesos

Em muitos caso, por falta de dados históricos, os pesos (w_j) são definidos como *1/n*. Neste caso, o valor ponderado passa à ser apenas um valor médio. Uma variação sobre esta estratégia é definir $\sum_j w_j = 1$. O ideal é que os pesos representem efetivamente a penalidade associada ao não alcance de um determinado objetivo.

1.8 Métodos Utilizados para o Problema de Sequenciamento

O problema de "scheduling" tem sido estudado de forma bastante detalhada nas últimas décadas através de diversas abordagens como pode ser exemplificado por trabalhos como BUFFA (1975), FOX; ZWEBEN (1994), KUNG; MARSDEN (1995), LANE; EVANS (1995), LITTLE; HEMMINGS (1994) e MAMALIS; MALAGARDIS (1994).

1.8.1 Métodos de Simulação

A simulação tem como vantagem permitir que sistemas reais sejam representados à custos computacionais relativamente baixos conforme exemplificado por BAKER (1974

e 1997), KIM (1994), KLAFEHN et al. (1996) e MOORE; WILSON (1967). Outra vantagem é permitir uma maior interação com o conhecimento de especialistas humanos. A desvantagem consiste no fato de que os resultados são em geral distantes da solução ótima para o problema. Além disso, existe uma grande dificuldade em se avaliar os resultados obtidos.

1.8.2 Métodos Matemáticos

Na maioria das vezes, os métodos matemáticos buscam soluções exatas. O grande aumento da capacidade de processamento dos computadores na década de 60 impulsionou bastante os métodos baseados em programação inteira como apresentado por BALAS (1965 e 1967) apud MORTON (1993) e, em seguida, aqueles baseados em programação dinâmica à exemplo de SRINIVASAN (1971) apud MORTON (1993). Em qualquer caso, apenas problemas de dimensão limitada podem ser resolvidos de maneira exata pois, em geral, a complexidade computacional dos problemas de seqüenciamento cresce exponencialmente com o tamanho dos mesmos, inviabilizando a obtenção de soluções exatas para problemas maiores.

Este fato fez com que se pesquisasse variações sobre os métodos matemáticos as quais, em conjunção com diversas heurísticas, possibilitassem a solução destes problemas.

1.8.3 Métodos Heurísticos

Como explicado acima, o objetivo destes métodos é contornar as dificuldades existentes na busca de soluções exatas para os problemas de seqüenciamento. Em geral, são

utilizadas técnicas de busca "melhoradas" por diversas heurísticas que buscam reduzir o espaço de soluções possíveis. Entre os métodos mais estudados podemos citar:

1.8.3.1 Métodos Manuais

Os métodos manuais podem ser divididos em dois tipos básicos: intervalo e despacho. No primeiro caso, a programação costuma ser feita partindo-se das datas de entrega e voltando-se no tempo de forma a criar uma seqüência de tarefas que permita o atendimento das datas de entrega. Já no seqüenciamento manual por despacho, as decisões são tomadas na direção normal do tempo através de diversas heurísticas baseadas na experiência existente.

Algumas destas regras se baseiam na programação de uma única máquina. Apesar da simplicidade deste caso, o mesmo serve como base para o desenvolvimento de diversos métodos voltados para o seqüenciamento de problemas mais complexos. CONWAY (1967) e MORTON (1993) propõe um conjunto de condições para a caracterização deste caso. Um resumo destas condições é apresentado abaixo:

- Existe apenas uma máquina e esta não sofre interrupções;
- Cada ordem é composta por uma única operação;
- A quantidade de ordens (N) é finita;
- Todas as ordens devem ser seqüenciadas;
- A quantidade de ordens é conhecida antes que o processo de seqüenciamento seja iniciado;
- Todas as ordens são disponibilizadas simultaneamente;
- Os tempos de processamento (p_j) são arbitrários, independentes da seqüência e conhecidos *a priori*.

- O tempo de preparação pode ser incluído no tempo de processamento;
- O tempo no qual a ordem é terminada (C_j) é igual ao tempo de processo (p_j) já que o tempo começa a ser contado quando a ordem se torna disponível;
- As máquinas não sofrem quebras ou deixam de estar disponíveis por outras razões.

Como demonstrado por CONWAY (1967), desde que o seqüenciamento seja avaliado através de objetivos regulares não é necessário considerar programações que envolvam "preemption" a inserção de tempos de espera.

Uma regra de prioridade bastante utilizada é conhecida como SPT[2], a qual determina o seqüenciamento das ordens de forma crescente com os tempos de processamento (p_j). De acordo com os resultados apresentados por CONWAY (1967) e MORTON (1993), esta regra minimiza o tempo médio de permanência (F_{wt}) no sistema. Esta regra também minimiza o desvio médio (L_{avg}) segundo apresentado por BAKER (1997).

Outra regra que fornece resultados úteis é a EDD[3], a qual determina o seqüenciamento das ordens de forma crescente de datas de entrega (d_j). Como demonstrado em diversos trabalhos, notadamente em CONWAY (1967) e BAKER (1993), a regra EDD fornece o resultado ótimo quando o objetivo à ser alcançado é minimizar o desvio máximo (L_{max}). Para o caso estático, a regra EDD também minimiza o atraso máximo (T_{max}).

1.8.3.2 "Neighborhood Search"

A "Busca na Vizinhança" é um método genérico no qual se inicia com uma solução obtida através de um método qualquer e em seguida se tentam diversas pequenas

[2] SPT: Do inglês "Shortest Processing Time"

[3] EDD: Do inglês "Earliest Due Date"

mudanças na programação buscando uma melhoria da solução como no artigo de WILKERSON; IRWIN (1971). Se nenhuma melhoria é obtida, o método está encerrado. Caso contrário, a melhor mudança é escolhida como a nova solução e o método recomeça.

1.8.3.3 "Tabu Search"

Esta é uma variação do método anterior onde se mantém uma lista das etapas anteriores visando evitar movimentos que sabidamente não trazem melhoria. Este método é explorado por diversos autores como GLOVER (1990) e WIDMER; HERTZ (1989).

1.8.3.4 "Simulated Annealing"

Esta é uma outra variação sobre a Busca na Vizinhança. Neste caso, um valor aleatório é adicionado a cada nova avaliação da função objetivo visando escapar de mínimos locais. Este valor é reduzido à medida em que o método avança numa tentativa de simulação dos processos de resfriamento de ligas metálicas onde a movimentação entre as fases de uma determinada liga diminui à medida que a liga sofre um resfriamento. O trabalho de KIRKPATRICK et al. (1983) é uma das principais referências na aplicação deste método a problemas de otimização. Outras referências úteis são ISHIBUCHI et al. (1991) apud MORTON (1993), MAMALIS; MALAGARDIS (1996), VAN LAARHOVEN; AARTS (1987) e YIP; PAO (1995).

1.8.3.5 Algoritmos Genéticos

Neste método, populações de soluções são submetidas a um processo evolutivo no qual as melhores soluções de uma determinada geração produzem novas soluções que combinam características da geração atual. Algumas referências importantes são FALKENAUER; BOUFFOIX (1991), HOLLAND (1975) apud MORTON (1993) e UCKUN et al. (1993). Nos últimos anos, o interesse por este tipo de metodologia tem aumentado à medida em que o interesse do grande público se volta para projetos como o mapeamento do DNA humano. Assim como acontece com as redes neurais, os algoritmos genéticos tem se beneficiado do substancial aumento na capacidade de processamento dos computadores.

1.8.3.6 Enumeração Parcial ("Beam Search")

Este método buscam simplificar a busca em árvores de soluções através da eliminação de ramos provavelmente ruins. Uma das aplicações utilizada para avaliar este tipo de sistema são jogos tradicionais como o xadrez, onde uma busca exaustiva por todas as possíveis soluções é inviável como no problema de seqüenciamento. A dificuldade destes métodos reside em se definir o que é "provavelmente" ruim. Existem diversas abordagens para este tipo de solução como as apresentadas por ANTHONY; SCHAFFER (1990), OW; MORTON (1988).

1.8.3.7 Métodos de Gargalo

Estes métodos buscam concentrar o esforço computacional naqueles recursos (ou máquinas) que apresentam a maior carga, os chamados "gargalos" (em inglês "bottlenecks"). Um variação sobre estes métodos proposta por ADAMS et al. (1988) busca alterar o "gargalo" ao longo do processo à fim de otimizar a solução. Um sistema comercial que ganhou bastante popularidade na década de oitenta foi o OPT, o qual está descrito em trabalhos como os de FOX (1994) e LUNDRIGAN (1986) apud MORTON (1993).

1.8.3.8 Sistemas Especialistas

Estes sistemas tentam simular o conhecimento utilizado por profissionais na tomada de decisões. Alguns exemplos de sistemas que utilizam esta metodologia são o OPAL, discutido por BENSANA et al. (1988) e o sistema ISIS, descrito nos trabalhos de FOX; SMITH (1984) e FOX; ZWEBEN (1994).

1.8.3.9 Redes Neurais

Basicamente, são sistemas que tentam simular a alta capacidade de processamento paralelo do cérebro humano. Uma variação deste método será discutida mais profundamente neste trabalho. A literatura sobre redes neurais é extensa e alguns textos fornecem uma boa visão geral como CHESTER (1993), FREEMAN et al. (1992), KUNG (1993), MAREN at al. (1990), SMITH et al. (1998) e SMITH (1999). Exemplos

de aplicação deste método na resolução de problemas podem ser encontrados em EL GHAZIRI (1991) e PERFETTI (1995).

1.8.4 Sequenciamento de Uma Única Máquina

TEDESCHI (1997) apresenta um resumo com algoritmos polinomiais utilizados na resolução de alguns problemas para o caso estático determinístico, destacando os casos para os quais não existe um algoritmo disponível.

Dentro do universo dos problemas de "scheduling", pode-se identificar um classe especial de problemas onde as decisões à serem tomadas se relacionam unicamente à seqüência em que um conjunto de tarefas será executado. No caso mais básico, existe apenas um recurso ou máquina aguardando para processar a seqüência de tarefas aguardando em uma fila. Apesar da aparente simplicidade deste caso, o mesmo serve como base para um estudo dos métodos utilizados para a resolução dos problemas mais gerais de "scheduling" por permitir que os aspectos importantes deste problema sejam vizualizados com mais clareza.

Segundo BAKER (1997), o problema básico de uma única máquina pode ser caracterizado pelas seguintes condições:

- Um conjunto de *N* tarefas independentes e compostas por uma única operação está disponível no início do intervalo de programação;
- Os tempos de preparação ("setup") das tarefas são independentes da seqüência e estão incluídos nos tempos de processamento;
- As tarefas possuem tempo de processo determinados e são conhecidas no início do intervalo de seqüenciamento;

- A máquina está continuamente disponível e nunca é mantida desocupada. Ou seja, não há inserção de esperas;

Uma vez que uma tarefa é iniciada, esta não é interrompida até que termine. Ou seja, o sistema não admite "preemption".

CAPÍTULO 2 - REVISÃO DA LITERATURA

2.1 Problemas Combinatórios

Os problemas combinatórios surgem em diversas áreas e, em geral, são característicos das situações onde se deseja definir seqüências de eventos. Estes problemas variam em complexidade desde aqueles que permitem soluções razoavelmente simples do ponto de vista matemático até aqueles conhecidos como Np-completos. Uma introdução aos problemas combinatórios e sua complexidade pode ser encontrada em BOSE; LIANG (1996). No trabalho compilado por CRESCENZI; KANN (1998), são apresentados diversos exemplos de problemas combinatórios.

2.1.1 Problemas Np-completos:

Muitos problemas de otimização do tipo Np-completo são caracterizados pela existência de diversos mínimos locais. Esta característica explica em parte a dificuldade na sua resolução como discutido em BOSE; LIANG (1996) e RITTER et al. (1992).

Para que um problema seja tratável de maneira eficiente do ponto de vista matemático, deve existir um algoritmo determinístico capaz de gerar uma solução através de um esforço computacional que não cresça assintoticamente mais rapidamente que de forma polinomial com o tamanho do problema. Este tipo de problema é classificado como do tipo P (Polinomial determinístico).

Já os problemas do tipo NP (Polinomial não-determinístico) são caracterizados quando a verificação da "certeza" de uma solução é verificável com um esforço computacional que cresça no máximo de forma polinomial com o tamanho do problema.

Uma variação dos problemas do tipo NP são aqueles do tipo Np-completos conforme exemplificado em REINHARDT; MULLER (1991). Estes problemas são caracterizados como sendo no mínimo tão difíceis quanto outros problemas do tipo NP e por não serem solucionáveis de forma determinística num tempo polinomial.

Um dos exemplos mais comum de problema do tipo Np-completo é o Problema do Caixeiro Viajante, também conhecido como TSP (do inglês "Traveling Salesperson Problem"). Neste problema, o esforço computacional cresce de maneira exponencial com o tamanho do problema.

2.1.2 Problemas de Otimização Quadrática

O problema de seqüenciamento pode ser definido como um problema de otimização quadrática. Estes problemas possuem a forma geral descrita por SMITH (1996) e SMITH et al. (1998):

$$\text{minimizar } F(\mathbf{X}) = \sum_{k=1}^{N}\sum_{j=1}^{M} X_{k,j} \sum_{i=1}^{N}\sum_{l=1}^{N} Q(k,j,i,l) X_{i,l} + \sum_{k=1}^{N}\sum_{j=1}^{M} C(k,j) X_{k,j}$$

$$\text{sujeito à} \quad \sum_{j} X_{k,j} = 1 \qquad \forall k = 1,\ldots,N$$

$$\sum_{k} X_{k,j} = D_j \qquad \forall j = 1,\ldots,M$$

$$X_{k,j} \in \{0,1\}$$

Onde $X_{k,j}$ representa a k-ésima linha e j-ésima coluna de uma matriz 0-1 com dimensão N x M, a função objetivo $F(\mathbf{X})$ é uma função quadrática que representa o custo da matriz de solução **X**, C(k , j) representa o custo associado com a existência de um "1" na posição $X_{k,j}$ e Q(k, j, i, l) representa o custo associado com a existência de um "1"

simultaneamente nas posições $X_{k,j}$ e $X_{i,l}$. D_j representa a "demanda" da coluna j e pode ser definido como:

$$\sum_{j=1}^{M} D_j = N$$

A matriz **X**, também chamada de "matriz de permutação", composta por uma determinada seqüência de N vetores com dimensão M, com o formato:

(1,0,0,...,0)

(0,1,0,...,0)

...

(0,0,0,...,1)

Para o problema de seqüenciamento, assim como para o problema do caixeiro viajante, $D_j = 1, \forall j = 1,...,M$. Desta forma, cada um dos vetores acima é representado apenas uma única vez na matriz de permutação.

2.2 Redes Neurais

2.2.1 Introdução as Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) surgiram do esforço de se tentar utilizar a capacidade de processamento dos computadores digitais para a resolução de problemas que, intrinsecamente, não se adaptam ao paradigma do processamento em série como discutido por FREEMAN et al. (1992).

Embora as RNAs tenham apenas alguns pontos de analogia com os sistema Neurais Biológicos reais, pode-se dizer que elas em maior ou menor grau simulam a forma de

funcionamento e as estruturas de processamento existentes no córtex animal o qual, em última análise, é um sistema distribuído de processamento em paralelo. Uma abordagem recente do funcionamento do cérebro humano pode ser encontrada em DAMASIO (1994).

Muitos problemas de interesse, como o de seqüenciamento da produção, se encaixam numa classe de problemas que, por sua natureza, requerem uma modelagem de processamento em paralelo, o que motivou o desenvolvimento das RNA.

Ao longo das últimas décadas as RNAs passaram por fases alternadas de entusiasmo e descrédito. Isto porque os diversos estudos realizados assim como a literatura sobre o assunto muitas vezes eram obscuros e apresentavam resultados difíceis de serem verificados por outros pesquisadores. Em parte, isto acontecia devido ao limitado poder de processamento dos computadores existentes.

SMITH (1999a) propõe a divisão do desenvolvimento das redes neurais em 5 estágios conforme reproduzido abaixo:

Histórico de Desenvolvimento das Redes Neurais		
Ambiente Científico		**Ambiente Empresarial**
Babbage inventa o princípio da máquina analítica (1834) William James escreve "Psychology"(1890) Trabalho de Pavlov sobre aprendizado condicional (1904) Alan Turing usa o cérebro como paradigma computacional (1936) McCulloch e Pitts escrevem artigo sobre Neurônios (1943)	**ESTÁGIO 1**	(1914) Fundada a IBM
Wilkes projeta o primeiro computador baseado em programas armazenados (1946) Donald Hebb escreve "The Organization of Behavior": Regra de aprendizado "Hebbiana" Minsky constrói o primeiro Neurocomputador (1954) Projeto de verão na Universidade de Dartmouth: Lançados os campos de pesquisa em Inteligência Artificial e Redes Neurais (1956) Rosenblatt desenvolve o "Perceptron" (1957)	**ESTÁGIO 2**	(1954) A General Electric Co. utiliza o primeiro sistema computadorizado de folha de pagamento
Minsky e Papert escrevem "Perceptrons" (1969) Werbos propõe "Backpropagation" (1974) Trabalho de Willshaw e von der Malsburg sobre "Self-Organization" (1976) Trabalho de Hopfield em memória associativa (1982) Trabalho de Kohonen em "Self-Organizing Maps" (1982)	**ESTÁGIO 3**	(1971) A Intel Corp. desenvolve o primeiro microprocessador (1975) Fundada a SPSS Inc. e Nestor Inc. (1976) Formado o Instituto SAS (1977) Fundada a Apple (1981) Lançamento do IBM PC

Histórico de Desenvolvimento das Redes Neurais (continuação)		
Ambiente Científico		**Ambiente Empresarial**
A DARPA (Defense Advanced Research Projects Agency) patrocina Redes Neurais (1983) Descoberto o aprendizado por "Backpropagation" (1985) Trabalho de Rumelhart e McClelland sobre Processamento Paralelo Distribuído (1986) IEEE realiza sua primeira conferência internacional sobre Redes Neurais (1987) Fundado o periódico "Neural Networks" (1988) Fundado o periódico "Neural Computation" (1989) Fundado o periódico "IEEE Transactions on Neural Networks" (1990)	**ESTÁGIO 4**	(1987) Fundada a Neuralware Inc.
A pesquisa prossegue com ênfase nas necessidades empresariais	**ESTÁGIO 5**	(1991) Redes neurais são utilizadas em sistemas bancários (1993) Formada a Neuraltech Inc. (1993) Lançado o NeuroShell pela Ward Systems Group,Inc. (1994) A IBM lança o "Neural networks Utility" (1004) A Apple lança o "Newton Message Pad" (1995) A IBM forma a divisão de "Business Intelligence" (1995) Formada a Trajecta Inc. (1995) A Neuralware Inc. lança o NeuralWorks (1995) 95% dos bancos americanos usam técnicas de "data mining" (1997) Lançado o SAS Enterprise Miner (1998) A IBM anuncia investimentos de US$70 bilhões no mercado de "Business Intelligence"

No últimos anos, com o aumento quase exponencial da capacidade de processamento digital, foi possível a implementação de algoritmos que simulam de forma bastante eficiente o processamento em paralelo. Além disso, surgiram novas possibilidades de codificação de algoritmos diretamente em circuitos eletrônicos concebidos para o processamento em paralelo. Estes avanços trouxeram um novo interesse na tecnologia de Redes Neurais como comenta LOOI (1992).

2.2.2 Princípios Básicos de Redes Neurais

Como discutido em SMITH (1999), algumas décadas de estudo tem deixado claro ser extremamente difícil se construir sistemas artificiais capazes de exibir as características

de sistemas biológicos. As Redes Neurais são uma tentativa de se modelar estas características em um computador.

Existem diferenças essenciais entre o funcionamento do cérebro humano e a forma como os computadores seriais funcionam. Estas diferenças praticamente impossibilitam que estes últimos possam apresentar algum tipo de pensamento inteligente. Na realidade, o próprio conceito de "Inteligência" é de difícil definição.

2.2.2.1 Definição de Inteligência

De acordo com o NOVO DICIONÁRIO AURÉLIO (1999), "Inteligência" é a " Faculdade de aprender, apreender ou compreender; percepção, apreensão, intelecto, intelectualidade" ou ainda "Capacidade de resolver situações problemáticas novas mediante reestruturação dos dados perceptivos". Apesar da capacidade crescente de processamento seqüencial dos computadores atuais, ainda existe uma distância quase intransponível até que estes sejam capazes de realizar as tarefas que caracterizam inteligência.

Apesar da nossa limitada capacidade de processamento serial, nosso cérebro é capaz de processar diversas informações em paralelo, realizando em frações de segundo o que computadores seriais podem levar tempos além do aceitável para diversas aplicações. Esta capacidade do cérebro humano de realizar operações em paralelo nos permite desempenhar com relativa simplicidade tarefas como reconhecer padrões visuais, diferenciar sons, classificar fragrâncias ou tomar decisões com a seqüência de um determinado conjunto de atividades.

Ao se falar de inteligência, é necessário também se falar sobre "aprendizado", o qual pode ser definido como a capacidade de se extrair regras à partir de um conjunto de fatos, regras essas que serão utilizadas para decisões futuras.

As redes neurais são uma tentativa de se simular em computadores seriais o processamento paralelo que acontece no cérebro e, conseqüentemente, exibir outras características dos sistemas biológicos como o aprendizado.

2.2.2.2 O Cérebro Humano

A idéia de que o nosso cérebro é composto por unidades de processamento chamadas neurônios surgiu no início deste século. Estudos recentes mostram que os neurônios, enquanto unidades individuais de processamento, são algumas ordens de grandeza mais lentos do que as unidades de processamento encontradas hoje nos computadores.

A baixa velocidade de processamento dos nossos neurônios é compensada pela estrutura existente no cérebro. Estima-se que, em média, um ser humano tenha 10 bilhões de neurônios conectados entre si através de aproximadamente 60 trilhões de sinapses. Esta enorme integração possibilita uma grande eficiência em termos de volume e consumo de energia.

As conexões entre os neurônios, chamadas de sinapses, possibilitam que os neurônios se comuniquem através de sinais elétricos e químicos. Os neurônios são compostos por um corpo principal chamado "soma", por dentritos, o quais permitem que sinais cheguem até o soma e por um axônio, o qual transmite a outros neurônios o sinal de "saída" uma vez que o conjunto de sinais de "entrada" recebidos através dos dentritos causem a ativação do neurônio fazendo com que o mesmo "dispare".

Nos animais, o cérebro recebe sinais de entrada de diversos órgãos receptores (olhos, ouvidos, etc.) e envia sinais de saída para órgãos atuadores (músculos, glândulas, etc...).

2.2.2.3 Modelagem Biológica

Quando sinais de entrada são recebidos pelo cérebro, estes são avaliados e comparados com conhecimentos existentes e com as memórias armazenadas. Este processo permite que o aprendizado aconteça.

O aprendizado tem como resultado geral a mudança em um determinado comportamento o que, muitas vezes, pode ser percebido como uma variação na forma como o sistema reage a um determinado impulso. No cérebro humano, esta variação é causada por mudanças biológicas que ocorrem durante o aprendizado. Nesta fase do processamento das informações, as ligações existentes nas sinapses se tornam mais "fortes" ou "fracas" de forma a refletir a experiência adquirida, permitindo ao cérebro armazenar esta informação de maneira permanente.

Nas Redes Neurais Artificiais, princípios semelhantes são utilizados para permitir o aprendizado.

2.2.2.4 Modelo Artificial Simples

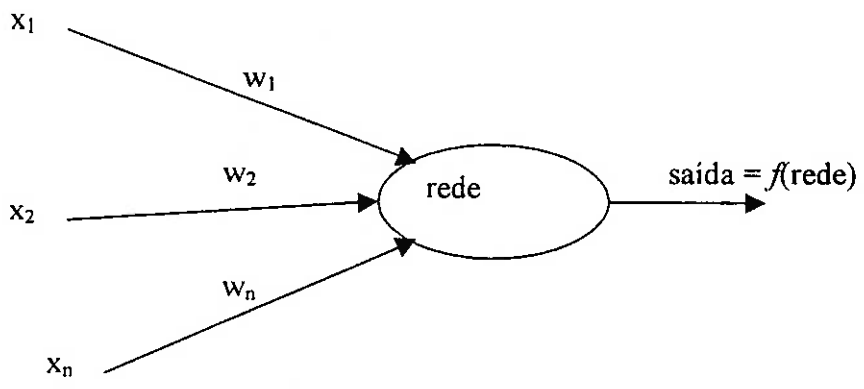
Algumas características básicas dos sistemas biológicos devem estar presentes nos modelos artificiais, conforme comentam EBERHART; DOBBINS (1990) e SMITH (1999). Primeiramente, é necessário que exista um conjunto de unidades simples conectadas entre si. Outra característica é que os neurônios recebam sinais daqueles

neurônios com os quais estão conectados e que a intensidade destes sinais seja proporcional a "força" da sinapse entre estes neurônios. A terceira característica básica é que se defina uma regra indicativa da intensidade necessária à soma dos sinais de entrada para que o neurônio "dispare".

A figura abaixo representa este modelo básico sem especificar as regras utilizadas. Cada neurônio recebe, através de seus "dentritos" (as ramificações receptoras) um sinal de entrada x_i de cada neurônio com o qual está conectado, sinais estes que são multiplicados pelo peso w_i de sua respectiva sinapse. O somatório de todas as entradas é obtido por:

$$rede = \sum_i x_i w_i$$

Este somatório é utilizado por uma determinada função *f*(rede) para determinar se o neurônio irá disparar e o "quanto" o neurônio irá disparar. Por disparar, entende-se o envio de um sinal de saída através da ramificação chamada "axônio" a qual, por sua vez, está conectada a dentritos de outros neurônios.



O objetivo da função de saída é simular o funcionamento dos neurônios. Em geral, são utilizados dois grupos básicos de funções. O primeiro tipo de função faz com que o

neurônio dispare assim que o somatório dos sinais de entrada alcança um determinado valor T (chamado "valor de limiar")(*), conforme a função abaixo:

$$saída = f(rede) = \begin{cases} 1 & se \quad rede \geq T \\ 0 & se \quad rede < T \end{cases}$$

O segundo tipo de função, ao invés de apresentar uma saída discreta, possibilita uma resposta contínua proporcional ao somatório dos sinais de entrada. Neste tipo de função, se o somatório dos sinais de entrada for bem maior do que T, será gerado um sinal de saída equivalente à 1. Já um somatório bem menor do que T irá gerar um sinal de saída equivalente à 0. No caso de um somatório equivalente a T, o sinal de saída será igual à 0,5. Desta forma, o que esta sendo representado é a média de disparos de um determinado neurônio ao longo do tempo. Um exemplo deste tipo de função é representado abaixo:

$$saída = f(rede) = \frac{1}{1 + e^{-\lambda(rede-T)}}$$

onde λ é um parâmetro que controla a forma do sinal de saída que será enviado para os neurônios conectados ao neurônio sendo avaliado.

2.2.2.5 Aprendizado

No cérebro humano, as sinapses tornam-se mais "fortes" ou "fracas" como forma de refletir o processo de aprendizado. No modelo artificial, este efeito é conseguido através da alteração dos multiplicadores w_n, comumente chamados de "pesos".

Um exemplo clássico deste tipo de aprendizado condicional foi demonstrado pelo psicólogo Ivan Pavlov, através de um experimento onde um cachorro foi treinado para

(*) Nomenclatura apresentada nas Notas de Aula do Prof. Jun Okamoto.

salivar ao ouvir o som de uma campainha. Os cães possuem um reflexo natural de salivar quando são confrontados com alimentos. Durante o período de treinamento, Pavlov soava uma campainha sempre que o cão era confrontado com o alimento. Após um período de tempo, o cão passou a salivar involuntariamente ao toque da campainha mesmo que sem a presença de alimento.

Traçando uma analogia ao modelo artificial descrito anteriormente, pode-se considerar o alimento e a campainha como dois sinais de entrada que contribuem para que um determinado neurônio "dispare" causando a salivação, a qual é o sinal de saída. Inicialmente, o peso atribuído ao alimento é razoavelmente alto (já que este estímulo provoca a salivação) e o peso atribuído a campainha é relativamente baixo (visto que, inicialmente, o som da mesma não provoca a salivação. A medida em que os dois estímulos são apresentados simultaneamente, o cérebro do cão começa a ter dificuldades em definir se o estímulo que provoca a salivação é o alimento ou a campainha. Analogamente no modelo artificial, isto significa que os pesos estão sendo modificados e tendendo para um valor intermediário. No final do período de treinamento, os pesos atingem valores capazes de fazer com que ambos os estímulos possam, individualmente, causar o reflexo da salivação.

2.2.2.6 Arquiteturas e Regras de Aprendizado

Além do modelo básico apresentado, existem diversas outras arquiteturas desenvolvidas para diferentes aplicações. Estas diversas arquiteturas podem ser divididas em dois grupos, conforme apresenta SMITH (1999a):

- "Feedforward"
- "Feedback"

Na redes do tipo "feedforward", o fluxo de informações entre os neurônios caminha em uma única direção. Já nas redes do tipo "feedback", existe uma retroalimentação. Desta maneira os sinais de saída são somados aos sinais de entrada.

A escolha da regra de aprendizado à ser utilizada depende até certo ponto da arquitetura escolhida para a rede. As regras de aprendizado podem ser divididas em dois grupos:

- Supervisionado
- Não-Supervisionado

Embora a literatura muitas vezes indique como sendo a diferença básica entre estes dois grupos o fato de que o aprendizado supervisionado requer treinamento ao contrário do não-supervisionado, uma análise mais detalhada do funcionamento das redes neurais demonstra que sempre existe um treinamento, fazendo com que os termos "aprendizado" e "treinamento" se tornem sinônimos. Desta maneira, a real diferença entre os dois grupos consiste em que no aprendizado supervisionado, é necessária a existência de um conjunto completo de valores de entrada e correspondentes saídas corretas para que a rede seja treinada. Em diversas situações este conjunto inicial não está disponível e a rede precisa ser capaz de aprender por si mesma, caracterizando o aprendizado não-supervisionado.

Durante o processo de aprendizado, quando um sinal de entrada é apresentado à rede, um sinal de saída o é gerado por cada neurônio. Em seguida, de acordo com a regra de aprendizado utilizada, um sinal de aprendizado r é gerado e utilizado para modificar os pesos. Em geral, os pesos são adaptados por uma regra do tipo:

$$w_i(t+1) = w_i(t) + cr(\mathbf{w}, \mathbf{x}, \mathbf{d})x_i(t)$$

onde c é uma constante que determina a razão de aprendizado e $\mathbf{d}$ é um vetor contendo as respostas conhecidas (para o caso do aprendizado supervisionado). O sinal de aprendizado é uma função do vetor de pesos $\mathbf{w}$, do sinal de entrada $\mathbf{x}$ e das saídas

desejadas (no caso de aprendizado supervisionado). A função *r* toma diversas formas de acordo com a arquitetura da rede e o tipo de aprendizado desejado. Um sumário pode ser vista em seguida, de acordo com SMITH (1999a):

Algumas Regras de Aprendizado Utilizadas em Redes Neurais				
Regra de Aprendizado	**Arquitetura**	**Tipo de Aprendizado**	**Sinal de Aprendizado** $r(\mathbf{w}, \mathbf{x}, d)$	**Comentário**
Hebb	"Feedforward"	Não-supervisionado	$= o$	
Perceptron	"Feedforward"	Supervisionado	$= d - o$	apenas para neurônios discretos
Delta	"Feedforward"	Supervisionado	$= (d - o)\frac{df}{d(net)}$	apenas para neurônios contínuos
Widrow-Hoff	"Feedforward"	Supervisionado	$= (d - o)net$	independe da função de ativação

2.2.3 Histórico das Redes Neurais

A literatura apresenta diversos históricos para o desenvolvimento das redes neurais como em EBERHART; DOBBINS (1990), FREEMAN et al. (1992), RITTER et al. (1992) e Rumelhart & McClelland apud SMITH (1999a). A divisão em cinco estágios básicos apresentada por SMITH (1999a) parece mais adequada pois abrange um período de tempo que se extende até o momento atual.

Em 1834, Charles Babbage definiu os princípios básicos de funcionamento das máquinas automáticas. Estes princípios vieram mais tarde a possibilitar a construção dos

computadores eletrônicos. No início do século 20, as máquinas automáticas começaram à ser largamente utilizadas para a realização de cálculos numéricos, especialmente quando a quantidade de operações tornava inviável a utilização de calculistas. Isto criou um mercado para este tipo de equipamento que ensejou a criação da IBM em 1914.

Simultaneamente ao desenvolvimento das máquinas utilizadas para cálculos, vários pesquisadores do campo da psicologia começaram a estudar o funcionamento do cérebro humano e os fundamentos do aprendizado.

Em 1890, William James publicou "Psychology" apud EBERHART; DOBBINS (1990), onde sistematizou pela primeira vez o estudo do cérebro. Em seguida, o trabalho de Ivan Pavlov, demonstrou a existência do aprendizado condicionado, o qual se tornou essencial no desenvolvimento das redes neurais artificiais. No intervalo entre as duas grandes guerras, Alan Turing lançou as bases dos sistemas de computação modernos.

Apesar dos estudos teóricos nos campos da computação e inteligência, apenas em 1943, McCulloch & Pitts publicaram um artigo definindo um modelo matemático básico para os sistemas neurais. Neste artigo entitulado "A Logical Calculus of the Ideas Immanent in Nervous Activity" apud EBERHART; DOBBINS (1990), os autores definiram uma estrutura simples composta por neurônios conectados através de ligações ponderadas por pesos. No entanto, a inexistência de recursos computacionais, impossibilitava uma experimentação das teorias recém desenvolvidas. A unidade de processamento definida por McCulloch & Pitts apud EBERHART; DOBBINS (1990), chamada de neurônio por analogia com o cérebro humano, estava baseada em cinco pressupostos:

- Os neurônios são ativados através de um processo do tipo "tudo ou nada". Ou seja, existem apenas dois estados possíveis: ativado e não-ativado;
- Um certo número de sinais de entrada deve ser mantido durante um intervalo de tempo definido para que o neurônio seja ativado;

- O único atraso existente na transmissão de sinais ocorre nas sinapses entre os neurônios;
- A existência de qualquer sinapse inibidora impede a ativação do neurônio enquanto o sinal inibidor estiver presente;
- A estrutura de rede não se altera ao longo do tempo.

Embora este modelo tenha lançado as bases para o estudo das redes neurais artificiais, alguns dos pressupostos listados (como a limitação de um estado binário) praticamente deixaram de ser empregados nos modelos usados atualmente.

A II Guerra Mundial trouxe notáveis avanços às "máquinas de somar" desenvolvidas até então, aumentando seu poder de processamento. Isto possibilitou que, em 1946, Wilkes desenvolvesse o primeiro "computador" capaz de armazenar um conjunto de instruções, o que viria mais tarde à ser chamado de "programa". Ao longo dos anos seguintes os computadores continuaram evoluindo em capacidade de armazenamento e velocidade de processamento. Um marco neste desenvolvimento foi a colocação em funcionamento, em 1954, do UNIVAC I, utilizado para automação da folha de pagamento da General Electric.

No campo da pesquisa, em 1949 Donald Hebb escreveu "The Organization of Behaviour" apud SMITH (1999a) onde propôs um modelo matemático capaz de possibilitar que os pesos de uma rede de neurônios fossem adaptados de forma à simular o processo de aprendizado condicional proposto por Pavlov. Finalmente em 1954, Marvin Minsky construiu o primeiro "Neuro-computador". À partir daí, o estudo das redes neurais começou à se configurar em uma área de estudo definida, passando a atrair diversos pesquisadores.

Em seu trabalho sobre o tema, ROSENBLATT (1957) apud EBERHART; DOBBINS (1990) definiu uma unidade de processamento chamada "Perceptron" e apresentou

alguns exemplos simples de como estes poderiam ser utilizados para demonstrar a faculdade de aprendizado das redes neurais. Este trabalho fez com que as redes neurais atraíssem mais ainda a atenção do meio acadêmico e despertassem a atenção do público leigo para o que poderia ser um grande passo na direção de máquinas capazes de tomar decisões como seres humanos. Um dos pontos altos da popularidade das redes neurais aconteceu em 1962 quando Bernard Widrow e Marcian Hoff demonstraram a capacidade das redes neurais de realizar diversas tarefas de impacto na vida cotidiana como previsão do tempo, jogos de azar e mercado de ações. Em WIDROW; HOFF (1962) apud EBERHART; DOBBINS (1990), foi introduzido um novo modelo simples de neurônio conhecido como "Adaline", o qual oferecia mais flexibilidade do que os modelos existentes até então.

O entusiasmo terminou em 1969 quando Minsky e Papert publicaram "Perceptrons" apud SMITH (1999a). Embora o objetivo inicial de Minsky e Papert fosse aprofundar o estudo dos Perceptrons, a sua pesquisa terminou por mostrar que estes eram incapazes de classificar corretamente dados que não fossem linearmente independentes. O prestígio destes dois autores na comunidade científica aliado ao trabalho detalhado, praticamente extinguiram a pesquisa em redes neurais. Como comentam EBERHART; DOBBINS (1990), o trabalho de Minsky e Papert foi baseado em sistemas compostos por Perceptrons simples com apenas uma camada de entrada e uma camada de saída (sem camadas ocultas), o que explica em parte os resultados limitados obtidos.

Os anos que se seguiram foram marcados por um grande avanço dos computadores eletrônicos, os quais foram rapidamente atingindo capacidades de armazenagem e velocidades além do imaginado nas décadas anteriores. Em vista da crescente capacidade de cálculo dos computadores, a qual passou a possibilitar a simulação dos

sistemas neurais de forma mais eficiente, alguns pesquisadores voltaram a estudar as redes neurais procurando formas de vencer os obstáculos apresentados em *Perceptrons*.

No final dos anos 70, Willshaw e von der Malsburg propuseram o conceito de auto-organização (do inglês "Self-Organizing")(*). Em 1982, Teuvo Kohonen publicou um estudo formal sobre os sistemas com características "self-organizing", KOHONEN (1982), o qual se tornou a referência básica para os estudo das redes "self-organizing".

O modelo apresentado por Kohonen foi responsável não só pelo desenvolvimento dos sistemas "Self-Organizing" como também pelo desenvolvimento de novas arquiteturas como a ART (do inglês "Adaptive Ressonance Theory"). Neste modelo proposto por GROSSBERG (1987), utiliza o princípio da "inibição lateral" para inibir os neurônios próximos a um neurônio ativado. Várias arquiteturas evoluíram deste modelo como ART1, ART2 e ART3.

Ainda em 1982, John J. Hopfield apresentou o seu modelo de rede neural num trabalho que ajudou a trazer de volta o interesse da comunidade científica para as redes neurais. Em HOPFIELD (1982), o objetivo inicial era a criação de um sistema capaz de localizar informações armazenadas em uma memória tendo apenas informações parciais sobre o conteúdo armazenado. Em trabalhos posteriores, Hopfield demonstrou a aplicabilidade do modelo a outros problemas como em HOPFIELD (1984) e HOPFIELD; TANK (1985).

Em trabalhos independentes, LECUN (1985) apud SMITH (1999a) e PARKER (1985) apud (SMITH 1999a), introduziram o conceito de "backpropagation" como forma de resolver as limitações apresentadas pelo modelo baseado em "Perceptrons". Isto deu um impulso definitivo ao estudo das redes neurais. À partir daí, o campo se firmou com a criação de diversos periódicos e conferências dedicados ao tema.

No decorrer da década de 90, o amadurecimento das redes neurais aliado à capacidade crescente dos computadores, fez com que as redes neurais encontrassem aplicação em diversas áreas, o que incentivou o surgimento de diversas empresas dedicadas ao desenvolvimento e comercialização de sistemas baseados nesta tecnologia. Simultaneamente, grandes corporações, juntamente com o meio acadêmico, passaram a investir de forma consistente no desenvolvimento de novas aplicações e no aprimoramento do conhecimento existente.

Atualmente, existem várias pesquisas em andamento buscando aliar as redes neurais à outras técnicas de inteligência artificial como sistemas especialistas, lógica nebulosa (do inglês "fuzzy systems") e algoritmos genéticos.

2.2.4 Revisão das Arquiteturas Básicas de Redes Neurais

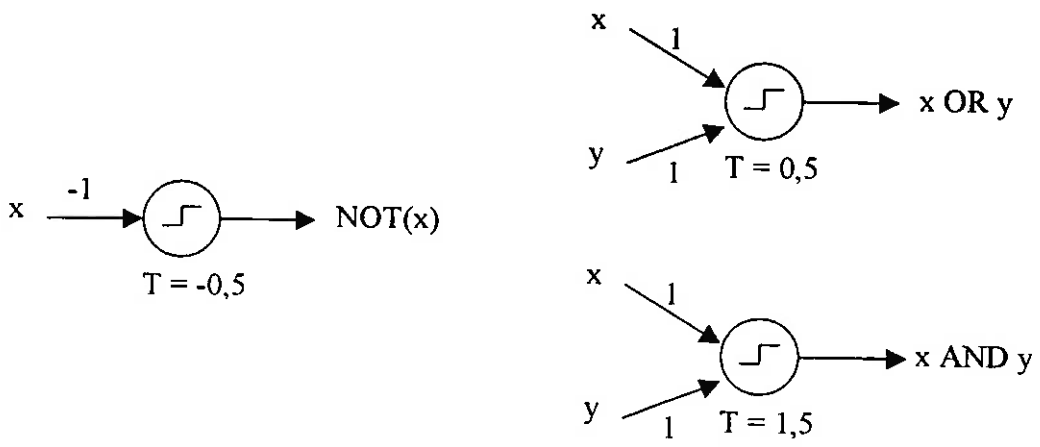
2.2.4.1 O Modelo de McCulloch-Pitts

Este é o modelo básico que iniciou todo o desenvolvimento das arquiteturas existentes para as redes neurais. Ele se baseia no conceito de uma unidade de processamento que, por analogia ao cérebro humano, é denominada de neurônio. Os sinais de entrada podem ser excitadores (com valor +1) ou inibidores (com valor −1). A função de ativação simples utilizada permite apenas os sinais de saída 0 ou 1, conforme a equação abaixo:

$$saida = f(rede) = \begin{cases} 1 & se \quad rede \geq T \\ 0 & se \quad rede < T \end{cases}$$

(*) Ao longo deste trabalho, será utilizado o termo em inglês "Self-Organizing"

Apesar da simplicidade deste modelo, o mesmo é capaz de realizar operações lógicas básicas como "AND", "OR" e "NOT". Estas operações podem ser realizadas através do uso de determinados pesos para os sinais de entrada *x*, *y* e valores de limiar *T*.



Operações mais complexas como NOR e NAND podem ser executadas com a utilização de neurônios conectados em série.

É importante notar que, neste caso, os pesos e valores de limiar são determinados *a priori*. Desta forma, nenhum tipo de aprendizado pode ser caracterizado.

2.2.4.2 O Modelo de Rosenblatt (Perceptron)

O Perceptron se baseia numa modificação do modelo de McCulloch-Pitts onde o limiar T é definido como 0 (zero). Para que isto seja possível, é necessário adicionar-se um sinal de entrada com valor −1 e peso equivalente ao limiar inicialmente desejado para a ativação do neurônio.

A diferença entre este modelo de McCulloch-Pitts modificado e o Perceptron é que este último possui pesos que podem assumir qualquer valor e não apenas 1 ou −1. Os

Perceptrons são capazes de aprender a resolver diversos problemas (como os de lógica booliana) através do ajuste dos pesos como apresentado por FREEMAN et al. (1992).

Uma das primeiras demonstrações da capacidade de aprendizado dos Perceptrons foi feita com a aplicação destes em problemas de classificação onde o objetivo é separar um conjunto de dados de entrada em diversas categorias. Um exemplo pode ser um sistema utilizado na concessão limites de crédito para clientes interessados em comprar bens produzidos por uma determinada empresa como descreve SMITH (1999a)

Este tipo de problema de classificação foi historicamente tratado através de técnicas estatísticas como regressão, as quais basicamente tentam ajustar um determinado modelo aos dados existentes. A vantagem dos Perceptrons na abordagem deste tipo de problema está no fato de que estas podem ser treinadas para aprender a detectar as características particulares de cada grupo existente através da análise de exemplos conhecidos.

Colocado de outra forma, os Perceptrons são capazes classificar dados através da criação de fronteiras de decisão entre a população de dados de entrada.

2.2.4.2.1 Algoritmos de Aprendizado para Perceptrons

Como em outras arquiteturas de redes neurais, os Perceptrons aprendem através de exemplos conhecidos; ou seja, através de um aprendizado supervisionado. Existem basicamente dois tipos de Perceptron, utilizados de acordo com o tipo de classificação que se deseja realizar:

- Classificação binária: onde existem apenas dois grupos de classificação possíveis. Neste caso apenas um neurônio é utilizado.

- Classificação múltipla: onde existem diversos grupos possíveis para a classificação dos dados. Neste caso diversos neurônios são utilizados.

Note-se que, em ambos os casos acima, o número de neurônios empregado é equivalente à quantidade de grupos nos quais se deseja classificar os dados de entrada. Além disso, cada um dos casos acima irá requerer uma regra de aprendizado específica. Para o caso de uma classificação binária, o algoritmo utilizado para o aprendizado pode ser descrito como:

- Passo 1: Inicializa-se todos os pesos com 0 ou um valor aleatório em torno de 0. Este vetor inicial de pesos é denominado $\mathbf{w}^0$.
- Passo 2: Apresenta-se um vetor de entrada (x_1, x_2, ...,x_n, -1) para a camada de entrada e calcula-se o valor de saída o da rede de acordo com:

$$o = f\left(\sum_{i=1}^{n+1} w_i x_i\right)$$

onde $f()$ é a função discreta de ativação.

- Passo 3: Os pesos são adaptados de acordo com:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + c(d-o)\mathbf{x}^t$$

onde c é a constante de aprendizado e d a saída desejada (conhecida) para um certo valor de entrada.

- Passo 4: Repete-se desde o Passo 2 até que todas as entradas tenham sido apresentadas e continua-se repetindo todo o processo até que os pesos não mais se alterem.

A saída desejada para uma determinada entrada é definida como tendo valor 1 caso a entrada pertença à classe A de dados e 0 caso a entrada pertença à classe B de dados.

Observando-se a equação utilizada no passo 3, dois fatos importantes devem ser notados:

- Os pesos são atualizados apenas no caso em que a rede "erra". Ou seja, caso o valor de saída *o* para uma determinada entrada seja semelhante ao valor esperado *d*, os pesos não se alteram;
- O aprendizado é proporcional à distância entre a saída obtida *o* e a saída esperada *d*. O valor utilizado para a constante *c* também influi diretamente no aprendizado.

Apesar da simplicidade da equação de ajuste de pesos descrita acima, ela é representativa de diversos casos práticos onde as características básicas são mantidas.

2.2.4.2.2 Perceptrons Multi-Categoria

Como foi dito anteriormente, o modelo básico de Perceptron se aplica a classificação de dados em dois conjuntos distintos. Esta classificação binária, embora possibilite um melhor entendimento do funcionamento deste modelo, é de reduzida aplicação em casos reais onde existem diversas categorias para classificação.

Num caso mais real, podemos assumir a existência de R categorias nas quais um conjunto de dados deve ser classificado. Para que a classificação seja possível, serão necessários R Perceptrons. Ao longo do aprendizado, cada neurônio será treinado para reconhecer uma determinada classe. Uma vez que todos os valores de entrada estarão conectados a todos os Perceptrons, teremos uma matriz de pesos W_{ij} ao invés de um vetor.

O treinamento será semelhante ao visto para o caso da classificação em duas classes:

- Passo 1: Inicializa-se todos os pesos com 0 ou um valor aleatório em torno de 0. Esta matriz inicial de pesos é denominada $\mathbf{W}^0$.

- Passo 2: Apresenta-se um vetor de entrada (x_1, x_2, ...,x_n, -1) para a camada de entrada e calcula-se o valor de saída *o* da rede de acordo com:

$$o_j = f\left(\sum_{i=1}^{n+1} w_{ji} x_i\right)$$

onde *f*() é a função discreta de ativação.

- Passo 3: Os pesos são adaptados de acordo com:

$$\mathbf{W}_j^{t+1} = \mathbf{W}_j^t + c(d_j - o_j)\mathbf{x}^t$$

onde *c* é a constante de aprendizado e d_j a saída desejada (conhecida) para um certo valor de entrada.

- Passo 4: Repete-se desde o Passo 2 até que todas as entradas tenham sido apresentadas e continua-se repetindo todo o processo até que os pesos não mais se alterem.

2.2.4.2.3 Limitações dos Perceptrons

Rumelhart & McClelland apud SMITH (1999a) apresentam um "Teorema de Convergência para os Perceptrons" segundo o qual: "...se uma classificação pode ser aprendida por um Perceptron, a mesma será aprendida em um número finito de iterações durante a fase de treinamento". Este teorema, cuja demonstração está além do objetivo deste trabalho, garante o treinamento num tempo finito mas não define as situações nas quais uma determinada classificação pode ou não ser aprendida.

Esta definição foi apresentada por Minsky & Papert apud EBERHART (1991), cujo estudo demonstra que os Perceptrons são limitados a classificar dados linearmente separáveis. Para demonstrar este resultado, eles utilizaram como exemplo a operação lógica XOR (OU-exclusivo), a qual retorna um valor verdadeiro se e somente uma das

variáveis de entrada for verdadeira e a outra for falsa. Ao se tentar utilizar os Perceptrons para resolver este tipo de operação, pode-se demonstrar que os pesos nunca convergem para valores definidos.

A força deste resultado criou um razoável ceticismo em relação ao uso das redes neurais como ferramenta efetiva na resolução de problemas práticos. Este ceticismo foi vencido aos poucos na medida em que outros trabalhos demonstraram que mudanças na arquitetura da rede e nas regras de aprendizado podem permitir que uma rede neural classifique dados mesmo que as classes desejadas sejam linearmente dependentes.

2.2.4.3 Redes Neurais de Várias Camadas

O modelo inicial de rede neural composto por Perceptrons era composto por apenas uma camada de unidades de processamento (neurônios).

Uma rede composta por uma combinação de Perceptrons arranjados em camadas é capaz de separar dados de entrada em classes linearmente dependentes como demonstrado por Rumelhart & McClelland apud SMITH (1999a). Na prática, este arranjo é obtido com a inclusão de uma camada "invisível" de Perceptrons entre o sinal de entrada e os neurônios de saída. Em seu trabalho, os autores demonstram o emprego de uma rede com esta arquitetura para a resolução do problema de simulação do operador lógico XOR.

2.2.4.3.1 Redes Neurais de Várias Camadas do Tipo "Feedforward"

As redes neurais do tipo "feedforward" compostas por várias camadas de Perceptrons são tratadas comumente pela sigla MFNN (do inglês "Multilayered feedforward neural

network"). No caso mais comum utilizado em aplicações práticas, esta rede é composta por uma camada de entrada, uma camada invisível de Perceptrons e outra camada de Perceptrons para a saída. Neste modelo, o número de posições de entrada é equivalente ao número de variáveis sendo utilizado para a classificação e o número de Perceptrons na camada de saída é equivalente ao número de classes em que se deseja classificar os dados.

A quantidade de Perceptrons na camada "invisível" ou "oculta" deve ser determinada de forma experimental para cada tipo de problema. Como regra básica, nota-se que um número excessivo de neurônios nesta camada faz com que a rede "decore" os padrões apresentados durante o treinamento e não seja capaz de generalizar para novos dados de entrada. Já um número insuficiente de neurônios faz com que a rede não seja capaz de identificar o relacionamento existente entre os valores de entrada e os valores conhecidos de saída utilizados durante o treinamento.

2.2.4.3.2 Regra de Aprendizado do Tipo "Backpropagation"

A regra de aprendizado inicialmente utilizada nas Redes Neurais para ajuste dos pesos é conhecida como Regra Delta e tem a forma:

$$w_i(t+1) = w_i(t) + cr(\mathbf{w}, \mathbf{x}, \mathbf{d})x_i(t)$$

onde c é uma constante que regula a taxa de aprendizado e $\mathbf{d}$ é um vetor com respostas corretas conhecidas. Nesta equação, o sinal de aprendizado é uma função do vetor de pesos $\mathbf{w}$, do vetor de entrada $\mathbf{x}$ e do vetor de saída $\mathbf{d}$. Este sinal é utilizado para modificar os pesos até que o sinal de saída o seja semelhante ao sinal correto.

Este sinal de aprendizado pode ser expresso como:

$$r = (d_k - o_k)\frac{df}{d(rede)}$$

Desta maneira, o sinal aprendizado é proporcional ao erro ou seja, à diferença entre a saída *o* obtida e a saída correta desejada *d*. Diferentemente da regra de aprendizado utilizada para o Perceptron, o sinal de aprendizado é multiplicado pela derivada da função de ativação.

Para o caso de uma MFNN, a Regra Delta pode ser escrita como:

$$v_{kj}(t+1) = v_{kj}(t) + c\lambda(d_k - o_k)o_k(1-o_k)y_j(t)$$

Neste caso, para se calcular o ajuste de pesos para a camada oculta de neurônios é necessário que se saiba a saída esperada para os neurônios desta camada oculta. Em geral, isto não é possível pois estes valores não são conhecidos. Como conseqüência, a Regra Delta não pode ser utilizada para ajuste dos pesos dos neurônios na camada oculta.

Para solucionar estes problemas, utiliza-se o método de "backpropagation", o qual permite que se calcule o erro dos neurônios de trás para frente, ou seja, partindo-se da camada de saída (cujos valores esperados são conhecidos) para a camada oculta.

O processo de "backpropagation" pode ser descrito de forma resumida através dos seguintes passos:

- Passo 1: Um vetor de entrada **x** com dimensão N é apresentado à camada de entrada, incluindo a entrada extra com valor -1;
- Passo 2: Calcula-se o somatório dos valores de entrada para a camada oculta de acordo com:

$$rede_j^h = \sum_{i=1}^{N+1} w_{ji} x_i$$

- Passo 3: Calcula-se os valores de saída para os neurônios da camada oculta de acordo com:

$$y_j = \frac{1}{1+e^{-\lambda(rede_j^h)}}$$

- Passo 4: Calcula-se o somatório dos valores de entrada para a camada de saída de acordo com:

$$rede_k^o = \sum_{j=1}^{J+1} v_{kj} y_j$$

- Passo 5: Calcula-se os valores de saída para os neurônios da camada oculta de acordo com:

$$o_k = \frac{1}{1+e^{-\lambda(rede_k^o)}}$$

- Passo 6: Calcula-se o sinal de aprendizado para os neurônios da camada de saída:

$$r_k^o = \lambda(d_k - o_k)o_k(1-o_k)$$

- Passo 7: Calcula-se o sinal de aprendizado para os neurônios da camada oculta:

$$r_j^h = \lambda(\sum_{k=1}^{K} r_k^o v_{kj})y_j(1-y_j)$$

- Passo 8: Atualiza-se os pesos dos neurônios da camada de saída:

$$v_{kj}(t+1) = v_{kj}(t) + cr_k^o y_j(t) = v_{kj}(t) + c\lambda(d_k - o_k)o_k(1-o_k)y_j(t)$$

- Passo 9: Atualiza-se os pesos dos neurônios da camada oculta:

$$w_{ji}(t+1) = w_{ji}(t) + cr_j^h x_i(t) = w_{ji}(t) + c\lambda(\sum_{k=1}^{K} r_k^o v_{kj})y_j(1-y_j)x_i(t)$$

- Passo 10: Atualiza-se o erro para esta época:

$$E \leftarrow E + \sum_{k=1}^{K} (r_k^o)^2$$

- Passo 11: Repete-se desde o passo 1 para o próximo vetor de entrada. Ao final de cada época, o erro E deve ser reiniciado com o valor 0. O algoritmo é repetido até que o erro se torne menor do que um valor bem pequeno previamente estabelecido.

O algoritmo descrito nos passos acima busca, em última análise, minimizar a função erro E. De forma semelhante à outros métodos matemáticos que buscam minimizar uma determinada função, ele garante que o erro seja diminuído à cada iteração, ou seja, a convergência é garantida. Em contrapartida, não necessariamente o mínimo atingido será o mínimo global, existindo uma dependência entre o local de início e o mínimo alcançado.

2.2.4.3.3 Treinamento e Implementação

Como foi demonstrado ao longo dos itens anteriores, as redes neurais são capazes de extrair relações existentes em conjuntos de dados, possibilitando a resolução de diversos problemas. Porém, para que isto seja possível, o treinamento deve acontecer de forma que a rede consiga generalizar as relações aprendidas e não apenas "memorize" os padrões apresentados durante o treinamento. Em geral, desde que exista um número suficiente de neurônios na camada oculta, mesmo relações extremamente complexas podem ser extraídas. Além desta quantidade, outros parâmetros são importantes para que a rede alcance os resultados desejados. Abaixo está um resumo destes parâmetros:

a) Quantidade de neurônios na camada oculta:

A quantidade de neurônios na camada oculta está diretamente relacionada à capacidade da rede de classificar os dados de entrada. Uma quantidade excessiva de neurônios nesta camada faz com que a rede perca sua capacidade de generalização e responda corretamente apenas aos padrões apresentados durante a fase de treinamento. Já no caso

de a quantidade ser menor do que o devido, a rede não será capaz de extrair todas as relações existentes entre os dados e o erro dificilmente será reduzido ao nível esperado.

A literatura, como visto em SMITH (1999a) apresenta diferentes formas de se estimar a quantidade J de neurônios na camada oculta como por exemplo:

- $J = \sqrt{NxK}$, onde N é a dimensão dos padrões de entrada e K a quantidade de neurônios na camada de saída;
- $J = \frac{1}{2}(N+K) + \sqrt{P}$, onde P é a quantidade de padrões a serem apresentados durante a fase de treinamento.

Caso se decida por tentar encontrar a quantidade ideal de neurônios por tentativa e erro, a experiência mostra ser mais vantajoso começar com uma quantidade inferior e aumentar gradativamente.

b) Valor inicial para os pesos:

Novamente aqui a literatura propõe diversas abordagens para a determinação deste valores iniciais como em EBERHART; DOBBINS (1990) e WELSTEAD (1994). O senso comum parece ser de que 0 não é uma boa escolha e que, o ideal é escolher pequenos valores aleatórios em torno de 0 ou outra referência.

c) Taxa de aprendizado:

Este parâmetro controla o quanto os pesos são ajustados durante cada iteração do algoritmo. Caso este parâmetro seja grande demais, muito provavelmente acontecerá uma oscilação em torno de um mínimo local sem no entanto convergir. Já no caso de um valor muito pequeno, o número de iterações necessárias para a convergência pode se tornar excessivamente alto, fazendo com que o processo se torne muito demorado. Em geral, casos práticos citados na literatura como em SMITH (1999a) utilizam taxas de aprendizado variando entre 10^{-4} e 10^{1}.

d) Função de ativação:

Embora os modelos inicialmente propostos para redes neurais utilizassem funções de ativação não contínuas, as arquiteturas que se seguiram passaram a utilizar funções contínuas do tipo $f(rede) = \frac{1}{1+e^{-\lambda(rede)}}$ onde se deve escolher o parâmetro λ, o qual controla a taxa de crescimento da função. Desta forma, este parâmetro λ também irá influenciar o aprendizado de cada neurônio uma vez que o sinal de aprendizado é diretamente proporcional à derivada da função de ativação. Como conseqüência, quanto maior o valor de λ, maior será a derivada e maior será a correção (quantidade de aprendizado) à cada iteração. Outra conseqüência é que aqueles pesos que ainda não se aproximaram de seu valor definitivo sofrem um aprendizado maior do que aqueles que já se estabilizaram. Existem diversas possibilidades para as funções de ativação para as redes do tipo "backpropagation", bastando que estas funções sejam crescentes, monótonas e diferenciáveis.

e) Escolha do conjunto de treinamento:

Quando a escolha dos parâmetros e o treinamento são bem sucedidos, a rede deve apresentar, com um novo conjunto de dados de entrada, a mesma performance obtida com o conjunto de treinamento.

Para se ter certeza que o treinamento atingiu os objetivos desejados, é preciso reservar parte dos dados para que sejam feitos testes após o treinamento. Em geral, este conjunto de teste contém entre 10% e 20% da quantidade total de padrões existentes e é selecionado de forma aleatória.

Caso o erro medido tenha sido bastante pequeno durante o treinamento e se tornou grande quando da apresentação dos dados de teste, a conclusão é que a rede não foi capaz de aprender a generalizar o o conhecimento adquirido durante o treinamento;

tendo apenas memorizado os padrões de entrada. Entre as razões para este insucesso podem estar:

- Escolha ruim dos parâmetros
- Quantidade excessiva de neurônios na camada oculta;
- O conjunto de teste não representa corretamente as relações aprendidas durante o treinamento.

No tocante a este último item, é importante tomar alguns cuidados na seleção do conjunto de testes para garantir que padrões pouco comuns existentes no universo de dados estejam presentes tanto no conjunto de treinamento quanto no conjunto de testes.

2.2.5 Redes Neurais "Self-Organizing"

Diversas arquiteturas foram propostas na tentativa de se aproximar da melhor forma possível o funcionamento do cérebro humano ao mesmo tempo em que se mantém a aplicabilidade a problemas práticos e a possibilidade de implementação em circuitos eletrônicos (com a conseqüente simulação em sistemas digitais).

Dentre estas arquiteturas, encontram-se os "Self Organizing Feature Maps" propostos por Teuvo Kohonen no início da década de 80.

A arquitetura proposta por Kohonen se baseia numa característica dos sistemas neurais biológicos que é a existências de regiões do córtex cerebral com respostas seletivas a determinados impulsos. Desta forma, certos impulsos são mapeados por conjuntos específicos de neurônios, criando assim uma "imagem" do estímulo no córtex cerebral.

Embora exista muita discussão no campo da neurofisiologia quanto a validade deste modelo, sendo que uma discussão detalhada pode ser encontrada em DAMASIO (1994).

Os algoritmos derivados do modelo proposto por Kohonen são baseados em evoluções dos métodos de aprendizado não-supervisionado. Neste métodos, padrões de entrada são apresentados e agrupados em determinadas regiões em razão de sua "similaridade", a qual normalmente é avaliada como sendo a "distância" entre estes padrões no espaço multidimensional em que se encontram. Ao longo do processo, a rede neural cria fronteiras de decisão entre os diversos grupos de padrões identificados.

A rede "Self-Organizing" constrói um mapeamento a partir de um conjunto ordenado topologicamente em um conjunto topológico contínuo ou discreto que preserva as propriedades do conjunto inicial como discutido em BOSE; LIANG (1996), FREEMAN et al. (1992), LOOI (1992) e REINHARDT; MULLER (1991). A análise matemática destas redes é complexa e requer conceitos probabilísticos além do escopo deste trabalho, podendo ser encontrada em KOHONEN (1997).

2.2.5.1 Princípios dos Sistemas "Self-Organizing"

A Auto-Organização é um exemplo de processo on ocorre um aprendizado não-supervisionado. Desta forma, padrões de entrada são apresentados para uma rede neural "Self-Organizing" (também chamada de SONN, do inglês "Self-Organizing Neural Network") cuja saída agrupa estes padrões de entrada de acordo com critérios de similaridade extraídos dos próprios dados.

Para que as SONN possam ser analisadas, é preciso definir o conceito de similaridade o qual, em geral, é a distância entre dois padrões de entrada com dimensão N medida no espaço N-dimensional.

A rede neural do tipo SONN adapta seus pesos durante o treinamento ao mesmo tempo em que define as classes existentes dentro do conjunto de dados de entrada. Ao longo da

últimas décadas, diversas arquiteturas e regras de aprendizado foram propostas para este fim.

2.2.5.2 Redes Neurais Não-Supervisionadas

O processo de se encontrar subconjunto com características semelhantes dentro de conjuntos maiores é comum no estudo dos métodos estatísticos conforma textos básicos como PARZEN (1960). Assim como no caso das redes neurais, os métodos estatísticos utilizados para classificar dados também podem ser baseados em processos onde existe um conjunto de dados para treinamento, como no caso da regressão e processos onde não existe um conjunto de dados disponível para treinamento como no caso do agrupamento.

O agrupamento de dados possibilita o alcance de três objetivos principais quando aplicado à um determinado conjunto de dados:

- Pré-processamento dos dados, removendo padrões que possam representar erros na fase de aquisição dos dados;
- Permite que características específicas de determinados subconjuntos se tornem evidentes, possibilitando uma visão mais clara dos universo de dados;
- Permite que as classes identificadas sejam utilizadas para classificação de novos conjuntos de dados no futuro.

Para se criar um agrupamento, é preciso inicialmente se definir a forma como a similaridade entre os dados será medida.

Utilizando-se a fórmula para cálculo da distância Euclidiana entre dois pontos:

$$\|x-y\| = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2}$$

pode-se calcular a distância entre dois padrões de entrada **x** e **y** com dimensão N:

$$\|x-y\| = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$$

Nas redes do tipo MFNN, durante a fase de treinamento, os pesos das ligações entre os neurônios são atualizados de forma a reduzir o erro entre a saída obtida e as saídas conhecidas. Já nas redes não-supervisionadas, apenas aqueles neurônios declarados "vencedores" sofrem atualização.

Uma das arquiteturas básicas para redes não-supervisionadas é aquela onde o "vencedor-leva-tudo" (do inglês "Winner-Takes-All). Como o nome indica, à cada iteração, apenas um neurônio terá seus pesos atualizados. Este tipo de rede é composto por apenas uma camada de neurônios conectados aos sinais de entrada. Neste caso, a quantidade de sinais de entrada é equivalente à dimensão dos padrões de entrada e a quantidade de neurônios é equivalente ao número de classes nas quais se deseja classificar os padrões de entrada.

Como nas redes MFNN, quando um padrão de entrada é apresentado, os valores de saída são calculados para cada neurônio através de uma função de ativação contínua. O neurônio com o maior sinal de saída é chamado de vencedor e passa a ser chamado de neurônio m. Os pesos das ligações entre os sinais de entrada e o neurônio m são atualizados de acordo com:

$$w_{mi}(t+1) = w_{mi}(t) + c(x_i - w_{mi})$$

onde c é a taxa de aprendizado. À medida que o processo avança, os pesos se movem na direção dos padrões de entrada com maior grau de similaridade.

Uma das limitações desta arquitetura onde apenas o neurônio vencedor é beneficiado é o fato de que, por possuir apenas uma camada de neurônios, não é capaz de classificar corretamente dados que não sejam linearmente separáveis, de forma similar ao que

acontece com as redes do tipo MFNN. Para solucionar este problema, foram propostas estruturas alternativas como as redes neurais do tipo Contra-Propagação ou de forma abreviada, CPM (do inglês "Counter-Propagation Network") ou como os "Self-Organized Feature Maps" (SOFM) propostos por KOHONEN (1982), os quais serão utilizados mais adiante para o problema de seqüenciamento.

2.2.5.3 "Self-Organizing Feature Maps"

Nos problemas de classificação geralmente encontrados, existem diversas variáveis à serem levadas em conta, o que torna difícil uma visualização das diversas classes nas quais os dados se encaixam. Um dos objetivos dos SOFM é criar um mapeamento destes conjuntos de diversas variáveis em um espaço de características com menos variáveis, facilitando o entendimento das classificação obtida.

2.2.5.3.1 Arquitetura das Redes do Tipo SOFM

Em geral, as redes do tipo SOFM são compostas por um vetor de entrada **x**, com dimensão n, conectado com um vetor ou uma matriz de neurônios.

Quando um determinado padrão de entrada é apresentado, determinadas regiões da camada de saída são ativadas e, em seguida, os pesos que conectam os elementos de entrada aos neurônios dentro destas regiões são atualizados. À medida que o processo evolui, determinadas regiões da camada de saída passam a responder à padrões de entrada específicos.

2.2.5.3.2 Comparação Com os Sistemas Biológicos

Diversos estudos tem demonstrado o fato de que o cérebro humano desenvolve regiões especializadas em determinadas atividades como recebimento de sinais advindos dos órgãos sensitivos, regulagem de funções corporais através da liberação de hormônios, ações físicas através dos músculos, etc... Isto torna o cérebro extremamente tolerante a falhas já que, numa determinada região, um neurônio pode ser substituído por seus vizinhos em caso de falha.

Outra semelhança entre o cérebro humano e as redes neurais do tipo SOFM está no fato que ambos os sistemas realizam a extração de características presentes em sinais de entrada. Um exemplo é a capacidade do cérebro de identificar o intérprete de uma determinada música mesmo que o ouvinte nunca tenha ouvido esta determinada canção, bastando para isso que o cérebro tenha aprendido (através de outras canções do mesmo intérprete), características específicas do intérprete em questão como timbre de voz, uso de determinadas associações, etc...

Assim como nas redes do tipo SOFM, os neurônios em nosso cérebro também desenvolvem uma resposta seletiva à determinados impulsos, fazendo com que apenas um determinado grupo de neurônios responda à um determinado estímulo. Este conjunto de similaridades, aliado ao interesse renovado nas redes neurais trazido em parte pelo avanço nos computadores digitais, fez com que as redes do tipo SOFM ganhassem popularidade e passassem à ser estudadas com mais profundidade.

2.2.5.3.3 O Conceito de "Vizinhança"

Assim como no significado mais comum desta palavra, em redes neurais do tipo SOFM o termo vizinhança é aplicado aqueles neurônios que sofrem efeitos sensíveis quando um dos neurônios da vizinhança é declarado "vencedor". Assim como em outras situações, o "quanto" os neurônios da vizinhança são afetados (tem seus pesos ajustados) depende da distância existente entre um determinado neurônio da vizinhança e o neurônio vencedor. Em geral, a literatura utiliza a letra N para indicar o tamanho da vizinhança ao redor do neurônio vencedor, indicado pela letra m.

Para o caso de uma rede com uma camada de saída unidimensional (apenas um vetor), a vizinhança é definida como um certo número de neurônios à esquerda e à direita do neurônio vencedor. Para uma vizinhança de tamanho unitário ($N = 1$), apenas os dois neurônios imediatamente adjacentes ao neurônio vencedor fazem parte da vizinhança.

Já no caso de uma rede com uma camada de saída bidimensional (uma matriz), a vizinhança é definida como uma certa área ao redor do neurônio vencedor. Esta área pode assumir diversos formatos, sendo o hexagonal um dos mais comumente utilizados. Neste caso, uma vizinhança de tamanho unitário ($N = 1$),compreende os seis neurônios adjacentes ao neurônio vencedor.

Nas redes neurais do tipo SOFM é comum a dimensão da vizinhança ser modificada ao longo do processo de aprendizado; começando ampla e sendo reduzida até que apenas o neurônio vencedor seja atualizado. Este artifício permite que a rede responda de forma mais livre no início do aprendizado e se torne menos flexível à medida que o processo evolua. O aprendizado, por sua vez, também é proporcional à distância entre o neurônio sendo avaliado (dentro da vizinhança) e o neurônio vencedor m. Em geral, a quantidade

de aprendizado c que um determinado neurônio dentro da vizinhança recebe é expresso por:

$$c = \alpha(t)e^{(-|r_i - r_m|/\sigma^2(t))}$$

onde $r_i - r_m$ mede a a distância entre o neurônio i sendo avaliado e o neurônio vencedor m. As funções $\alpha(t)$ e $\sigma^2(t)$ servem para controlar a quantidade de aprendizado recebida por cada neurônio. Dependendo da implementação, diferentes funções são usadas. Normalmente, estas funções são decrescentes ao longo do tempo mas também é comum encontrar casos em que uma delas ou ambas são constantes ao longo do tempo.

Na maioria dos casos, os neurônios fora da vizinhança não recebem nenhum aprendizado. Em algumas situações porém, pode ser conveniente que os neurônios fora da vizinhança tenham seus pesos atualizados mesmo que numa espécie de "aprendizado negativo" onde o valor dos pesos é reduzido ao invés de aumentado.

2.2.5.3.4 O Funcionamento das Redes Neurais do Tipo SOFM

De um ponto de vista que leve em conta o algoritmo utilizado nas redes do tipo SOFM, existe uma grande semelhança com os algoritmo de outros tipos de rede como as MFNN. A principal diferença está na forma como os pesos são atualizados. Um algoritmo básico possui os seguintes passos:

- Passo 1: Inicialização
    - Os pesos são inicializados com valore aleatórios (em geral entre 0 e 1);

- A vizinhança é inicializada com valores grandes. Um certo cuidado deve ser tomado para que a vizinhança não "caia fora" dos limites da rede;
- As funções $\alpha(t)$ e $\sigma^2(t)$ são inicializadas com valores entre 0 e 1 (caso uma delas ou ambas não sejam definidas como constantes);

➢ Passo 2: Um padrão de entrada **x** é apresentado na camada de entrada e a distância d deste padrão em relação à cada um dos pesos j é calculada de acordo com:

$$d_j = \|\mathbf{x} - \mathbf{w}_j\| = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$$

➢ Passo 3: O neurônio com a menor distância é designado vencedor m;

➢ Passo 4: Os pesos do neurônio vencedor e dos neurônios em sua vizinhança são atualizados de acordo com:

$$w_{ji}(t+1) = w_{ji}(t) + c\left[x_i - w_{ji}(t)\right]$$

onde $c = \alpha(t)e^{(-\|r_i - r_m\|/\sigma^2(t))}$ para todos os neurônios j na vizinhança $N_m(t)$;

➢ Passo 5: Repete-se o processo desde o passo 2 por Ω épocas e então reduzir:

- O tamanho da vizinhança;
- Os valores de $\alpha(t)$ e $\sigma^2(t)$;

Ao final deste processo, o algoritmo termina quando os pesos se estabilizam, ou seja, passam a variar menos do que um determinado valor ε bem pequeno.

Algumas propriedades importantes das redes neurais do tipo SOFM são importantes não só do ponto de vista da similaridade para com o cérebro humano mas também para justificar a sua aplicabilidade na resolução de problemas de natureza combinatória como o seqüenciamento. Algumas destas propriedades são descritas abaixo:

- Competição global e cooperação local. Ou seja, ao mesmo tempo em regiões distintas competem na seleção do neurônio vencedor, existe uma cooperação na medida em que os neurônios da vizinhança do neurônio vencedor se beneficiam ao ter seus pesos ajustados;
- Aproximam de forma bastante eficaz os espaço dos padrões de entrada através da extração de características principais;
- Criam um mapeamento topologicamente ordenado dos dados de entrada na medida em que os neurônios de uma determinada região passam à responder à determinadas características existentes nos padrões de entrada;
- O mapeamento criado guarda um relacionamento com a distribuição estatística dos padrões de entrada na medida em que padrões de entrada mais comuns causam a ativação de uma região mais extensa do que padrões menos freqüentes.

2.2.6 Aplicação das Redes Neurais à Problemas Combinatórios

Existem duas abordagens básicas para a resolução de problemas combinatórios através de redes neurais. A abordagem mais usual consiste em se utilizar a rede para minimizar uma determinada função custo que represente o problema em questão. Outro método é desenvolver uma rede onde os neurônios podem competir entre si para que sejam ativados em determinadas situações.

Nos problemas de otimização combinatória geralmente se busca o mínimo global de uma determinada função. Esta busca se torna difícil computacionalmente não só por se tratar de um problema Np-completo como também porque estes problemas geralmente possuem diversos mínimos locais como apresentam BURKE; IGNIZIO (1992) e VAITHYANATHAN; IGNIZIO (1992).

O seqüenciamento, enquanto problema combinatório, tem sido objeto de diversos trabalhos como TZAFESTAS (1994). Dependendo do tipo de problema, principalmente do objetivo que se deseja atingir, o seqüenciamento de ordens de produção se caracteriza como um problema Np-completo. Uma análise dos trabalhos de BAKER (1997), CONWAY (1967) e MORTON (1993) mostra que, em geral, problemas com objetivos vinculados aos "pesos" (ou seja, as prioridades) das tarefas a serem seqüenciadas tendem à ser Np-completos.

2.3 Problema do "Caixeiro Viajante"

Um dos problemas utilizado com "benchmark" para a avaliação de algoritmos que se propõe a solucionar problemas combinatórios é o Problema do Caixeiro Viajante ilustrado em BOSE; LIANG (1996), BURKE; IGNIZIO (1992) e SMITH (1996).

O objetivo é se encontrar o caminho mais curto entre um conjunto de N cidades. Através de uma busca exaustiva, seria necessário verificar-se todos os $\frac{1}{2}(N-1)!$ caminhos possíveis, o que seria inviável mesmo para pequenos valores de N.

Diversos métodos foram tentados para se buscar aproximações convenientes para este problema, sendo um deles o proposto por FAVATA; WALKER (1991).

Ao longo do desenvolvimento das redes neurais, diversas tentativas foram feitas para se adaptar os modelos existentes para a solução deste problema. Uma das primeiras tentativas foi através da utilização do modelo de Hopfield e Tank como pode ser visto em BURKE; IGNIZIO (1992) e HOPFIELD; TANK (1985). Diversos autores se manifestaram contra este método alegando que a dificuldade em se conseguir repetir os resultados obtidos torna difícil levá-los a crédito como argumenta SHIRAZI (1998).

A forte característica espacial do problema do caixeiro viajante atraiu o interesse de diversos pesquisadores para a aplicação da arquitetura proposta por Kohonen na resolução deste problema.

Estas abordagens se baseiam de uma maneira ou de outra no modelo da rede elástica onde as cidades a serem percorridas representam nós de uma malha que se deforma buscando atingir todas as cidades.

Uma destas abordagens é proposta por FORT (1988) onde um conjunto circular unidimensional de neurônios é sobreposto as cidades a serem percorridas de maneira que dois neurônios adjacentes representem duas cidades vizinhas em distância.

Outra abordagem proposta em seguida por FAVATA (1991) utiliza um método semelhante e consegue resultados satisfatórios em curtos intervalos de tempo.

CAPÍTULO 3 - SEQUENCIAMENTO ATRAVÉS DE REDES NEURAIS "SELF-ORGANIZING"

3.1. Proposta de uma Rede Neural "Self-Organizing" Modificada

Como explicado acima, os modelos existentes para a aplicação das redes "self-organizing" utilizam algumas adaptações a proposta inicial de Kohonen mas se baseiam na forte analogia entre a arquitetura das redes "self-organizing" e a distribuição espacial das cidades a serem visitadas no espaço bidimensional.

Esta dependência, conveniente para o problema do caixeiro viajante, torna bastante difícil a extensão destes modelos para aplicação em outros problemas combinatórios como o seqüenciamento como argumenta SMITH (1996).

Neste trabalho, será explorada uma nova proposta de rede "Self-Organizing" para a resolução de problemas combinatórios. Esta proposta, desenvolvida por GUERRERO at al. (1998 e 1999) e LOZANO et al. (2000) foi baseada nos trabalhos de SMITH (1996) e SMITH et al. (1998).

O modelo inicialmente proposto por SMITH (1996) é composto por dois estágios. No primeiro estágio, uma variação da rede "self-organizing" busca otimizar a função objetivo. Embora os conceitos básicos propostos por KOHONEN (1997) sejam mantidos, tanto o nó vencedor como a vizinhança são definidos em termos da função objetivo ao invés da proposta original onde estes eram definidos em relação à distribuição espacial dos nós. Os pesos são então adaptados levando-se em conta a "distância" existente entre os membros da vizinhança.

No segundo estágio, uma rede neural utilizando a arquitetura proposta por Hopfield e Tank é empregada para restringir a convergência da rede ao espaço de soluções

permitido para o problema. Para uma revisão mais detalhada do modelo de Hopfield e Tank, deve-se buscar HOPFIELD (1982 e 1984) e HOPFIELD; TANK (1985).

Esta arquitetura híbrida foi modificada por GUERRERO et al. (1998 e 1999), utilizando-se apenas a rede "self-organizing".

3.1.1 Definição do Problema de Sequenciamento

- Índices: j,l representam ordens
  s representam posições na seqüência
- Dimensões: M representa a quantidade de posições na seqüência
  N representa a quantidade de ordens a serem seqüenciadas
  Como todas as ordens serão seqüenciadas, tem-se M = N
- Parâmetros: S_{jl} representa o efeito causado na função objetivo sendo avaliada quando as ordens j e l são tem suas posições invertidas na seqüência.
- Variável de Decisão:

$$X_j = \begin{cases} 0 & \textit{se a ordem } j \textit{ está atrasada na sequência sendo avaliada} \\ 1 & \textit{se a ordem } j \textit{ não está atrasada na sequência sendo avaliada} \end{cases}$$

O problema de seqüenciamento em uma única máquina pode ser modelado como:

$$\text{maximizar} \qquad F(X) = \sum_{k=1}^{M} \sum_{j=1}^{N} S_{jl} X_j \tag{1}$$

3.1.2 Arquitetura da Rede "Self-Organizing" Modificada

Nesta seção será apresentada a rede neural proposta inicialmente por SMITH (1996) e SMITH et al. (1996) e modificada posteriormente por GUERRERO et al. (1998 e 1999).

Esta rede "self-organizing" é composta por uma camada de entrada contendo "N" nós e uma camada de saída contendo "M" nós. Ambas as camadas são totalmente conectadas por pesos, sendo W_{ks} o peso que conecta o nó de entrada *k* ao nó de saída *s*. A camada de entrada recebe vetores com uma determinada seqüência de ordens. A camada de saída contém as possíveis posições na seqüência. Ao longo do processo, os pesos variam para indicar onde uma determinada ordem deve se posicionar na seqüência de maneira a maximizar o critério de desempenho desejado.

Ao contrário das propostas de aplicação de redes neurais "self-organizing" adotadas em outros trabalhos como FAVATA; WALKER (1991) e FORT (1998) os quais são baseados na rede elástica, aqui os nós são mantidos fixos ao longo do processo e os pesos são modificados à medida em que nós da camada de saída competem para minimizar o custo associado a um determinado padrão apresentado na camada de entrada.

Ao se apresentar o padrão de entrada da ordem *k**, calcula-se o potencial V_{sk*} para cada um dos nós de saída *s* utilizando-se:

$$V_k = \sum_{j=1}^{N} W_{kj} w_j X_j \qquad (2)$$

onde W_{ks} representa o peso da ligação entre a ordem k e a posição s, w_k o peso da ordem k na avaliação do seqüenciamento e X_k indica se a ordem k está ou não atrasada numa determinada seqüência. Desta forma:

$$X_j = \begin{cases} 0 & \textit{se a ordem } j \textit{ está atrasada na sequência sendo avaliada} \\ 1 & \textit{se a ordem } j \textit{ não está atrasada na sequência sendo avaliada} \end{cases}$$

Após o calculo de todos os potenciais, localiza-se o nó vencedor (m_0), o qual representa o maior potencial V:

$$V_{m_0} \geq V_k \quad \forall k \tag{3}$$

Em seguida, é determinada a vizinhança do nó vencedor ordenando-se os potenciais em ordem decrescente:

$$V_{m_0} \geq V_{m_1} \geq V_{m_2} \geq \ldots \geq V_{m_\eta} \tag{4}$$

A vizinhança é composta por η nós. Novamente aqui evidencia-se a diferença entra esta abordagem, onde a vizinhança é determinada em razão dos potenciais e a abordagem clássica, onde a vizinhança é determinada por critérios espaciais como em FAVATA; WALKER (1991), FORT (1988) e KOHONEN (1982).

Após a definição da vizinhança, os pesos conectando o nó k^* aos diversos nós s são atualizados através de uma variação sobre a regra originalmente proposta por KOHONEN (1982):

$$\Delta W_{ks} = \begin{cases} \alpha(\eta,t)\left[1 - W_{ks}\right] & \forall k \textit{ dentro da vizinhança} \\ -\beta(t) W_{ks} & \forall s \textit{ fora da vizinhança} \end{cases} \tag{5}$$

$$\text{onde: } \alpha(\eta,t) = \beta(t)\exp\left[-\frac{\left[V_{m_0} - V_k\right]}{\left[V_{m_0} - V_{m_{N-1}}\right]}\right] \text{ e } t \text{ indica a iteração corrente} \tag{6}$$

β é um valor positivo que diminui a medida que o processo avança. Isto é feito multiplicando-se β por um valor fixo σ através da seguinte equação:

$$\beta(t) = \beta(t-1) * \sigma \tag{7}$$

Os novos pesos são obtidos através de:

$$W_{ks} \leftarrow W_{ks} + \Delta W_{ks} \tag{8}$$

Ao final do processo, os pesos W_{ks} são normalizados.

3.1.2.1 Algoritmo

- Passo 0: Os pesos são inicializados com valores aleatórios e em seguida normalizados de maneira que:

$$0 \leq W_{ks} \leq 1 \quad \forall k, s \tag{9}$$

$$\sum_{s=1}^{N} W_{ks} = 1 \quad \forall k \tag{10}$$

- Passo 1: As ordens são colocadas em seqüência segundo a regra EDD;
- Passo 2: É feita uma permutação entre duas ordens vizinhas e a seqüência correspondente a esta permutação é apresentada na camada de entrada.
- Passo 3: É calculado o potencial V_k para cada um dos nós de saída *k* de acordo com (2).
- Passo 4: As demais permutações dois-à-dois são realizadas, repetindo-se desde o passo 2.
- Passo 5: É identificado o nó vencedor m_0, utilizando-se (3) e definida a sua vizinhança (4). O nó vencedor representa a seqüência que obteve o melhor resultado na avaliação do objetivo (no caso, que obteve o menor N_{wt}).
- Passo 6: Os pesos W_{k^*s} na vizinhança do nó vencedor são ajustados de acordo com (8). Em seguida os pesos são normalizados de acordo com (9) e (10).
- Passo 7: Repete-se o processo desde o passo 2, encerrando-se uma iteração. Repete-se o processo até que a variação dos pesos se torne menor que um valor ε pré-definido ou se alcance um número máximo de iterações. Em seguida β e η são reduzidos.

- Passo 8: Repete-se desde o passo 2, apresentando-se desta vez uma seqüência aleatória de ordens até que a vizinhança se reduza apenas ao nó vencedor ($\eta = 1$).

3.1.3 Simulação da Rede

Para a simulação da rede, foi implementado um algoritmo em Visual Basic cuja listagem se encontra no Anexo C. Esta implementação realiza as seguintes tarefas:

- Lê um arquivo de entrada de ordens
- Ordena estas ordens usando o método EDD
- Avalia as funções objetivo com base na seqüência obtida com a regra EDD
- Ordena estas ordens utilizando a rede neural
- Avalia as funções objetivo com base na seqüência obtida com a rede neural
- Compara os resultados dos dois métodos

À seguir será apresentado um exemplo ilustrativo do funcionamento do algoritmo. Este exemplo é baseado num conjunto de 5 ordens com as seguintes características:

Conjunto de ordens à ser seqüenciado			
id_j (identificador)	p_j (tempo de processo)	d_j (data de entrega)	z_j (prioridade)
1	1	2	1
2	5	7	1
3	3	8	1
4	9	13	1
5	7	11	1

Inicialmente é calculado o peso de cada ordem de forma a manter a proporcionalidade entre as prioridades (z) e respeitar a restrição de que o somatório dos pesos é igual a unidade. Desta forma, pode-se construir a seguinte tabela:

Conjunto de ordens à ser seqüenciado com pesos calculados			
id_j (identificador)	p_j (tempo de processo)	d_j (data de entrega)	w_j (peso)
1	1	2	0,2
2	5	7	0,2
3	3	8	0,2
4	9	13	0,2
5	7	11	0,2

A primeira etapa de utilização da rede neural é a criação de uma matriz de pesos **W** para a rede neural, os quais medem a "força" da ligação entre uma determinada ordem e sua posição na seqüência. Por uma questão de nomenclatura, é preciso cuidado para não confundir a matriz de pesos **W** da rede neural com os pesos w_j de cada ordem na avaliação da função objetivo. Para este exemplo, esta matriz terá dimensão 5x5 já que, ao final do seqüenciamento, cada uma das 5 ordens existentes ocupará uma posição. A matriz **W** é inicializada com valores aleatórios entre 0 e 1.

$$\mathbf{W} = \begin{vmatrix} 0{,}357 & 0{,}300 & 0{,}659 & 0{,}123 & 0{,}408 \\ 0{,}661 & 0{,}409 & 0{,}801 & 0{,}680 & 0{,}741 \\ 0{,}098 & 0{,}384 & 0{,}371 & 0{,}522 & 0{,}800 \\ 0{,}198 & 0{,}041 & 0{,}424 & 0{,}464 & 0{,}588 \\ 0{,}911 & 0{,}316 & 0{,}445 & 0{,}378 & 0{,}061 \end{vmatrix}$$

Os valores do parâmetro que regula a velocidade do ajuste de pesos (β) e da vizinhança (η) são inicializados com:

$$\beta = 0{,}95$$
$$\eta = N - 1 = 5 - 1 = 4$$

onde N = 5 é a quantidade de ordens a serem seqüenciadas.

Em seguida a seqüência inicial é criada utilizando-se a regra EDD:

Situação Inicial – Seqüência EDD			
j (identificador)	**p_j** (tempo de processo)	**d_j** (data de entrega)	**w_j** (peso)
1	1	2	0,2
2	5	7	0,2
3	3	8	0,2
5	7	11	0,2
4	9	13	0,2

À seguir serão calculados os potenciais V_k para as diversas permutações dois-a-dois realizadas sobre esta seqüência inicial. A primeira permutação à ser considerada é o estado atual da seqüência, o qual apresenta três ordens em atraso conforme mostrado abaixo:

Avaliação da Seqüência Inicial				
j (identificador)	**p_j** (tempo de processo)	**d_j** (data de entrega)	**c_j** (data de término)	**X_j** (situação)
1	1	2	1	1
2	5	7	6	1
3	3	8	9	0
5	7	11	16	0
4	9	13	25	0

Onde X_j indica se uma determinada ordem está atrasada ($X_j = 0$) ou não ($X_j = 1$).

Para esta primeira seqüência, V_k será calculado da seguinte forma:

$$V_1 = X_1 * w_1 * W_{11} + X_2 * w_2 * W_{22} + X_3 * w_3 * W_{33} + X_5 * w_5 * W_{54} + X_4 * w_4 * W_{45}$$

Substituindo-se os valores, temos:

$$V_1 = 1*0,2*0,357 + 1*0,2*0,409 + 0*0,2*0,371 + 0*0,2*0,378 + 0*0,2*0,588 \Rightarrow$$
$$\Rightarrow V_1 = 0,153$$

À seguir é realizada a primeira troca dois-a-dois entre, obtendo-se a nova seqüência:

Avaliação da Segunda Seqüência				
j (identificador)	**p_j** (tempo de processo)	**d_j** (data de entrega)	**c_j** (data de término)	**X_j** (situação)
2	5	7	5	1
1	1	2	6	0
3	3	8	9	0
5	7	11	16	0
4	9	13	25	0

O novo valor para V é calculado:

$$V_2 = X_2 * w_2 * W_{21} + X_1 * w_1 * W_{12} + X_3 * w_3 * W_{33} + X_5 * w_5 * W_{54} + X_4 * w_4 * W_{45}$$

Substituindo-se os valores, temos:

$$V_2 = 1*0,2*0,661 + 0*0,2*0,300 + 0*0,2*0,371 + 0*0,2*0,378 + 0*0,2*0,588 \Rightarrow$$
$$\Rightarrow V_2 = 0,132$$

A troca é desfeita para que se retorne a configuração inicial e realiza-se a troca seguinte, obtendo-se a seqüência:

Avaliação da Terceira Seqüência				
j (identificador)	**p_j** (tempo de processo)	**d_j** (data de entrega)	**c_j** (data de término)	**X_j** (situação)
1	1	2	1	1
3	3	8	4	1
2	5	7	9	0
5	7	11	16	0
4	9	13	25	0

Seguindo o raciocínio anterior:

$$V_3 = X_1 * w_1 * W_{11} + X_3 * w_3 * W_{32} + X_2 * w_2 * W_{23} + X_5 * w_5 * W_{54} + X_4 * w_4 * W_{45}$$

$$V_3 = 1*0,2*0,357 + 1*0,2*0,384 + 0*0,2*0,801 + 0*0,2*0,378 + 0*0,2*0,588 \Rightarrow$$
$$\Rightarrow V_3 = 0,148$$

Avaliação da Quarta Seqüência				
j (identificador)	**p_j** (tempo de processo)	**d_j** (data de entrega)	**c_j** (data de término)	**X_j** (situação)
1	1	2	1	1
2	5	7	6	1
5	7	11	13	0
3	3	8	16	0
4	9	13	25	0

$$V_4 = X_1 * w_1 * W_{11} + X_2 * w_2 * W_{22} + X_5 * w_5 * W_{53} + X_3 * w_3 * W_{34} + X_4 * w_4 * W_{45}$$

$$V_4 = 1*0,2*0,357 + 1*0,2*0,409 + 0*0,2*0,445 + 0*0,2*0,522 + 0*0,2*0,588 \Rightarrow$$
$$\Rightarrow V_3 = 0,153$$

Avaliação da Quinta Seqüência				
j (identificador)	**p_j** (tempo de processo)	**d_j** (data de entrega)	**c_j** (data de término)	**X_j** (situação)
1	1	2	1	1
2	5	7	6	1
3	3	8	9	0
4	9	13	18	0
5	7	11	25	0

$$V_5 = X_1 * w_1 * W_{11} + X_2 * w_2 * W_{22} + X_3 * w_3 * W_{33} + X_4 * w_4 * W_{44} + X_5 * w_5 * W_{55}$$

$$V_5 = 1*0,2*0,357 + 1*0,2*0,409 + 0*0,2*0,371 + 0*0,2*0,464 + 0*0,2*0,061 \Rightarrow$$
$$\Rightarrow V_5 = 0,153$$

Conseqüentemente, após a avaliação das cinco seqüências, tem-se o vetor V:

$$\mathbf{V} = \begin{vmatrix} 0{,}153 \\ 0{,}132 \\ 0{,}148 \\ 0{,}153 \\ 0{,}153 \end{vmatrix}$$ onde cada V_k representa uma determinada seqüência.

A etapa seguinte é a criação do vetor **m**, o qual contém os elementos do vetor V ordenados em ordem decrescente; ou seja, da melhor para a pior seqüência. Desta forma, o vetor **m** é representado como:

$$\mathbf{m} = \begin{vmatrix} 1 \\ 4 \\ 5 \\ 3 \\ 2 \end{vmatrix}$$ Ou seja, a seqüência "1" foi considerada a "vencedora".

A fase seguinte será o ajuste de pesos. Na primeira passagem, o valor da vizinhança é $\eta = 4$, fazendo com que as seqüência 1, 4, 5 e 3 estejam dentro da vizinhança. Inicia-se então a fase de ajuste de pesos para cada uma das seqüências avaliadas.

Primeiramente calcula-se o valor de α:

$$\alpha = \beta * \exp\left(\frac{V_1 - V_1}{V_1 - V_5}\right) \Rightarrow \alpha = 0{,}95 * \exp\left(\frac{0{,}153 - 0{,}153}{0{,}153 - 0{,}132}\right) = 0{,}95 * \exp(0) = 0{,}95$$

Para a seqüência de número 1, a qual está dentro da vizinhança, calcula-se os novos pesos para as ordens que não estavam atrasadas:

$$W_{11} = W_{11} + \alpha * [1 - W_{11}] \Rightarrow W_{11} = 0{,}357 + 0{,}95 * [1 - 0{,}357] \Rightarrow W_{11} = 0{,}968$$

$$W_{22} = W_{22} + \alpha * [1 - W_{22}] \Rightarrow W_{22} = 0{,}409 + 0{,}95 * [1 - 0{,}409] \Rightarrow W_{22} = 0{,}970$$

Repete-se o processo para a seqüência de número 4:

$$\alpha = \beta * \exp\left(\frac{V_1 - V_4}{V_1 - V_5}\right) \Rightarrow \alpha = 0{,}95 * \exp\left(\frac{0{,}153 - 0{,}153}{0{,}153 - 0{,}132}\right) = 0{,}95 * \exp(0) = 0{,}95$$

$$W_{11} = W_{11} + \alpha * [1 - W_{11}] \Rightarrow W_{11} = 0{,}968 + 0{,}95 * [1 - 0{,}968] \Rightarrow W_{11} = 0{,}998$$

$$W_{22} = W_{22} + \alpha * [1 - W_{22}] \Rightarrow W_{22} = 0{,}970 + 0{,}95 * [1 - 0{,}970] \Rightarrow W_{22} = 0{,}999$$

Repete-se o processo para a seqüência de número 5:

$$\alpha = \beta * \exp\left(\frac{V_1 - V_5}{V_1 - V_5}\right) \Rightarrow \alpha = 0{,}95 * \exp\left(\frac{0{,}153 - 0{,}153}{0{,}153 - 0{,}132}\right) = 0{,}95 * \exp(0) = 0{,}95$$

$$W_{11} = W_{11} + \alpha * [1 - W_{11}] \Rightarrow W_{11} = 0{,}998 + 0{,}95 * [1 - 0{,}998] \Rightarrow W_{11} = 1{,}000$$

$$W_{22} = W_{22} + \alpha * [1 - W_{22}] \Rightarrow W_{22} = 0{,}999 + 0{,}95 * [1 - 0{,}999] \Rightarrow W_{22} = 1{,}000$$

Repete-se o processo para a seqüência de número 3:

$$\alpha = \beta * \exp\left(\frac{V_1 - V_3}{V_1 - V_5}\right) \Rightarrow \alpha = 0{,}95 * \exp\left(\frac{0{,}153 - 0{,}148}{0{,}153 - 0{,}132}\right) = 0{,}95 * \exp(0{,}238) = 1{,}205$$

$$W_{11} = W_{11} + \alpha * [1 - W_{11}] \Rightarrow W_{11} = 1{,}000 + 1{,}205 * [1 - 1{,}000] \Rightarrow W_{11} = 1{,}000$$

$$W_{32} = W_{32} + \alpha * [1 - W_{32}] \Rightarrow W_{32} = 0{,}384 + 1{,}205 * [1 - 0{,}384] \Rightarrow W_{32} = 1{,}126$$

Finalmente, realiza-se o ajuste de pesos para a seqüência 2, a qual está fora da vizinhança, penalizando as ordens atrasadas:

$$W_{12} = W_{12} - \beta * W_{12} \Rightarrow W_{12} = 0{,}300 - 0{,}95 * 0{,}300 \Rightarrow W_{12} = 0{,}015$$
$$W_{33} = W_{33} - \beta * W_{33} \Rightarrow W_{33} = 0{,}371 - 0{,}95 * 0{,}371 \Rightarrow W_{33} = 0{,}019$$
$$W_{54} = W_{54} - \beta * W_{54} \Rightarrow W_{54} = 0{,}378 - 0{,}95 * 0{,}378 \Rightarrow W_{54} = 0{,}019$$
$$W_{45} = W_{45} - \beta * W_{45} \Rightarrow W_{45} = 0{,}588 - 0{,}95 * 0{,}588 \Rightarrow W_{45} = 0{,}029$$

Ao final deste processo, a matriz de pesos apresenta a seguinte configuração:

$$\mathbf{W} = \begin{vmatrix} 1{,}000 & 0{,}015 & 0{,}659 & 0{,}123 & 0{,}408 \\ 0{,}661 & 1{,}000 & 0{,}801 & 0{,}680 & 0{,}741 \\ 0{,}098 & 1{,}126 & 0{,}019 & 0{,}522 & 0{,}800 \\ 0{,}198 & 0{,}041 & 0{,}424 & 0{,}464 & 0{,}588 \\ 0{,}911 & 0{,}316 & 0{,}445 & 0{,}019 & 0{,}029 \end{vmatrix}$$

Terminada a atualização dos pesos, é necessário normalizar a matriz **W**:

$$\mathbf{W} = \begin{vmatrix} 0{,}349 & 0{,}006 & 0{,}281 & 0{,}068 & 0{,}159 \\ 0{,}230 & 0{,}400 & 0{,}341 & 0{,}376 & 0{,}289 \\ 0{,}034 & 0{,}450 & 0{,}008 & 0{,}289 & 0{,}312 \\ 0{,}069 & 0{,}016 & 0{,}181 & 0{,}257 & 0{,}229 \\ 0{,}318 & 0{,}126 & 0{,}190 & 0{,}011 & 0{,}011 \end{vmatrix}$$

Terminada esta iteração, o processo de avaliação de cinco novas seqüências recomeça com a diferença de que a seqüência inicial da nova iteração a seqüência vencedora da iteração atual. Após um certo número de iterações (ou caso a variação nos pesos se torne menor que um certo ε, os valores de β e da vizinhança são reduzidos e o processo recomeça usando como base uma permutação aleatória sobre a última seqüência vencedora. As etapas são repetidas até que a vizinhança se torne unitária ($\eta = 1$).

Uma vez terminado o processo, é preciso construir a seqüência final. Para isto se seleciona, para cada ordem, a posição que obteve o maior peso. Desta forma se constrói uma seqüência final que reflete a situação final do ajuste de pesos.

CAPÍTULO 4 - SIMULAÇÃO E ESTUDO DE CASOS

4.1 Características dos Problemas estudados

4.1.1 Problemas Baseados em Dados Reais

O primeiro grupo de problemas, é composto por três casos com a quantidade de ordens variando entre 20 e 40. Estes dados foram baseados em uma situação real existente numa fábrica de CDs (Compact-Discs).

Embora o processo de fabricação de CDs seja razoavelmente padronizado, existem alguns CDs especiais que requerem etapas adicionais de fabricação. O exemplo abordado aqui trata de um tipo de CD especial (aqui tratado por CD-X) o qual, após o processo normal de fabricação, sofre uma operação final em uma máquina especial.

O processo normal de fabricação de CDs compreende várias etapas como pré-masterização, masterização, replicação e embalagem. Os CDs to tipo CD-X, sofrem um desvio de roteiro entre a replicação e a embalagem.

Estes CDs especiais são comercializados para um grupo composto por quatro clientes com prioridades diferentes e assim definidas:

Prioridade das Ordens por Cliente		
Cliente	**Prioridade**	**% Ordens**
A	1	25%
B	2	25%
C	3	25%
D	3	25%

Ou seja, o cliente A tem a prioridade mais baixas e os clientes C e D a prioridade mais alta. Em outras palavras, a penalidade por atrasos nas ordens destinadas ao cliente A são menores do que às destinadas ao cliente B que por sua vez são menores que às

destinadas aos clientes C e D. Cada um destes quatro clientes é responsável por cerca de 25% dos pedidos de CDs especiais. Devido à natureza deste produto, o atendimento às datas de entrega é crítico e os atrasos devem ser evitados ao máximo.

Existe apenas uma máquina capaz de realizar o processo especial requerido pelos CDs do tipo CD-X. Na maior parte do ano, a demanda por este tipo de CD é relativamente baixa, não causando maiores problemas de programação. Porém, em determinadas épocas, novos CDs deste tipo são lançados e o fato de existir apenas uma máquina se torna crítico, causando um gargalo na produção. Por diversas razões, não é possível aumentar o número de máquinas específicas para a fabricação de CD-X. Desta forma, cabe aos programadores de produção reduzirem o impacto negativo dos atrasos que se tornam praticamente inevitáveis.

À medida em que chegam à fábrica pedidos de CD-X, os mesmos entram no processo normal de fabricação, atingindo em determinado momento a fase de replicação. Esta etapa é realizada em um conjunto de máquinas operando em paralelo capazes de alta produção, não sendo portanto um gargalo na produção de CD-X. Isto faz com que um grupo de ordens de produção se torne disponível simultaneamente para processo na máquina especial, caracterizando um ambiente estático de programação onde os planejadores irão tentar seqüenciar este grupo de ordens da melhor forma possível para passar pelo "gargalo".

As datas de entrega aos clientes são definidas através de níveis de serviço especificados em contratos. Para fins de programação da máquina especial para os CD-X, os programadores descontam o tempo já utilizado em etapas anteriores da programação. Assim sendo, a programação da máquina especial é feita com base nas seguintes datas de entrega.

Perfil das Ordens por Datas de Entrega		
Data de Entrega (dias)	**Data de Entrega (horas)**	**% Ordens**
2	48	80%
3	72	15%
4	96	5%

Como evidenciado na tabela acima, a maioria das ordens de CD-X deve estar pronta 48 horas após ser programada para processo na máquina especial.

Devido ao fato de ser este um processo crítico, esta máquina passa por um programa detalhado de manutenção preventiva para que possa operar sem interrupções durante 24 horas, 7 dias por semana nas épocas de pico.

A velocidade de processamento da máquina é praticamente constante, fazendo com que o tempo de processo depende unicamente da quantidade de CD-X à serem processados.

O tempo de preparação para cada ordem é pequeno e está incluído no tempo de processo. Uma análise histórica de um conjunto de ordens de produção mostra o seguinte perfil para estas ordens:

Perfil das Ordens por Tempo de Processo	
Tempo de Processo (Horas)	**% Ordens**
1	36,80%
2	14,80%
3	6,40%
4	6,40%
5	8,40%
6	5,20%
7	1,60%
8	0,00%
9	2,80%
10	3,60%
11	4,00%
12	2,40%
13	1,60%
14	0,00%
15	2,40%
16	0,00%
17	2,00%
18	0,00%
19	0,00%
20	1,60%

Com base nas características descritas acima, foram gerados conjuntos de dados que aproximam, para diferentes tamanhos de amostra, o perfil das ordens de produção reais. Para efeito de testes neste trabalho, foram gerados arquivos com 20, 30 e 40 ordens, os quais podem ser encontrados no Anexo A.

4.1.2 Problemas Baseados na Literatura

Para ilustrar este trabalho de maneira mais abrangente, foram incluidos na análise dois problemas apresentados por BAKER (1997), compostos cada um por 20 ordens de produção. Os dados de entrada podem ser vistos no Anexo A.

4.1.3 Parâmetros da Rede Neural

Como normalmente acontece com as redes neurais, existem alguns parâmetros que causam variações na solução obtida. A literatura apresenta diversas propostas para a escolha destes parâmetros como em GUERRERO et al. (1998 e 1999) e SMITH (1996). No caso do modelo implementado, existe um parâmetro denominado β que regula a taxa de aprendizado à cada iteração. O valor de β decresce à medida em que o algoritmo avança de forma a reforçar a convergência em torno de uma determinada solução. Este decréscimo é obtido multiplicando-se o valor de β por um valor constante σ, ou seja:

$$\begin{cases} \beta(0) = 0{,}95 \\ \beta(t) = \beta(t-1) * \sigma \end{cases}$$

Para todos os exemplos analisados, o valor inicial de β foi fixado em 0,95 e o valor de σ fixado em 0,99. Também foram realizados testes com diferentes valores de σ para avaliar o efeito deste parâmetro na solução obtida.

Outro parâmetro importante é o tamanho η da vizinhança no neurônio vencedor. Na proposta apresentada por GUERRERO et al. (1998 e 1999), o valor inicial da vizinhança é equivalente a metade da quantidade de neurônios existentes. Testes realizados durante o presente trabalho mostraram ser conveniente utilizar-se uma vizinhança que comece com o máximo tamanho possível, decrescendo ao longo do processo. Desta forma:

$$\eta(0) = N - 1$$
$$\eta(t+1) = \eta(t) - 1$$

4.2 Resultados

Para avaliação do método, foram realizadas simulações com cinco problemas distintos conforme descrito abaixo:

- Problema 1 (P1): Baseados em dados reais. Contendo 20 ordens de produção.
- Problema 2 (P2): Baseados em dados reais. Contendo 30 ordens de produção.
- Problema 3 (P3): Baseados em dados reais. Contendo 40 ordens de produção.
- Problema 4 (P4): Baseados em BAKER (1997). Contendo 20 ordens de produção.
- Problema 5 (P5): Baseados em BAKER (1997). Contendo 20 ordens de produção.

Para cada problema, foi usado como critério de avaliação o número ponderado de ordens em atraso (N_{wt}). Este critério foi utilizado pelas seguintes razões:

- É interessante do ponto de vista do ambiente de produção que motivou este trabalho. Uma vez que os atrasos são praticamente inevitáveis, é importante tentar reduzir a quantidade de ordens em atraso, especialmente aquelas com maior prioridade;

- É interessante do ponto de vista teórico uma vez que a criação de uma seqüência que minimize N_{wt} é um problema np-completo.

No Apêndice A, foram colocados resultados adicionais obtidos durante as simulações. Uma vez que a rede neural foi treinada utilizando-se N_{wt} como objetivo à ser minimizado, estes resultados devem ser vistos apenas como referência e não como forma de avaliar o desempenho da rede.

Os resultados obtidos com a rede neural foram comparados por aqueles obtidos com as regra EDD, a qual é utilizada pelos programadores da produção no caso da fábrica de CDs que motivou este trabalho. Desta maneira, é possível verificar se a rede traz melhorias ao método utilizado atualmente. As seqüências obtidas com a rede neural e com a regra EDD podem ser vistas no Anexo B.

Para cada um dos problemas, foram executadas cinco "rodadas" e selecionado o melhor resultado. Em vista do fato de as redes neurais, à princípio apresentarem variações sensíveis devido à sua natureza estocástica, os resultados das diversas rodadas foram comparados e escolhido o melhor.

4.2.1 Resultados Obtidos

Objetivo: Reduzir Número Ponderado de Ordens Atrasadas (N_{wt})					
Problema	**1** **N = 20**	**2** **N = 30**	**3** **N = 40**	**4** **N = 20**	**5** **N = 20**
EDD	0,125	0,348	0,467	1,000	0,755
SOFM	0,125	0,261	0,400	0,561	0,633
% Melhoria	0%	25,0%	14,3%	43,9%	16,2%

4.3 Discussão dos Resultados

4.3.1 Resultados Obtidos para os Diversos Objetivos

A rede neural foi treinada utilizando-se o número ponderado de ordens em atraso (N_{wt}) para que a rede fosse capaz de fornecer respostas apropriadas para estes objetivos.

Observando-se os valores obtidos para os diversos critérios, notam-se os seguintes resultados:

- Os resultados para N_{wt} superaram os obtidos com o método EDD pelo menos em uma das tentativas com exceção do Problema 1 onde os resultados foram semelhantes.
- Pode-se concluir que a rede foi capaz de encontrar melhorias sobre a seqüência inicial construída com a regra EDD, o que significa que o método é útil especialmente para problemas np-completos onde não existe uma solução ótima conhecida.

4.3.2 Repetibilidade dos Resultados

Para cada um dos cinco problemas utilizados, a rede neural foi executada cinco vezes para que se pudesse comparar os resultados. Em cada um dos problemas estudados, as cinco rodadas forneceram resultados razoavelmente próximos para o número ponderado de ordens em atraso (N_{wt}), demonstrando uma estabilidade no processo. Notou-se que esta estabilidade está relacionada à uma escolha adequada dos parâmetros da rede.

4.3.3 Impacto da Escolha dos Parâmetros

Como normalmente acontece com as redes neurais, notou-se uma dependência entre os parâmetros escolhidos para os testes e os resultados obtidos, especialmente quando a quantidade de ordens aumenta. Os exemplos utilizados neste trabalho foram baseados no seguintes valores para a taxa de aprendizado β:

$$\begin{cases} \beta(0) = 0{,}95 \\ \beta(t) = \beta(t-1) * \sigma \end{cases}$$

Foram realizados experimentos onde se variou o valor de σ e o valor de $\beta(0)$. Nestes testes, notou-se que valores baixos altos para estes dois parâmetros tendem a melhorar a convergência da rede e a repetibilidade dos resultados.

4.3.4 Convergência da Rede

A rede convergiu em todos os casos testados onde se limitou o número máximo de iterações à 2000. Porém, em alguns casos, o estado final apresentado indicava mais de uma posição na seqüência possível de ser ocupada por uma determinada ordem. Este fenômeno acontece nos casos em que, dado um certo objetivo utilizado para o treinamento, uma determinada ordem pode ocupar posições diferentes na seqüência sem que isto altere o objetivo. Para estes casos, o algoritmo simplesmente selecionou uma das ordens possíveis e considerou esta ordem como "alocada" para que a mesma não fosse repetida em outra posição. Desta forma, pode-se dizer que o algoritmo convergiu em todas as situações testadas.

CAPÍTULO 5 - CONCLUSÕES

No presente trabalho foi analisada a aplicação das redes neurais "Self-Organizing" ao problema de "scheduling", com o objetivo de verificar se este método pode auxiliar no seqüenciamento de ordens de produção aguardando numa fila para processamento em uma determinada máquina.

Esta análise é interessante na medida em que este problema é freqüente em diversas aplicações. Além disso, o caso específico de uma única máquina possibilita verificar com maior clareza alguns aspectos do modelo em questão.

O modelo estudado demonstrou bons resultados em problemas de diferentes tamanhos onde uma solução matemática exata não está disponível, superando os resultados obtidos com regras básicas de seqüenciamento. Para problemas onde esta solução existe, a rede neural forneceu resultados pelo menos tão bons quanto uma das regras utilizadas como comparação.

Apesar da notória sensibilidade das redes neurais aos parâmetros utilizados no treinamento, o algoritmo implementado exibiu pouca dependência destes parâmetros. Porém, para trabalhos futuros, sugere-se uma exploração mais detalhada destes parâmetros com o objetivo de se definir regras para a seleção dos mesmos. Também se torna interessante a tentativa de se utilizar diferentes regras para a criação da "raiz" (estado inicial) da rede neural.

A motivação para este trabalho veio de situações existentes na indústria para as quais existe uma carência de soluções relativamente rápidas mesmo que não sendo ótimas. Sem prejuízo da aplicabilidade dos resultados a situações reais, foram feitas algumas simplificações no modelo real de forma a possibilitar um entendimento das variáveis

importantes do modelo proposto. Desta forma, como extensão do presente trabalho, sugere-se uma análise do comportamento do modelo proposto nas seguintes situações:

- Existência de máquinas em paralelo;
- Permissão de "preemption";
- Permissão de inserção de esperas ("inserted idleness") ;
- Possibilidade de interrupção das máquinas por razões de manutenção.

Finalmente, a algoritmo implementado foi otimizado para atingir como principais objetivos uma redução no número de ordens em atraso e no atraso ponderado da seqüência obtida. Ainda como sugestão para trabalhos futuros, propõe-se a utilização de objetivos múltiplos agregados através de um sistema baseado em lógica nebulosa, permitindo a combinação de diversos objetivos simultaneamente.

ANEXOS

Anexo A – Arquivos de Entrada Utilizados

Nas tabelas apresentadas abaixo, o significado de cada coluna é o seguinte:

- id: É o "nome" da ordem que a identifica unicamente;
- p: Quantidade de tempo necessária para o processamento da ordem;
- r: É a data em que a ordem é liberada para a produção. Neste caso, o conjunto de ordens à ser programado é liberado para a produção simultaneamente, fazendo com que se possa considerar todas as ordens disponíveis para trabalho no tempo inicial (zero), caracterizando um ambiente estático;
- d: Data de entrega da ordem;
- z: Índice de prioridade da j-ésima ordem. Quanto maior o valor de z_j, maior a prioridade.

Problema 1 – Baseado em dados reais - 20 Ordens de Produção				
id	**p**	**r**	**d**	**z**
1	1	0	48	3
2	1	0	48	3
3	1	0	48	3
4	1	0	48	1
5	1	0	48	1
6	1	0	48	2
7	1	0	48	2
8	2	0	48	3
9	2	0	48	3
10	5	0	48	3
11	2	0	48	1
12	2	0	48	2
13	1	0	72	3
14	3	0	48	3
15	4	0	48	3
16	6	0	48	3
17	5	0	48	1
18	5	0	48	2
19	11	0	48	3
20	10	0	48	3

Problema 2 – Baseado em BAKER (1997) - 20 Ordens de Produção				
id	**p**	**r**	**d**	**z**
1	55	0	109	10
2	68	0	169	9
3	70	0	1039	7
4	73	0	1158	6
5	77	0	1107	13
6	78	0	0	9
7	85	0	767	12
8	86	0	993	15
9	89	0	643	14
10	89	0	667	10
11	92	0	75	6
12	93	0	612	8
13	94	0	780	12
14	94	0	816	13
15	98	0	721	12
16	108	0	555	5
17	126	0	1166	13
18	138	0	529	8
19	143	0	0	8
20	170	0	1237	6

Problema 3 – Baseado em BAKER (1997) - 20 Ordens de Produção				
id	**p**	**r**	**d**	**z**
1	68	0	437	7
2	79	0	521	5
3	80	0	678	13
4	86	0	841	7
5	89	0	746	6
6	94	0	520	7
7	96	0	610	12
8	97	0	1112	12
9	100	0	772	12
10	105	0	566	8
11	106	0	928	8
12	109	0	472	8
13	109	0	910	12
14	112	0	499	9
15	118	0	498	15
16	119	0	1084	6
17	120	0	617	12
18	124	0	1153	5
19	127	0	1120	14
20	135	0	974	10

Problema 4 – Baseado em dados reais – 30 Ordens de Produção				
id	**p**	**r**	**d**	**z**
1	1	0	48	3
2	1	0	48	3
3	1	0	48	3
4	1	0	48	3
5	1	0	48	3
6	1	0	48	1
7	1	0	48	1
8	1	0	48	2
9	1	0	48	2
10	2	0	48	3
11	2	0	48	3
12	5	0	48	3
13	2	0	48	1
14	2	0	48	2
15	1	0	72	3
16	3	0	48	3
17	4	0	48	3
18	6	0	48	3
19	5	0	48	1
20	5	0	48	2
21	11	0	48	3
22	10	0	48	3
23	1	0	72	1
24	1	0	72	2
25	3	0	48	1
26	3	0	48	2
27	4	0	48	1
28	4	0	48	2
29	9	0	48	3
30	2	0	72	3

Problema 5 – Baseado em dados reais - 40 Ordens de Produção				
id	**p**	**r**	**d**	**z**
1	1	0	48	3
2	1	0	48	3
3	1	0	48	3
4	1	0	48	3
5	1	0	48	3
6	1	0	48	1
7	1	0	48	1
8	1	0	48	1
9	1	0	48	2
10	1	0	48	2
11	1	0	48	2
12	2	0	48	3
13	2	0	48	3
14	5	0	48	3
15	2	0	48	1
16	2	0	48	2
17	1	0	72	3
18	3	0	48	3
19	4	0	48	3
20	6	0	48	3
21	5	0	48	1
22	5	0	48	2
23	11	0	48	3
24	10	0	48	3
25	1	0	72	1
26	1	0	72	2
27	3	0	48	1
28	3	0	48	2
29	4	0	48	1
30	4	0	48	2
31	9	0	48	3
32	2	0	72	3
33	6	0	48	1
34	6	0	48	2
35	12	0	48	3
36	15	0	48	3
37	1	0	96	3
38	11	0	48	1
39	11	0	48	2
40	17	0	48	3

Anexo B – Seqüências Obtidas

Nas tabelas apresentadas abaixo, o significado de cada coluna é o seguinte:

- id: É o "nome" da ordem que a identifica unicamente;
- s: É a data de início do processamento da ordem;
- p: É a quantidade de tempo necessária para o processamento da ordem;
- c: É a data de término do processamento da ordem;
- d: É a data de entrega da ordem;
- T: É o atraso da ordem. Ou seja, a diferença positiva entre a data de término e a data de entrega;
- w: É o peso da ordem, calculado de acordo com a prioridade z da ordem.

1. Seqüências Obtidas com a Regra EDD

Problema 1 – Baseado em dados reais - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,063
2	1	1	2	48	0	0,063
3	2	1	3	48	0	0,063
4	3	1	4	48	0	0,021
5	4	1	5	48	0	0,021
6	5	1	6	48	0	0,042
7	6	1	7	48	0	0,042
8	7	2	9	48	0	0,063
9	9	2	11	48	0	0,063
10	11	5	16	48	0	0,063
11	16	2	18	48	0	0,021
12	18	2	20	48	0	0,042
14	20	3	23	48	0	0,063
15	23	4	27	48	0	0,063
16	27	6	33	48	0	0,063
17	33	5	38	48	0	0,021
18	38	5	43	48	0	0,042
19	43	11	54	48	6	0,063
20	54	10	64	48	16	0,063
13	64	1	65	72	0	0,063
Regra EDD : $N_{tot} = 2$; $N_{wt} = 0,125$; $T_{max} = 16$; $T_{tot} = 22$; $T_{wt} = 1,375$						

Problema 2 – Baseado em dados reais – 30 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,043
2	1	1	2	48	0	0,043
3	2	1	3	48	0	0,043
4	3	1	4	48	0	0,043
5	4	1	5	48	0	0,043
6	5	1	6	48	0	0,014
7	6	1	7	48	0	0,014
8	7	1	8	48	0	0,029
9	8	1	9	48	0	0,029
10	9	2	11	48	0	0,043
11	11	2	13	48	0	0,043
12	13	5	18	48	0	0,043
13	18	2	20	48	0	0,014
14	20	2	22	48	0	0,029
16	22	3	25	48	0	0,043
17	25	4	29	48	0	0,043
18	29	6	35	48	0	0,043
19	35	5	40	48	0	0,014
20	40	5	45	48	0	0,029
21	45	11	56	48	8	0,043
22	56	10	66	48	18	0,043
25	66	3	69	48	21	0,014
26	69	3	72	48	24	0,029
27	72	4	76	48	28	0,014
28	76	4	80	48	32	0,029
29	80	9	89	48	41	0,043
15	89	1	90	72	18	0,043
23	90	1	91	72	19	0,014
24	91	1	92	72	20	0,029
30	92	2	94	72	22	0,043
Regra EDD : $N_{tot} = 11$; $N_{wt} = 0,348$; $T_{max} = 41$; $T_{tot} = 251$; $T_{wt} = 7,84$						

Problema 3 – Baseado em dados reais – 40 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,033
2	1	1	2	48	0	0,033
3	2	1	3	48	0	0,033
4	3	1	4	48	0	0,033
5	4	1	5	48	0	0,033
6	5	1	6	48	0	0,011
7	6	1	7	48	0	0,011
8	7	1	8	48	0	0,011
9	8	1	9	48	0	0,022
10	9	1	10	48	0	0,022
11	10	1	11	48	0	0,022
12	11	2	13	48	0	0,033
13	13	2	15	48	0	0,033
14	15	5	20	48	0	0,033
15	20	2	22	48	0	0,011
16	22	2	24	48	0	0,022
18	24	3	27	48	0	0,033
19	27	4	31	48	0	0,033
20	31	6	37	48	0	0,033
21	37	5	42	48	0	0,011
22	42	5	47	48	0	0,022
23	47	11	58	48	10	0,033
24	58	10	68	48	20	0,033
27	68	3	71	48	23	0,011
28	71	3	74	48	26	0,022
29	74	4	78	48	30	0,011
30	78	4	82	48	34	0,022
31	82	9	91	48	43	0,033
33	91	6	97	48	49	0,011
34	97	6	103	48	55	0,022
35	103	12	115	48	67	0,033
36	115	15	130	48	82	0,033
38	130	11	141	48	93	0,011
39	141	11	152	48	104	0,022
40	152	17	169	48	121	0,033
17	169	1	170	72	98	0,033
25	170	1	171	72	99	0,011
26	171	1	172	72	100	0,022
32	172	2	174	72	102	0,033
37	174	1	175	96	79	0,033
Regra EDD : N_{tot} = 19 ; N_{wt} = 0,467 ; T_{max} = 121 ; T_{tot} = 1235 ; T_{wt} = 31,09						

Problema 4 – Baseado em BAKER (1997) - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
6	0	78	78	0	78	0,046
19	78	143	221	0	221	0,041
11	221	92	313	75	238	0,031
1	313	55	368	109	259	0,051
2	368	68	436	169	267	0,046
18	436	138	574	529	45	0,041
16	574	108	682	555	127	0,026
12	682	93	775	612	163	0,041
9	775	89	864	643	221	0,071
10	864	89	953	667	286	0,051
15	953	98	1051	721	330	0,061
7	1051	85	1136	767	369	0,061
13	1136	94	1230	780	450	0,061
14	1230	94	1324	816	508	0,066
8	1324	86	1410	993	417	0,077
3	1410	70	1480	1039	441	0,036
5	1480	77	1557	1107	450	0,066
4	1557	73	1630	1158	472	0,031
17	1630	126	1756	1166	590	0,066
20	1756	170	1926	1237	689	0,031
Regra EDD : N_{tot} = 20 ; N_{wt} = 1,000 ; T_{max} = 689 ; T_{tot} = 6621 ; T_{wt} = 343,69						

Problema 5 – Baseado em BAKER (1997) - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	68	68	437	0	0,037
12	68	109	177	472	0	0,043
15	177	118	295	498	0	0,080
14	295	112	407	499	0	0,048
6	407	94	501	520	0	0,037
2	501	79	580	521	59	0,027
10	580	105	685	566	119	0,043
7	685	96	781	610	171	0,064
17	781	120	901	617	284	0,064
3	901	80	981	678	303	0,069
5	981	89	1070	746	324	0,032
9	1070	100	1170	772	398	0,064
4	1170	86	1256	841	415	0,037
13	1256	109	1365	910	455	0,064
11	1365	106	1471	928	543	0,043
20	1471	135	1606	974	632	0,053
16	1606	119	1725	1084	641	0,032
8	1725	97	1822	1112	710	0,064
19	1822	127	1949	1120	829	0,074
18	1949	124	2073	1153	920	0,027
Regra EDD : N_{tot} = 15 ; N_{wt} = 0,755 ; T_{max} = 920 ; T_{tot} = 6803 ; T_{wt} = 345,57						

2. Seqüências Obtidas com a Rede Neural

Problema 1 – Baseado em dados reais - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,063
2	1	1	2	48	0	0,063
3	2	1	3	48	0	0,063
4	3	1	4	48	0	0,021
5	4	1	5	48	0	0,021
6	5	1	6	48	0	0,042
7	6	1	7	48	0	0,042
8	7	2	9	48	0	0,063
9	9	2	11	48	0	0,063
10	11	5	16	48	0	0,063
11	16	2	18	48	0	0,021
12	18	2	20	48	0	0,042
14	20	3	23	48	0	0,063
13	23	1	24	72	0	0,063
15	24	4	28	48	0	0,063
16	28	6	34	48	0	0,063
17	34	5	39	48	0	0,021
18	39	5	44	48	0	0,042
19	44	11	55	48	7	0,063
20	55	111	65	48	17	0,063
Método SOFM : $N_{tot} = 2$; $N_{wt} = 0,125$; $T_{max} = 17$; $T_{bot} = 24$; $T_{wt} = 1,50$						

Problema 2 – Baseado em dados reais – 30 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,043
2	1	1	2	48	0	0,043
3	2	1	3	48	0	0,043
4	3	1	4	48	0	0,043
5	4	1	5	48	0	0,043
6	5	1	6	48	0	0,014
7	6	1	7	48	0	0,014
8	7	1	8	48	0	0,029
9	8	1	9	48	0	0,029
10	9	2	11	48	0	0,043
11	11	2	13	48	0	0,043
12	13	5	18	48	0	0,043
13	18	2	20	48	0	0,014
14	20	2	22	48	0	0,029
16	22	3	25	48	0	0,043
17	25	4	29	48	0	0,043
15	29	1	30	72	0	0,043
18	30	6	36	48	0	0,043
19	36	5	41	48	0	0,014
20	41	5	46	48	0	0,029
21	46	11	57	48	9	0,043
22	57	10	67	48	19	0,043
23	67	1	68	72	0	0,014
24	68	1	69	72	0	0,029
25	69	3	72	48	24	0,014
26	72	3	75	48	27	0,029
27	75	4	79	48	31	0,014
28	79	4	83	48	35	0,029
29	83	9	92	48	44	0,043
30	92	2	94	72	22	0,043
Método SOFM : $N_{tot} = 8$; $N_{wt} = 0,261$; $T_{max} = 44$; $T_{tot} = 211$; $T_{wt} = 6,68$						

Problema 3 – Baseado em dados reais – 40 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
2	0	1	1	48	0	0,033
1	1	1	2	48	0	0,033
3	2	1	3	48	0	0,033
4	3	1	4	48	0	0,033
5	4	1	5	48	0	0,033
6	5	1	6	48	0	0,011
7	6	1	7	48	0	0,011
8	7	1	8	48	0	0,011
9	8	1	9	48	0	0,022
10	9	1	10	48	0	0,022
11	10	1	11	48	0	0,022
12	11	2	13	48	0	0,033
13	13	2	15	48	0	0,033
14	15	5	20	48	0	0,033
15	20	2	22	48	0	0,011
16	22	2	24	48	0	0,022
18	24	3	27	48	0	0,033
17	27	1	28	72	0	0,033
19	28	4	32	48	0	0,033
20	32	6	38	48	0	0,033
21	38	5	43	48	0	0,011
22	43	5	48	48	0	0,022
23	48	11	59	48	11	0,033
24	59	10	69	48	21	0,033
25	69	1	70	72	0	0,011
26	70	1	71	72	0	0,022
27	71	3	74	48	26	0,011
28	74	3	77	48	29	0,022
29	77	4	81	48	33	0,011
30	81	4	85	48	37	0,022
31	85	9	94	48	46	0,033
32	94	2	96	72	24	0,033
33	96	6	102	48	54	0,011
34	102	6	108	48	60	0,022
35	108	12	120	48	72	0,033
36	120	15	135	48	87	0,033
38	135	11	146	48	98	0,011
37	146	11	147	96	51	0,033
39	147	11	158	48	110	0,022
40	158	17	175	48	127	0,033
Método SOFM : N_{tot} = 19 ; N_{wt} = 0,400 ; T_{max} = 127 ; T_{tot} = 886 ; T_{wt} = 22,22						

Problema 4 – Baseado em BAKER (1997) - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	68	68	437	0	0,037
2	68	79	147	521	0	0,027
3	147	80	227	678	0	0,069
5	227	89	316	746	0	0,032
6	316	94	410	520	0	0,037
4	410	86	496	841	0	0,037
7	496	96	592	610	0	0,064
8	592	97	689	1112	0	0,064
10	689	105	794	566	228	0,043
9	794	100	894	772	122	0,064
12	894	109	1003	472	531	0,043
11	1003	106	1109	928	181	0,043
14	1109	112	1221	499	722	0,048
13	1221	109	1330	910	420	0,064
15	1330	118	1448	498	950	0,080
16	1448	119	1567	1084	483	0,032
17	1567	120	1687	617	1070	0,064
20	1687	135	1822	974	848	0,053
19	1822	127	1949	1120	829	0,074
18	1949	124	2073	1153	920	0,027
Método SOFM : N_{tot} = 12 ; N_{wt} = 0,561 ; T_{max} = 1618 ; T_{tot} = 7417 ; T_{wt} = 314,75						

Problema 5 – Baseado em BAKER (1997) - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	68	68	437	0	0,037
3	68	80	148	678	0	0,069
2	148	79	227	521	0	0,027
5	227	89	316	746	0	0,032
4	316	86	402	841	0	0,037
6	402	94	496	520	0	0,037
7	496	96	592	610	0	0,064
8	592	97	689	1112	0	0,064
10	689	105	794	566	228	0,043
9	794	100	894	772	122	0,064
12	894	109	1003	472	531	0,043
11	1003	106	1109	928	181	0,043
14	1109	112	1221	499	722	0,048
13	1221	109	1330	910	420	0,064
15	1330	118	1448	498	950	0,080
16	1448	119	1567	1084	483	0,032
17	1567	120	1687	617	1070	0,064
20	1687	135	1822	974	848	0,053
19	1822	127	1949	1120	829	0,074
18	1949	124	2073	1153	920	0,027
Método SOFM : N_{tot} = 12 ; N_{wt} = 0,633 ; T_{max} = 1070 ; T_{tot} = 7304 ; T_{wt} = 399,98						

Anexo C – Listagem do Programa em Visual Basic

```
'/*****************************************************
'*                                                    *
'*                   SISTEMA COELET                   *
'*                                                    *
'*                                                    *
'*   AUTOR:  Leonardo Wellisch de Barros Barreto      *
'*                                                    *
'*   ORIENTADOR: Prof. Marcos R. P. Barretto          *
'*                                                    *
'*   DEPARTAMENTO DE ENGENHARIA MECÂNICA              *
'*                                                    *
'*   UNIVERSIDADE DE SÃO PAULO                        *
'*                                                    *
'*   MÓDULO:     COELET.BAS                           *
'*   CONTEÚDO:   Funções Globais                      *
'*                                                    *
'*   REVISÃO:    3.00                                 *
'*   DATA:       27 / Nov / 2000                      *
'*                                                    *
'*                                                    *
'*                                                    *
'*****************************************************

'*****************************************************
'*                                                    *
'*  CONSTANTES GLOBAIS                                *
'*                                                    *
'*****************************************************

Global Const c_Sigma_Step = 0.1
Global Const c_Sigma = 0.99
Global Const c_Beta_Inicial_Step = 0.1
Global Const c_Beta_Inicial = 0.95
Global Const c_COL_MAX = 200
Global Const c_Makespan_MAX = 200
Global Const c_Epoca_MAX = 2000
Global Const c_OPs_MAX = c_COL_MAX
Global Const c_ROW_MAX = c_Makespan_MAX
Global Const c_Epsilon = 0.00001
Global Const c_Sim_MAX = 2

'*****************************************************
'*                                                    *
'*  DEFINIÇÃO DE TIPOS                                *
'*                                                    *
'*****************************************************

Type Tipo_OI
    Status  As Integer
    C   As Integer
    id  As Integer
    d   As Integer
    L   As Integer
    p   As Integer
    r   As Integer
    i   As Integer ' data de inicio
    T   As Integer
    Tw  As Double
    w   As Double
    z   As Integer
End Type

Type Tipo_SCHD
    Cmax    As Integer
    Lavg    As Double
    Lmax    As Integer
    Ltot    As Integer
    Ntot    As Integer
    Nwt     As Double
    Tmax    As Integer
    Ttot    As Integer
    Twt     As Double
End Type

'*****************************************************
'*                                                    *
'*  DECLARAÇÃO DE VARIÁVEIS GLOBAIS                   *
'*                                                    *
'*****************************************************
```

```
Global ArqInOp                  As String
Global ArqOutEDD                As String
Global ArqOutSim                As String
Global ArqOutSOFM               As String
Global ArqOutSPT                As String
Rem Global Sigma                    As Double
Global intMsg                   As Integer
Global M                        As Integer
Global m0                       As Integer
Global N_OPs                    As Integer
Global N                        As Integer
Global Schd_EDD                 As Tipo_SCHD
Global Schd_SOFM                As Tipo_SCHD
Global Schd_SPT                 As Tipo_SCHD
Global V(c_ROW_MAX)             As Double
Global Vet_OPs(c_OPs_MAX)       As Tipo_OP
Global Vet_OPs_AUX(c_OPs_MAX)   As Tipo_OP
Global Vet_OPs_EDD(c_OPs_MAX)   As Tipo_OP
Global Vet_OPs_SOFM(c_OPs_MAX)  As Tipo_OP
Global Vet_OPs_SPT(c_OPs_MAX)   As Tipo_OP
Global Vet_OPs_X(c_OPs_MAX)         As Tipo_OP
Global Mat_W(c_ROW_MAX, c_COL_MAX)  As Double



'*****************************************************
'*                                                   *
'*  DECLARAÇÃO DE FUNÇÕES GLOBAIS                    *
'*                                                   *
'*****************************************************

Public Sub Aloca_OPs(opcao As Byte)
    Dim d1, d2  As Integer
    Dim j       As Integer

    d1 = 0
    d2 = 0

    Select Case opcao
        Case 1
            For j = 0 To N_OPs
                Vet_OPs_EDD(j).i = d1
                d2 = d1 + Vet_OPs_EDD(j).p
                Vet_OPs_EDD(j).C = d2
                Vet_OPs_EDD(j).L = d2 - Vet_OPs_EDD(j).d
                If (Vet_OPs_EDD(j).L >= 0) Then
                    Vet_OPs_EDD(j).T = Vet_OPs_EDD(j).L
                Else
                    Vet_OPs_EDD(j).T = 0
                End If
                d1 = d2
            Next j
        Case 2
            For j = 0 To N_OPs
                Vet_OPs_SOFM(j).i = d1
                d2 = d1 + Vet_OPs_SOFM(j).p
                Vet_OPs_SOFM(j).C = d2
                Vet_OPs_SOFM(j).L = d2 - Vet_OPs_SOFM(j).d
                If (Vet_OPs_SOFM(j).L >= 0) Then
                    Vet_OPs_SOFM(j).T = Vet_OPs_SOFM(j).L
                Else
                    Vet_OPs_SOFM(j).T = 0
                End If
                d1 = d2
            Next j
        Case 3
            For j = 0 To N_OPs
                Vet_OPs_SPT(j).i = d1
                d2 = d1 + Vet_OPs_SPT(j).p
                Vet_OPs_SPT(j).C = d2
                Vet_OPs_SPT(j).L = d2 - Vet_OPs_SPT(j).d
                If (Vet_OPs_SPT(j).L >= 0) Then
                    Vet_OPs_SPT(j).T = Vet_OPs_SPT(j).L
                Else
                    Vet_OPs_SPT(j).T = 0
                End If
                d1 = d2
            Next j
        End Select

End Sub

'*****************************************************
```

```
'*                                                    *
'*  FUNÇÃO Cal_SCHD                                   *
'*                                                    *
'******************************************************

Public Sub Cal_SCHD(opcao As Byte)
    Dim j   As Integer
    Dim aux As Double

    Select Case opcao
    Case 1

    '***** Calcula Cmax

        Schd_EDD.Cmax = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = Vet_OPs_EDD(j).C
            If (aux > Schd_EDD.Cmax) Then
                Schd_EDD.Cmax = aux
            End If
        Next j

    '***** Calcula Lavg

        Schd_EDD.Lavg = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = aux + Vet_OPs_EDD(j).L
        Next j
        Schd_EDD.Lavg = aux / N_OPs

    '***** Calcula Lmax

        Schd_EDD.Lmax = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = Abs(Vet_OPs_EDD(j).L)
            If (aux > Schd_EDD.Lmax) Then
                Schd_EDD.Lmax = aux
            End If
        Next j

    '***** Calcula Ltot

        Schd_EDD.Ltot = 0
        For j = 0 To (N_OPs - 1)
            Schd_EDD.Ltot = Schd_EDD.Ltot + Vet_OPs_EDD(j).L
        Next j

    '***** Calcula Ntot

        Schd_EDD.Ntot = 0
        For j = 0 To (N_OPs - 1)
            If (Vet_OPs_EDD(j).T > 0) Then
                Schd_EDD.Ntot = Schd_EDD.Ntot + 1
            End If
        Next j

    '***** Calcula Nwt

        Schd_EDD.Nwt = 0
        For j = 0 To (N_OPs - 1)
            If (Vet_OPs_EDD(j).T > 0) Then
                Schd_EDD.Nwt = Schd_EDD.Nwt + Vet_OPs_EDD(j).w
            End If
        Next j

    '***** Calcula Tmax

        Schd_EDD.Tmax = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = Vet_OPs_EDD(j).T
            If (aux > Schd_EDD.Tmax) Then
                Schd_EDD.Tmax = aux
            End If
        Next j

    '***** Calcula Ttot

        Schd_EDD.Ttot = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = Vet_OPs_EDD(j).T
```

```
        Schd_EDD.Ttot = Schd_EDD.Ttot + aux
    Next j

'***** Calcula Twt

    Schd_EDD.Twt = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_EDD(j).T * Vet_OPs_EDD(j).w
        Schd_EDD.Twt = Schd_EDD.Twt + aux
    Next j


Case 2

'***** Calcula Cmax

    Schd_SOFM.Cmax = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_SOFM(j).C
        If (aux > Schd_SOFM.Cmax) Then
            Schd_SOFM.Cmax = aux
        End If
    Next j

'***** Calcula Lavg

    Schd_SOFM.Lavg = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = aux + Vet_OPs_SOFM(j).L
    Next j
    Schd_SOFM.Lavg = aux / N_OPs

'***** Calcula Lmax

    Schd_SOFM.Lmax = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Abs(Vet_OPs_SOFM(j).L)
        If (aux > Schd_SOFM.Lmax) Then
            Schd_SOFM.Lmax = aux
        End If
    Next j

'***** Calcula Ltot

    Schd_SOFM.Ltot = 0
    For j = 0 To (N_OPs - 1)
        Schd_SOFM.Ltot = Schd_SOFM.Ltot + Vet_OPs_SOFM(j).L
    Next j

'***** Calcula Ntot

    Schd_SOFM.Ntot = 0
    For j = 0 To (N_OPs - 1)
        If (Vet_OPs_SOFM(j).T > 0) Then
            Schd_SOFM.Ntot = Schd_SOFM.Ntot + 1
        End If
    Next j

'***** Calcula Nwt

    Schd_SOFM.Nwt = 0
    For j = 0 To (N_OPs - 1)
        If (Vet_OPs_SOFM(j).T > 0) Then
            Schd_SOFM.Nwt = Schd_SOFM.Nwt + Vet_OPs_SOFM(j).w
        End If
    Next j

'***** Calcula Tmax

    Schd_SOFM.Tmax = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_SOFM(j).T
        If (aux > Schd_SOFM.Tmax) Then
            Schd_SOFM.Tmax = aux
        End If
    Next j

'***** Calcula Ttot

    Schd_SOFM.Ttot = 0
```

```
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_SOFM(j).T
        Schd_SOFM.Ttot = Schd_SOFM.Ttot + aux
    Next j

'***** Calcula Twt

    Schd_SOFM.Twt = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_SOFM(j).T * Vet_OPs_SOFM(j).w
        Schd_SOFM.Twt = Schd_SOFM.Twt + aux
    Next j

Case 3
    '***** Calcula Cmax

    Schd_SPT.Cmax = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_SPT(j).C
        If (aux > Schd_SPT.Cmax) Then
            Schd_SPT.Cmax = aux
        End If
    Next j

'***** Calcula Lavg

    Schd_SPT.Lavg = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = aux + Vet_OPs_SPT(j).L
    Next j
    Schd_SPT.Lavg = aux / N_OPs

'***** Calcula Lmax

    Schd_SPT.Lmax = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Abs(Vet_OPs_SPT(j).L)
        If (aux > Schd_SPT.Lmax) Then
            Schd_SPT.Lmax = aux
        End If
    Next j

'***** Calcula Ltot

    Schd_SPT.Ltot = 0
    For j = 0 To (N_OPs - 1)
        Schd_SPT.Ltot = Schd_SPT.Ltot + Vet_OPs_SPT(j).L
    Next j

'***** Calcula Ntot

    Schd_SPT.Ntot = 0
    For j = 0 To (N_OPs - 1)
        If (Vet_OPs_SPT(j).T > 0) Then
            Schd_SPT.Ntot = Schd_SPT.Ntot + 1
        End If
    Next j

'***** Calcula Nwt

    Schd_SPT.Nwt = 0
    For j = 0 To (N_OPs - 1)
        If (Vet_OPs_SPT(j).T > 0) Then
            Schd_SPT.Nwt = Schd_SPT.Nwt + Vet_OPs_SPT(j).w
        End If
    Next j

'***** Calcula Tmax

    Schd_SPT.Tmax = 0
    aux = 0
    For j = 0 To (N_OPs - 1)
        aux = Vet_OPs_SPT(j).T
        If (aux > Schd_SPT.Tmax) Then
            Schd_SPT.Tmax = aux
        End If
    Next j

'***** Calcula Ttot
```

```
        Schd_SPT.Ttot = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = Vet_OPs_SPT(j).T
            Schd_SPT.Ttot = Schd_SPT.Ttot + aux
        Next j

    '***** Calcula Twt

        Schd_SPT.Twt = 0
        aux = 0
        For j = 0 To (N_OPs - 1)
            aux = Vet_OPs_SPT(j).T * Vet_OPs_SPT(j).w
            Schd_SPT.Twt = Schd_SPT.Twt + aux
        Next j


    End Select

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Calcula_MN                                *
'*                                                   *
'*****************************************************

Public Sub Calcula_MN()
    Dim j   As Integer

    M = N_OPs
    N = N_OPs

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Calcula_T                                 *
'*                                                   *
'*****************************************************

Public Function Calcula_T(k, s As Integer) As Integer
    Dim Laux, Taux As Integer

    Laux = (s + Vet_OPs_SOFM(k).p) - Vet_OPs_SOFM(k).d
    If (Laux >= 0) Then
        Taux = Laux
    Else
        Taux = 0
    End If
    Calcula_T = Taux
End Function

'*****************************************************
'*                                                   *
'*  FUNÇÃO Cria_Nomes_ArqOut                         *
'*                                                   *
'*****************************************************

Public Sub Cria_Nomes_ArqOut()
    Dim aux As String
    Dim tam As Integer

    tam = Len(ArqInOp)
    aux = Left(ArqInOp, (tam - 4))

    ArqOutEDD = aux + "EDD" + "-v3.00" + ".txt"
    ArqOutSim = aux + "SIMULA" + "-v3.00" + ".txt"
    ArqOutSOFM = aux + "SOFM" + "-v3.00" + ".txt"
    ArqOutSPT = aux + "SPT" + "-v3.00" + ".txt"

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Imprime_Resultado                         *
'*                                                   *
'*****************************************************

Public Sub Imprime_Resultado(opcao As Byte)

    Dim j, k, s As Integer

    Select Case opcao
        Case 1
```

```
        Open ArqOutEDD For Output As #1
    Case 2
        Open ArqOutSOFM For Output As #1
    Case 3
        Open ArqOutSPT For Output As #1
End Select

Print #1, "Rafaela & Lêlê"
Print #1,
Print #1,
Print #1, "Sequencia:"
Print #1,
Print #1,
Print #1, "id";
Print #1, Tab(10); "i";
Print #1, Tab(20); "p";
Print #1, Tab(30); "C";
Print #1, Tab(40); "d";
Print #1, Tab(50); "L";
Print #1, Tab(60); "T";
Print #1, Tab(70); "w"
Print #1,
Select Case opcao
    Case 1
        For j = 0 To (N_OPs - 1)
            Print #1, Vet_OPs_EDD(j).id;
            Print #1, Tab(10); Vet_OPs_EDD(j).i;
            Print #1, Tab(20); Vet_OPs_EDD(j).p;
            Print #1, Tab(30); Vet_OPs_EDD(j).C;
            Print #1, Tab(40); Vet_OPs_EDD(j).d;
            Print #1, Tab(50); Vet_OPs_EDD(j).L;
            Print #1, Tab(60); Vet_OPs_EDD(j).T;
            Print #1, Tab(70); Format(Vet_OPs_EDD(j).w, "#0.000");
            Print #1,
        Next j
        Print #1, "Cmax: "; Schd_EDD.Cmax;
        Print #1,
        Print #1, "Lavg: "; Format(Schd_EDD.Lavg, "#0.000");
        Print #1,
        Print #1, "Lmax: "; Schd_EDD.Lmax;
        Print #1,
        Print #1, "Ltot: "; Schd_EDD.Ltot;
        Print #1,
        Print #1, "N: "; Schd_EDD.Ntot;
        Print #1,
        Print #1, "Nwt: "; Format(Schd_EDD.Nwt, "#0.000");
        Print #1,
        Print #1, "Tmax: "; Schd_EDD.Tmax;
        Print #1,
        Print #1, "Ttot: "; Schd_EDD.Ttot;
        Print #1,
        Print #1, "Twt: "; Format(Schd_EDD.Twt, "#0.000");
        Print #1,

    Case 2
        For j = 0 To (N_OPs - 1)
            Print #1, Vet_OPs_SOFM(j).id;
            Print #1, Tab(10); Vet_OPs_SOFM(j).i;
            Print #1, Tab(20); Vet_OPs_SOFM(j).p;
            Print #1, Tab(30); Vet_OPs_SOFM(j).C;
            Print #1, Tab(40); Vet_OPs_SOFM(j).d;
            Print #1, Tab(50); Vet_OPs_SOFM(j).L;
            Print #1, Tab(60); Vet_OPs_SOFM(j).T;
            Print #1, Tab(70); Format(Vet_OPs_SOFM(j).w, "#0.000");
            Print #1,
        Next j
        Print #1, "Cmax: "; Schd_SOFM.Cmax;
        Print #1,
        Print #1, "Lavg: "; Format(Schd_SOFM.Lavg, "#0.000");
        Print #1,
        Print #1, "Lmax: "; Schd_SOFM.Lmax;
        Print #1,
        Print #1, "Ltot: "; Schd_SOFM.Ltot;
        Print #1,
        Print #1, "N: "; Schd_SOFM.Ntot;
        Print #1,
        Print #1, "Nwt: "; Format(Schd_SOFM.Nwt, "#0.000");
        Print #1,
        Print #1, "Tmax: "; Schd_SOFM.Tmax;
        Print #1,
        Print #1, "Ttot: "; Schd_SOFM.Ttot;
        Print #1,
        Print #1, "Twt: "; Format(Schd_SOFM.Twt, "#0.000");
        Print #1,
        Print #1,
```

```
            Print #1,
            For s = 0 To (M - 1)
                For k = 0 To (N - 1)
                    Print #1, Spc(2); Format(Mat_W(k, s), "#0.000000");
                Next k
                Print #1,
            Next s
        Case 3
            For j = 0 To (N_OPs - 1)
                Print #1, Vet_OPs_SPT(j).id;
                Print #1, Tab(10); Vet_OPs_SPT(j).i;
                Print #1, Tab(20); Vet_OPs_SPT(j).p;
                Print #1, Tab(30); Vet_OPs_SPT(j).C;
                Print #1, Tab(40); Vet_OPs_SPT(j).d;
                Print #1, Tab(50); Vet_OPs_SPT(j).L;
                Print #1, Tab(60); Vet_OPs_SPT(j).T;
                Print #1, Tab(70); Format(Vet_OPs_SPT(j).w, "#0.000");
                Print #1,
            Next j
            Print #1, "Cmax: "; Schd_SPT.Cmax;
            Print #1,
            Print #1, "Lavg: "; Format(Schd_SPT.Lavg, "#0.000");
            Print #1,
            Print #1, "Lmax: "; Schd_SPT.Lmax;
            Print #1,
            Print #1, "Ltot: "; Schd_SPT.Ltot;
            Print #1,
            Print #1, "N: "; Schd_SPT.Ntot;
            Print #1,
            Print #1, "Nwt: "; Format(Schd_SPT.Nwt, "#0.000");
            Print #1,
            Print #1, "Tmax: "; Schd_SPT.Tmax;
            Print #1,
            Print #1, "Ttot: "; Schd_SPT.Ttot;
            Print #1,
            Print #1, "Twt: "; Format(Schd_SPT.Twt, "#0.000");
            Print #1,
    End Select

    Close (1)

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Inicializa_Pesos_SOFM                     *
'*                                                   *
'*****************************************************

Public Sub Inicializa_Pesos_SOFM()
    Dim k, s As Integer

    For s = 0 To (N - 1)
        For k = 0 To (M - 1)
            Randomize
            '23Nov00 troquei k por s
            Mat_W(k, s) = Rnd
        Next k
    Next s

    For k = 1 To (M - 1)
        Soma = 0
        For s = 0 To (N - 1)
            Soma = Soma + Mat_W(k, s)
        Next s
        For s = 0 To (N - 1)
            Mat_W(k, s) = Mat_W(k, s) / Soma
        Next s
    Next k

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Inicializa_Vetores                        *
'*                                                   *
'*****************************************************

Public Sub Inicializa_Vetores()
    Dim j As Integer
    Dim k As Integer

    For j = 0 To c_OPs_MAX
        Vet_OPs(j).Status = 0
        Vet_OPs(j).C = 0
```

```
            Vet_OPs(j).id = -1
            Vet_OPs(j).d = 0
            Vet_OPs(j).L = 0
            Vet_OPs(j).p = 0
            Vet_OPs(j).p = 0
            Vet_OPs(j).r = 0
            Vet_OPs(j).i = 0
            Vet_OPs(j).T = 0
            Vet_OPs(j).Tw = 0
            Vet_OPs(j).w = 0
            Vet_OPs(j).z = 0
        Next j

        For j = 0 To c_OPs_MAX
            Vet_OPs_EDD(j).Status = 0
            Vet_OPs_EDD(j).C = 0
            Vet_OPs_EDD(j).id = -1
            Vet_OPs_EDD(j).d = 0
            Vet_OPs_EDD(j).L = 0
            Vet_OPs_EDD(j).p = 0
            Vet_OPs_EDD(j).p = 0
            Vet_OPs_EDD(j).r = 0
            Vet_OPs_EDD(j).i = 0
            Vet_OPs_EDD(j).T = 0
            Vet_OPs_EDD(j).Tw = 0
            Vet_OPs_EDD(j).w = 0
            Vet_OPs_EDD(j).z = 0
        Next j

        For j = 0 To c_OPs_MAX
            Vet_OPs_SOFM(j).Status = 0
            Vet_OPs_SOFM(j).C = 0
            Vet_OPs_SOFM(j).id = -1
            Vet_OPs_SOFM(j).d = 0
            Vet_OPs_SOFM(j).L = 0
            Vet_OPs_SOFM(j).p = 0
            Vet_OPs_SOFM(j).p = 0
            Vet_OPs_SOFM(j).r = 0
            Vet_OPs_SOFM(j).i = 0
            Vet_OPs_SOFM(j).T = 0
            Vet_OPs_SOFM(j).Tw = 0
            Vet_OPs_SOFM(j).w = 0
            Vet_OPs_SOFM(j).z = 0
        Next j

        For j = 0 To c_OPs_MAX
            Vet_OPs_SPT(j).Status = 0
            Vet_OPs_SPT(j).C = 0
            Vet_OPs_SPT(j).id = -1
            Vet_OPs_SPT(j).d = 0
            Vet_OPs_SPT(j).L = 0
            Vet_OPs_SPT(j).p = 0
            Vet_OPs_SPT(j).p = 0
            Vet_OPs_SPT(j).r = 0
            Vet_OPs_SPT(j).i = 0
            Vet_OPs_SPT(j).T = 0
            Vet_OPs_SPT(j).Tw = 0
            Vet_OPs_SPT(j).w = 0
            Vet_OPs_SPT(j).z = 0
        Next j

    End Sub

    '*****************************************************
    '*                                                   *
    '*  FUNÇÃO Le_Arquivo_OP                             *
    '*                                                   *
    '*****************************************************

    Public Sub Le_Arquivo_OP()

        Inicializa_Vetores
        frmPrincipal.cdbArquivo.ShowOpen
        ArqInOp = frmPrincipal.cdbArquivo.FileName

        Open ArqInOp For Input As #1

        N_OPs = 0

        Do While Not EOF(1)
            Input #1, Vet_OPs(N_OPs).id, Vet_OPs(N_OPs).p, Vet_OPs(N_OPs).r,
    Vet_OPs(N_OPs).d, Vet_OPs(N_OPs).z
            N_OPs = N_OPs + 1
        Loop
```

```
    Close 1

    intMsg = MsgBox("Arquivo de OPs lido com sucesso", vbInformation, "Sistema Coelet")

    Normaliza_Pesos_OPs

    Cria_Nomes_ArqOut

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Normaliza_Pesos_OPs                       *
'*                                                   *
'*****************************************************

Public Sub Normaliza_Pesos_OPs()

    Dim j   As Integer
    Dim aux As Double

    aux = 0

    For j = 0 To (N_OPs - 1)
        aux = aux + Vet_OPs(j).z
    Next j

    For j = 0 To (N_OPs - 1)
        Vet_OPs(j).w = Vet_OPs(j).z / aux
    Next j

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Schedule_EDD                              *
'*                                                   *
'*****************************************************

Public Sub Schedule_EDD()
    Dim aux             As Tipo_OP
    Dim d1, d2, j, pass As Integer
    Dim troca           As Boolean

    For j = 0 To (N_OPs - 1)
        Vet_OPs_EDD(j) = Vet_OPs(j)
    Next j

    d1 = 0
    d2 = 0
    pass = 0
    troca = True

    While ((pass <= (N_OPs - 1)) And (troca))
        troca = False
        For j = 0 To (N_OPs - pass - 2)
            d1 = Vet_OPs_EDD(j).d
            d2 = Vet_OPs_EDD(j + 1).d
            If (d1 > d2) Then
                troca = True
                aux = Vet_OPs_EDD(j)
                Vet_OPs_EDD(j) = Vet_OPs_EDD(j + 1)
                Vet_OPs_EDD(j + 1) = aux
            End If
        Next j
        pass = pass + 1
    Wend

    Aloca_OPs (1)
    Cal_SCHD (1)
    Imprime_Resultado (1)

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Schedule_SOFM2                            *
'*                                                   *
'*****************************************************

Public Sub Schedule_SOFM2(Beta_Inicial, Sigma As Double)

    Dim Alfa, Beta          As Double
    Dim Delta_W             As Double
    Dim Delta_W_Max         As Double
```

```
Dim Delta_W_Aux          As Double
Dim L_aux                As Integer
Dim OP_aux               As Tipo_OP
Dim epoca                As Integer
Dim k, s                 As Integer
Dim T_man                As Integer
Dim Vet_m(c_ROW_MAX)     As Integer
Dim Vet_V(c_ROW_MAX)     As Double
Dim A_aux, B_aux, C_aux, D_aux, E_aux, F_aux, G_aux, H_aux    As Double
Dim p1, p2, pass As Integer
Dim troca          As Boolean
Dim aux As Integer
Dim Termina     As Boolean
Dim Covergiu    As Boolean
Dim Soma     As Double
Dim Memoria As Integer
Dim W_aux    As Double
Dim Nwt      As Double
Dim Twt      As Double
Dim Qualidade   As Double
Dim pos, kk As Integer
Dim Vaux     As Double
Dim X_aux As Integer
Dim Vet_X(c_ROW_MAX)     As Integer
Dim Vet_X_aux(c_ROW_MAX)     As Integer


'***** Calcula os Valores de M e N *****

Calcula_MN


'***** Inicializa os pesos da rede (Inicio) *****

Inicializa_Pesos_SOFM

'***** Inicializa o valor da vizinhança *****

Ni = (N - 1)

Beta = Beta_Inicial


'***** Ordena as ordens na sequencia EDD (Inicio) *****

For j = 0 To (N_OPs - 1)
    Vet_X(j) = j
Next j

d1 = 0
d2 = 0
pass = 0
troca = True

While ((pass <= (N_OPs - 1)) And (troca))
    troca = False
    For j = 0 To (N_OPs - pass - 2)
        d1 = Vet_OPs(j).d
        d2 = Vet_OPs(j + 1).d
        If (d1 > d2) Then
            troca = True
            X_aux = Vet_X(j)
            Vet_X(j) = Vet_X(j + 1)
            Vet_X(j + 1) = X_aux
        End If
    Next j
    pass = pass + 1
Wend

'***** Ordena as ordens na sequencia EDD (Fim) *****

While (Ni >= 1) '***** Loop Vizinhança (Ni) - Inicio *****


    '***** Cria uma sequência aleatória (Inicio) *****

    If (Ni < (N - 1)) Then
        For j = 0 To (N - 1)
            Vet_X_aux(j) = Vet_X(j)
        Next j

        For k = 0 To (N - 1)
            Randomize
            pos = Int(((N - k) - 1) * Rnd)
            Vet_X(k) = Vet_X_aux(pos)
```

```
            For kk = pos To (N - k - 1)
                Vet_X_aux(kk) = Vet_X_aux(kk + 1)
            Next kk
        Next k
    End If

    '***** Cria uma sequência de ordens aleatória (Fim) *****

    '***** Inicializa variaveis de controle *****

    Termina = False
    epoca = 0
    Delta_W_Max = 0


    While (Not (Termina)) '***** Loop epoca (Termina) -> Inicio *****

        For j = 0 To (N - 1) '***** Loop de ordens (j) -> Inicio *****

            '***** Troca as ordens de posição duas-a-duas *****

            If (j > 0) Then
                X_aux = Vet_X(j - 1)
                Vet_X(j - 1) = Vet_X(j)
                Vet_X(j) = X_aux
            End If

            '***** Avalia a sequencia obtida e calcula os valores de saida V *****

            C_aux = 0
            L_aux = 0
            Vet_V(j) = 0
            For jj = 0 To (N - 1) '***** Loop de posicoes (jj) -> Inicio *****
                C_aux = C_aux + Vet_OPs(Vet_X(jj)).p
                L_aux = C_aux - Vet_OPs(Vet_X(jj)).d
                If (L_aux <= 0) Then
                    Vet_V(j) = Vet_V(j) + (Vet_OPs(Vet_X(jj)).w) * (Mat_W(Vet_X(jj),
jj))
                End If
            Next jj '***** Loop de posicoes (jj) -> Fim *****

            '***** Destroca as ordens de posição duas-a-duas *****

            If (j > 0) Then
                X_aux = Vet_X(j - 1)
                Vet_X(j - 1) = Vet_X(j)
                Vet_X(j) = X_aux
            End If

        Next j '***** Loop de ordens (j) -> Fim *****

        '***** Ordena o vetor m (Inicio) *****

        For s = 0 To (N - 1)
            Vet_m(s) = s
        Next s

        p1 = 0
        p2 = 0
        pass = 0
        troca = True

        While ((pass <= (N - 1)) And (troca))
            troca = False
            For s = 0 To (N - pass - 2)
                p1 = Vet_V(Vet_m(s))
                p2 = Vet_V(Vet_m(s + 1))
                If (p1 > p2) Then
                    troca = True
                    aux = Vet_m(s)
                    Vet_m(s) = Vet_m(s + 1)
                    Vet_m(s + 1) = aux
                End If
            Next s
            pass = pass + 1
        Wend

        '***** Ordena o vetor m (Fim) *****

        '***** Atualiza os pesos (Inicio) *****


        For Nii = 0 To (N - 1) '***** Loop de vizinhança (Nii) -> Inicio *****
```

```
            '***** Dentro da vizinhança *****

            If (Nii <= Ni) Then
                A_aux = Vet_V(Vet_m(0))
                B_aux = Vet_V(Vet_m(Nii))
                C_aux = Vet_V(Vet_m(N - 1))
                D_aux = A_aux - B_aux
                E_aux = A_aux - C_aux
                F_aux = -(D_aux / E_aux)
                Alfa = Beta * Exp(F_aux)

                '***** Re-cria a sequencia *****

                If (Vet_m(Nii) > 0) Then
                    X_aux = Vet_X(Vet_m(Nii) - 1)
                    Vet_X(Vet_m(Nii) - 1) = Vet_X(Vet_m(Nii))
                    Vet_X(Vet_m(Nii)) = X_aux
                End If

                '***** Atualiza os pesos *****

                For jj = 0 To (N - 1)
                    G_aux = Mat_W(Vet_X(jj), jj)
                    H_aux = 1 - G_aux
                    Delta_W = Alfa * H_aux
                    Mat_W(Vet_X(jj), jj) = Mat_W(Vet_X(jj), jj) + Delta_W
                Next jj

                '***** Desfaz a sequencia *****

                If (Vet_m(Nii) > 0) Then
                    X_aux = Vet_X(Vet_m(Nii) - 1)
                    Vet_X(Vet_m(Nii) - 1) = Vet_X(Vet_m(Nii))
                    Vet_X(Vet_m(Nii)) = X_aux
                End If
            '***** Fora da vizinhança *****

            Else

                '***** Re-cria a sequencia *****

                If (Vet_m(Nii) > 0) Then
                    X_aux = Vet_X(Vet_m(Nii) - 1)
                    Vet_X(Vet_m(Nii) - 1) = Vet_X(Vet_m(Nii))
                    Vet_X(Vet_m(Nii)) = X_aux
                End If

                '***** Atualiza os pesos *****

                For jj = 0 To (N - 1)
                    G_aux = Mat_W(Vet_X(jj), jj)
                    Delta_W = -Beta * G_aux
                    Mat_W(Vet_X(jj), jj) = Mat_W(Vet_X(jj), jj) + Delta_W
                Next jj

                '***** Desfaz a sequencia *****

                If (Vet_m(Nii) > 0) Then
                    X_aux = Vet_X(Vet_m(Nii) - 1)
                    Vet_X(Vet_m(Nii) - 1) = Vet_X(Vet_m(Nii))
                    Vet_X(Vet_m(Nii)) = X_aux
                End If
            End If

            Delta_W_Aux = Abs(Delta_W)
            If (Delta_W_Aux > Delta_W_Max) Then
                Delta_W_Max = Delta_W_Aux
            End If

        Next Nii '***** Loop de vizinhança (Nii) -> Fim *****

        '***** Atualiza os pesos (Fim) *****

        '***** Normaliza os pesos (Inicio) *****

        For k = 0 To (N - 1)
            Soma = 0
            For s = 0 To (N - 1)
                Soma = Soma + Mat_W(k, s)
            Next s

            For s = 0 To (N - 1)
                Mat_W(k, s) = Mat_W(k, s) / Soma
            Next s
        Next k
```

```
            '***** Normaliza os pesos (Fim) *****

            '***** Refaz a sequencia vencedora *****

            If (Vet_m(0) > 0) Then
                X_aux = Vet_X(Vet_m(0) - 1)
                Vet_X(Vet_m(0) - 1) = Vet_X(Vet_m(0))
                Vet_X(Vet_m(0)) = X_aux
            End If

            '***** Atualiza variáveis de controle *****

            epoca = epoca + 1

            If ((epoca = c_Epoca_MAX) Or (Delta_W_Max < c_Epsilon)) Then
                Termina = True
            End If

            Beta = Beta * Sigma

        Wend '***** Loop epoca (Termina) -> Fim *****

        '***** Atualiza variáveis *****

        Ni = Ni - 1

        Rem Beta = Beta * Sigma

    Wend '***** Loop Vizinhança (Ni) -> Fim *****

    '***** Cria a sequencia (Inicio) *****

    For j = 0 To c_OPs_MAX
        Vet_OPs_SOFM(j).Status = 0
        Vet_OPs_SOFM(j).C = 0
        Vet_OPs_SOFM(j).id = -1
        Vet_OPs_SOFM(j).d = 0
        Vet_OPs_SOFM(j).L = 0
        Vet_OPs_SOFM(j).p = 0
        Vet_OPs_SOFM(j).p = 0
        Vet_OPs_SOFM(j).r = 0
        Vet_OPs_SOFM(j).i = 0
        Vet_OPs_SOFM(j).T = 0
        Vet_OPs_SOFM(j).Tw = 0
        Vet_OPs_SOFM(j).w = 0
        Vet_OPs_SOFM(j).z = 0
    Next j

    For s = 0 To (M - 1)
        W_aux = 0
        aux = 0
        For k = 0 To (N - 1)
            If (Mat_W(k, s) > W_aux) Then
                W_aux = Mat_W(k, s)
                aux = k
            End If
        Next k
        If (Vet_OPs(aux).Status = 0) Then
            Vet_OPs(aux).Status = 1
            Vet_OPs_SOFM(s) = Vet_OPs(aux)
        Else
            k = 0
            Termina = False
            '23Nov00 **** (N - 1)
            While ((Not (Termina)) And (k < (N - 1)))
                'If ((Mat_W(k, s) > c_Epsilon) And (Vet_OPs_SOFM(s).id = -1)) Then
                If (Vet_OPs(k).Status = 0) Then
                    Vet_OPs(k).Status = 1
                    Vet_OPs_SOFM(s) = Vet_OPs(k)
                    Termina = True
                End If
                k = k + 1
            Wend
        End If
    Next s

    '***** Cria a sequencia (Fim) *****

    Aloca_OPs (2)
    Cal_SCHD (2)
    Imprime_Resultado (2)

End Sub
```

```
'*****************************************************
'*                                                   *
'*  FUNÇÃO Schedule_SPT                              *
'*                                                   *
'*****************************************************

Public Sub Schedule_SPT()
    Dim aux             As Tipo_OP
    Dim p1, p2, j, pass As Integer
    Dim troca           As Boolean

    For j = 0 To (N_OPs - 1)
        Vet_OPs_SPT(j) = Vet_OPs(j)
    Next j

    p1 = 0
    p2 = 0
    pass = 0
    troca = True

    While ((pass <= (N_OPs - 1)) And (troca))
        troca = False
        For j = 0 To (N_OPs - pass - 2)
            p1 = Vet_OPs_SPT(j).p
            p2 = Vet_OPs_SPT(j + 1).p
            If (p1 > p2) Then
                troca = True
                aux = Vet_OPs_SPT(j)
                Vet_OPs_SPT(j) = Vet_OPs_SPT(j + 1)
                Vet_OPs_SPT(j + 1) = aux
            End If
        Next j
        pass = pass + 1
    Wend

    Aloca_OPs (3)
    Cal_SCHD (3)
    Imprime_Resultado (3)

End Sub

'*****************************************************
'*                                                   *
'*  FUNÇÃO Simula                                    *
'*                                                   *
'*****************************************************

Public Sub Simula()
    Dim cont As Integer
    Dim aux As String
    Dim tam As Integer
    Dim j As Integer
    Dim Sig As Double
    Dim Bet As Double


    Open ArqOutSim For Output As #2

    tam = Len(ArqOutSOFM)
    aux = Left(ArqOutSOFM, (tam - 4))

        Print #2, "Rafaela & Lêlê"
        Print #2,
        Print #2, "Método"; Tab(20); "Cmax"; Tab(30); "Lavg"; Tab(40); "Lmax"; Tab(50);
"Ltot"; Tab(60); "Ntot"; Tab(70); "Nwt"; Tab(80); "Tmax"; Tab(90); "Ttot"; Tab(100);
"Twt";
        Print #2,
        Print #2,

        Schedule_EDD
        Print #2, "EDD"; Tab(20); Schd_EDD.Cmax; Tab(30); Format(Schd_EDD.Lavg,
"#0.000"); Tab(40); Schd_EDD.Lmax; Tab(50); Schd_EDD.Ltot; Tab(60); Schd_EDD.Ntot;
Tab(70); Format(Schd_EDD.Nwt, "#0.000"); Tab(80); Schd_EDD.Tmax; Tab(90); Schd_EDD.Ttot;
Tab(100); Format(Schd_EDD.Twt, "#0.000#")
        Print #2,

        Schedule_SPT
        Print #2, "SPT"; Tab(20); Schd_SPT.Cmax; Tab(30); Format(Schd_SPT.Lavg,
"#0.000"); Tab(40); Schd_SPT.Lmax; Tab(50); Schd_SPT.Ltot; Tab(60); Schd_SPT.Ntot;
Tab(70); Format(Schd_SPT.Nwt, "#0.000"); Tab(80); Schd_SPT.Tmax; Tab(90); Schd_SPT.Ttot;
Tab(100); Format(Schd_SPT.Twt, "#0.000#")
        Print #2,
```

```
    For cont = 1 To c_Sim_MAX
        For j = 0 To c_OPs_MAX
            Vet_OPs(j).Status = 0
            Vet_OPs_SOFM(j).Status = 0
            Vet_OPs_SOFM(j).C = 0
            Vet_OPs_SOFM(j).id = -1
            Vet_OPs_SOFM(j).d = 0
            Vet_OPs_SOFM(j).L = 0
            Vet_OPs_SOFM(j).p = 0
            Vet_OPs_SOFM(j).p = 0
            Vet_OPs_SOFM(j).r = 0
            Vet_OPs_SOFM(j).i = 0
            Vet_OPs_SOFM(j).T = 0
            Vet_OPs_SOFM(j).Tw = 0
            Vet_OPs_SOFM(j).w = 0
            Vet_OPs_SOFM(j).z = 0
        Next j
        ArqOutSOFM = aux + "SOFM" + Format(cont) + ".txt"
        Bet = c_Beta_Inicial
        Sig = c_Sigma
        Schedule_SOFM2 Bet, Sig
        Print #2, "SOFM"; Tab(20); Schd_SOFM.Cmax; Tab(30); Format(Schd_SOFM.Lavg,
"#0.000"); Tab(40); Schd_SOFM.Lmax; Tab(50); Schd_SOFM.Ltot; Tab(60); Schd_SOFM.Ntot;
Tab(70); Format(Schd_SOFM.Nwt, "#0.000"); Tab(80); Schd_SOFM.Tmax; Tab(90);
Schd_SOFM.Ttot; Tab(100); Format(Schd_SOFM.Twt, "#0.000#")
        Print #2,
     Next cont

    Close (2)

    intMsg = MsgBox("Simulação realizada com sucesso", vbInformation, "Sistema Coelet")

End Sub

'*************************************************
'*                                               *
'*  FUNÇÃO Simula_Beta_Inicial                   *
'*                                               *
'*************************************************

Public Sub Simula_Beta_Inicial()
    Dim cont As Integer
    Dim aux As String
    Dim tam As Integer
    Dim j As Integer
    Dim Bet As Double
    Dim Sig As Double


    For Bet = 0.05 To 1 Step (c_Beta_Inicial_Step)

        Cria_Nomes_ArqOut

        tam = Len(ArqOutSim)
        aux = Left(ArqOutSim, (tam - 4))

        ArqOutSim = aux + "BetaInicial=" + Format(Bet, "#0.000") + ".txt"

        Open ArqOutSim For Output As #2

        tam = Len(ArqOutSOFM)
        aux = Left(ArqOutSOFM, (tam - 4))


        Print #2, "Rafaela & Lêlê"
        Print #2,
        Print #2, "Método"; Tab(20); "Cmax"; Tab(30); "Lavg"; Tab(40); "Lmax"; Tab(50);
"Ltot"; Tab(60); "Ntot"; Tab(70); "Nwt"; Tab(80); "Tmax"; Tab(90); "Ttot"; Tab(100);
"Twt";
        Print #2,
        Print #2,

        Schedule_EDD
        Print #2, "EDD"; Tab(20); Schd_EDD.Cmax; Tab(30); Format(Schd_EDD.Lavg,
"#0.000"); Tab(40); Schd_EDD.Lmax; Tab(50); Schd_EDD.Ltot; Tab(60); Schd_EDD.Ntot;
Tab(70); Format(Schd_EDD.Nwt, "#0.000"); Tab(80); Schd_EDD.Tmax; Tab(90); Schd_EDD.Ttot;
Tab(100); Format(Schd_EDD.Twt, "#0.000#")
        Print #2,

        Schedule_SPT
        Print #2, "SPT"; Tab(20); Schd_SPT.Cmax; Tab(30); Format(Schd_SPT.Lavg,
"#0.000"); Tab(40); Schd_SPT.Lmax; Tab(50); Schd_SPT.Ltot; Tab(60); Schd_SPT.Ntot;
Tab(70); Format(Schd_SPT.Nwt, "#0.000"); Tab(80); Schd_SPT.Tmax; Tab(90); Schd_SPT.Ttot;
Tab(100); Format(Schd_SPT.Twt, "#0.000#")
```

```
        Print #2,

        For cont = 1 To c_Sim_MAX
            For j = 0 To c_OPs_MAX
                Vet_OPs(j).Status = 0
                Vet_OPs_SOFM(j).Status = 0
                Vet_OPs_SOFM(j).C = 0
                Vet_OPs_SOFM(j).id = -1
                Vet_OPs_SOFM(j).d = 0
                Vet_OPs_SOFM(j).L = 0
                Vet_OPs_SOFM(j).p = 0
                Vet_OPs_SOFM(j).p = 0
                Vet_OPs_SOFM(j).r = 0
                Vet_OPs_SOFM(j).i = 0
                Vet_OPs_SOFM(j).T = 0
                Vet_OPs_SOFM(j).Tw = 0
                Vet_OPs_SOFM(j).w = 0
                Vet_OPs_SOFM(j).z = 0
            Next j
            ArqOutSOFM = aux + "SOFM" + Format(cont) + ".txt"
            Sig = c_Sigma
            Schedule_SOFM2 Bet, Sig
            Print #2, "SOFM"; Tab(20); Schd_SOFM.Cmax; Tab(30); Format(Schd_SOFM.Lavg,
"#0.000"); Tab(40); Schd_SOFM.Lmax; Tab(50); Schd_SOFM.Ltot; Tab(60); Schd_SOFM.Ntot;
Tab(70); Format(Schd_SOFM.Nwt, "#0.000"); Tab(80); Schd_SOFM.Tmax; Tab(90);
Schd_SOFM.Ttot; Tab(100); Format(Schd_SOFM.Twt, "#0.000#")
            Print #2,
        Next cont

        Close (2)

    Next Bet

    intMsg = MsgBox("Simulação realizada com sucesso", vbInformation, "Sistema Coelet")

End Sub




'*****************************************************
'*                                                   *
'*  FUNÇÃO Simula_Sigma                              *
'*                                                   *
'*****************************************************

Public Sub Simula_Sigma()
    Dim cont As Integer
    Dim aux As String
    Dim tam As Integer
    Dim j As Integer
    Dim Sig As Double


    For Sig = 0.05 To 1 Step (c_Sigma_Step)

        Cria_Nomes_ArqOut

        tam = Len(ArqOutSim)
        aux = Left(ArqOutSim, (tam - 4))

        ArqOutSim = aux + Format(Sig, "#0.000") + ".txt"

        Open ArqOutSim For Output As #2

        tam = Len(ArqOutSOFM)
        aux = Left(ArqOutSOFM, (tam - 4))


        Print #2, "Rafaela & Lêlê"
        Print #2,
        Print #2, "Método"; Tab(20); "Cmax"; Tab(30); "Lavg"; Tab(40); "Lmax"; Tab(50);
"Ltot"; Tab(60); "Ntot"; Tab(70); "Nwt"; Tab(80); "Tmax"; Tab(90); "Ttot"; Tab(100);
"Twt";
        Print #2,
        Print #2,

        Schedule_EDD
        Print #2, "EDD"; Tab(20); Schd_EDD.Cmax; Tab(30); Format(Schd_EDD.Lavg,
"#0.000"); Tab(40); Schd_EDD.Lmax; Tab(50); Schd_EDD.Ltot; Tab(60); Schd_EDD.Ntot;
Tab(70); Format(Schd_EDD.Nwt, "#0.000"); Tab(80); Schd_EDD.Tmax; Tab(90); Schd_EDD.Ttot;
Tab(100); Format(Schd_EDD.Twt, "#0.000#")
        Print #2,
```

```
        Schedule_SPT
        Print #2, "SPT"; Tab(20); Schd_SPT.Cmax; Tab(30); Format(Schd_SPT.Lavg,
"#0.000"); Tab(40); Schd_SPT.Lmax; Tab(50); Schd_SPT.Ltot; Tab(60); Schd_SPT.Ntot;
Tab(70); Format(Schd_SPT.Nwt, "#0.000"); Tab(80); Schd_SPT.Tmax; Tab(90); Schd_SPT.Ttot;
Tab(100); Format(Schd_SPT.Twt, "#0.000#")
        Print #2,

        For cont = 1 To c_Sim_MAX
            For j = 0 To c_OPs_MAX
                Vet_OPs(j).Status = 0
                Vet_OPs_SOFM(j).Status = 0
                Vet_OPs_SOFM(j).C = 0
                Vet_OPs_SOFM(j).id = -1
                Vet_OPs_SOFM(j).d = 0
                Vet_OPs_SOFM(j).L = 0
                Vet_OPs_SOFM(j).p = 0
                Vet_OPs_SOFM(j).p = 0
                Vet_OPs_SOFM(j).r = 0
                Vet_OPs_SOFM(j).i = 0
                Vet_OPs_SOFM(j).T = 0
                Vet_OPs_SOFM(j).Tw = 0
                Vet_OPs_SOFM(j).w = 0
                Vet_OPs_SOFM(j).z = 0
            Next j
            ArgOutSOFM = aux + "SOFM" + Format(cont) + ".txt"
            Schedule_SOFM2 (Sig)
            Print #2, "SOFM"; Tab(20); Schd_SOFM.Cmax; Tab(30); Format(Schd_SOFM.Lavg,
"#0.000"); Tab(40); Schd_SOFM.Lmax; Tab(50); Schd_SOFM.Ltot; Tab(60); Schd_SOFM.Ntot;
Tab(70); Format(Schd_SOFM.Nwt, "#0.000"); Tab(80); Schd_SOFM.Tmax; Tab(90);
Schd_SOFM.Ttot; Tab(100); Format(Schd_SOFM.Twt, "#0.000#")
            Print #2,
        Next cont

        Close (2)

    Next Sig

    intMsg = MsgBox("Simulação realizada com sucesso", vbInformation, "Sistema Coelet")

End Sub
```

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, J.; BALAS, E.; ZAWACK, D. The Shifting Bottleneck Procedure for Job Shop Scheduling. **Management Science**, v.34, p.391-401, 1988.

ANTHONY, M.T.; SCHAFFER, J. **Computers, Chess and Cognition**. New York, Springer-Verlag, 1990.

BAKER, K. P. **Introduction to Sequencing and Scheduling**. New Yorke, John Wiley & Sons, 1974.

BAKER, K. P. **Elements of Sequencing and Scheduling**. 1997.

BENSANA, E.; BELL, G.; DUBOIS, D. OPAL: A Multi-Knowledge-Based System for Industrial Job-Shop Scheduling. **International Journal of Production Research**, v.26, p.795-819, 1988.

BOSE, N. K.; LIANG, P. **Neural Networks Fundamentals with Graphs, Algorithms and Applications**. McGraw-Hill, Inc., 1996.

BURKE, L. ; IGNIZIO, J.P. Neural Networks and Operations Research – An Overview. **Computer Operations Research**, v.19, p.179-89, 1992.

BUFFA, E.S. **Modern Operations Management: a Course in day to day Operations**. New York, John Wiley and Sons, 1975.

CALFAT, R. A. **Um Sistema Especialista de Apoio na Seleção de Técnicas de Sequenciação**. São Paulo, 1990. 369p. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo.

CHESTER, M. **Neural Networks: a Tutorial**. PTR Prentice Hall, 1993.

CONWAY, R.W.; MAXWELL, W.L.; MILLER, L.W. **Theory Of Scheduling**, Addison-Wesley Publishing Company, 1967.

CRESCENZI, P.; KANN., V. **A Compendium of NP Optimization Problems**. http://www.nada.kth.se/~viggo/problemlist/compendium.html ,1998.

CYTED - 1as JORNADAS CENTROAMERICANAS DE INFORMATICA Y AUTOMATICA, San Jose de Costa Rica, Setembro de 1994. Programa Iberoamericano de Ciencia y Tecnologia para el Desarrollo (CYTED).

DAMASIO, A. R. **Descarte's Error.** Avon Books, 1994.

EBERHART, R.C.; DOBBINS, R.W. **Neural Network PC Tools: A Practical Guide**. Academic Press, Inc., 1990.

EL GHAZIRI, H. **Solving Routing Problems by a Self-Organizing Map**. In: Kohonen, T. et al (Eds.) Artificial Neural Networks, v.1, p.829-34, 1991.

FALKENAUER, E.; BOUFFOIX, S. A Genetic Algorithm for Job Shops. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1991. **Proceedings**. 1991.

FAVATA, F.; WALKER, R. A Study of the Application of Kohonen-type Neural Networks to the Travelling Salesman Problem. **Biological Cybernetics**, v.64, p.463-68, 1991.

FORT, J. C. Solving a combinatorial problem via Self-Organizing Process: An application of the Kohonen algorithm to the Travelling Salesman Problem. **Biological Cybernetics**, v.59, p.33-40, 1988.

FOX, M. S.; SMITH, S. F. **ISIS – A Knowledge-Based System for Factory Scheduling**. **Expert Systems**, v.1, p.25-49, 1984.

FOX, M.S.; ZWEBEN, M. **Intelligent Scheduling**. Morgan Kaufmann Publishers, 1994.

FREEMAN, J.A.; SKAPURA, D.M. **Neural Networks: Algorithms, Applications and Programming Techniques**. Addison-Wesley, 1992.

GLOVER, F. Tabu Search: A Tutorial. **Interfaces**, v.20, n.4, p.74-94, 1990.

GROSSBERG, S.; CARPENTER, G.A. A massively parallel architecture for a self-organizing neural pattern recognition machine. **Computer Vision, Graphics and Image Processing**, v.37, p.54-115, 1987.

GUERRERO, F.; LOZANO, S.; CANCA D.; SMITH, K.A. Machine Grouping in Cellular manufacturing: A Self-Organizing Neural Network. In: A. B. Bulsari et al (Eds.) Engineering Benefits from Neural Networks: 4th International Conference on Engineering Applications of Neural Networks, Systems Engineering Association, Turku, Finland, 1998. **Proceedings**, p.374-77.

GUERRERO, F.; LOZANO, S; SMITH, K.A.; KWOK, T. Manufacturing Cell Formation Using a New Self-Organizing Neural Network. In: International Conference on Computers and Industrial Engineering, 1999. **Proceedings**, v.1, p.668-72.

HAX, A.C; CANDEA, D. **Production and Inventory Management**. Englewood Cliffs, New Jersey, Prentice Hall, 1984.

HOPFIELD, J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. **Proceedings National Academy of Sciences**, v.79, p. 2554-58, 1982.

HOPFIELD, J. Neurons with Graded Response Have Collective Computational Properties Like Those of Two-state Neurons. National Academy of Sciences, v.81, p.3088-92, 1984. **Proceedings**.

HOPFIELD, J.; TANK, D. Neural computation of decisions in optimization problems. **Biological Cybernetics**, v.5, p.141-52, 1985.

ISHIBUCHI, H.; TAMURA, R.; TANAKA, H. Flow Shop Scheduling by Simulated Annealing. **Transactions of the Institute of Systems, Control and Information Engineers**, v.4, p.111-17, 1991.

KIM, M-H.; K, Y-D. Simulation-Based Real-Time Scheduling In A Flexible Manufacturing Systems. **Journal of Manufacturing Systems**, v.13, n.2, p.85-93, 1994.

KIRKPATRICK, S.; GELATT, C.D.; VECCHI, M.P. Optimization by simulated annealing. **Science**, v.220, n.671, 1983.

KLAFEHN, K.; WEINROTH, J.; BORONICO, J. **Computer Simulations in Operations Management**. Greenwood Publishing Group, 1996.

KOHONEN, T. Self-organized formation of topologically correct feature maps. **Biological Cybernetics**, v.43, p.59-69, 1982.

KOHONEN, T. **Self-Organization and Associative Memory**. Springer-Verlag, 1997.

KOHONEN, T. **Self-Organizing Maps**, 2nd ed., 1997.

KUNG, S. Y. **Digital Neural Networks**. PTR Prentice Hall, 1993.

KUNG, S-K.; MARSDEN, J.R. Development and implementation of a dispersed decision process: an FMS scheduling example. **Computer Integrated Manufacturing Systems**, v.8, n.2, p. 93-103, 1995.

LANE, R.; EVANS, S. Solving problems in production scheduling. **Computer Integrated Manufacturing Systems**, v.8, n.2, p.117-24, 1995.

LAWELER, B. L., LENSTRA, J. K., and RINNOOY KAN, A. H. G., 1981, *Minimize Maximum Lateness in a Two-Machine Open Job-Shop*, Mathematical of Operations Research, **6**, 153-158.

LENSTRA, J. K., RINNOOY KAN, A. H. G., and BRUCKER, P., 1977, *Complexity of Machine Scheduling Problems*, Annals of Discrete Mathematics, 7, 342-362.

LITTLE, D.; HEMMINGS, A. Automated assembly scheduling: A review. **Computer Integrated Manufacturing Systems**, v.7, n.1, p. 51-61, 1994.

LOOI, C-K. Neural Network Methods in Combinatorial Optimization. **Computer Operations Research**, v.19, p.191-208, 1992.

LOZANO, S.; GUERRERO, F.; CANCA, D.; SMITH K. A. **Generation of route timetables using self-organizing feature maps**. Submitted to C. Dagli et al. (Eds.). Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining and Complex Systems. ASME Press, v.10, 2000.

LUCENA, C. J. P. **Inteligência Artificial e Engenharia de Software**. Rio de Janeiro, PUC/RJ – IBM Brasil, 1987. p.106-67.

MAMALIS, A.G.; MALAGARDIS, I. Determination of Due Dates in Job Shop Scheduling by Simulated Annealing. **Computer Integrated Manufacturing Systems**, v.9, p.65-72, 1996.

MAREN, A.J.; HARTSON, C.T.; PAP, R.M. **Handbook of Neural Computing Applications**. Academis Press, Inc., 1990.

MOORE, J. M.; WILSON, R. C. A Review of Simulation Research in Job Shop Scheduling. **Production and Inventory Management**, v.8, p.1-10, 1967.

MORTON, T.; PENTICO, D.W. **Heuristic Scheduling Systems**. New York, John Wiley & Sons, 1993.

NOVO DICIONÁRIO AURÉLIO – SÉCULO XXI. Editora Nova Fronteira, 1999.

OW, P.S.; MORTON, T. E. Filtered beam Search in Scheduling. **International Journal of Production Research**, v.26, p.297-307, 1988.

PARZEN, E., 1960, *Modern Probability Theory and Its Applications*, John Wiley & Sons, Inc., NY.

PERFETTI, R. Optimization Neural Network For Solving Flow Problems. **IEEE Transactions on Neural Networks**, v.6, n.5, p.1287-91, 1995.

PINEDO, M. **Scheduling: Theory, Algorithms and Systems**. Prentice Hall, 1995. (International Series in Industrial and Systems Engineering)

REINHARDT, J.; MULLER, B. **Neural Networks – An Introduction**. Springer-Verlag, 1991.

RITTER, H.; MARTINETZ, T.; SCHULTEN, K. **Neural Computation and Self-Organizing Maps: An Introduction**. Addison-Wesley, 1992.

SANTORO, M.C. e PACHECO, R.F. Proposta de Classificação Hierarquizada dos Modelos de Solução para o Problema de Job Shop Scheduling, **Gestão e Produção**, v.6, n.1, p.1-15, 1999.

SHIRAZI, B.; YIH, S. Critical analysis of applying Hopfield neural net model to optimization problems. IEEE International Conference on Systems, Man and Cybernetics, p. 210-15, 1988. **Proceedings**.

SMITH, K. A. **Solving Combinatorial Optimization Problems Using Neural Networks**. Melbourne, 1996. Tese (Doutorado), The University of Melbourne.

SMITH, K. A.; PALANISWAMI, M.; KRISHNAMOORTHY, M. Neural Techniques for Combinatorial Optimization with Applications. **IEEE Transactions on Neural Networks**, v.9, n.6, p.1301-18, 1998.

SMITH, K. A.; PALANISWAMI, M.; KRISHNAMOORTHY, M. A hybrid neural approach to combinatorial optimization. **Comput. Operations Res.**, v.23, p.597-610, 1998.

SMITH, K. A. Neural Networks for Combinatorial Optimization: A Review of More than a Decade of Research. **INFORMS Journal on Computing**, v.11, n.1, p.15-34, 1999.

SMITH, K. A. **Introduction to Neural Networks and Data Mining for Business Applications**. Eruditions Publishing, 1999.

TEDESCHI, S.G.G. **Seleção de Meta-Regras Para Alteração Dinâmica do Despacho da Produção**. São Paulo, 1997. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo.

TZAFESTAS, S.; TRIANTAFYLLAKIS, A. A new adaptively weighted combinatorial dispatching rule for complex scheduling problems. **Computer Integrated Manufacturing Systems**, v.7, p.7-15, 1994.

UCKUN, S.; BAGCHI, S.; KAWAMURA, K.; MIYABE, Y. Managing Genetic Search in Job Shop Scheduling. **IEEE Expert**, v.8, p.15-26, 1993.

VAITHYANATHAN, S.; IGNIZIO, J.P. A Stochastic Neural Network for Resource Constrained Scheduling. **Computer Operations Research**, v.19, p.241-54, 1992.

VAN LAARHOVEN, P. J. M., AARTS, E. H. L., 1987, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Holland.

WELSTEAD, S.T. **Neural Network and Fuzzy Logic Applications in C/C++**. John Wiley & Sons, Inc., 1994.

WIDMER, M.; HERTZ, A. Tabu Search Techniques: A Tutorial and an Application to Neural Networks. **OR Spectrum**, v.11, p.131-41, 1989.

YIP, P.P.C.; PAO, Y–H. Combinatorial Optimization With Use Of Guided Evolutionary Simulated Annealing. **IEEE Transactions on Neural Networks**, v.6, p.290-96, 1995.

APÊNDICE A – RESULTADOS ADICIONAIS

Com o fim de ilustrar o problema de seqüenciamento de uma forma mais abrangente, o programa implementado em Visual Basic também outros critérios de desempenho e realizou comparações com a regra SPT (as seqüências obtidas com a regra SPT encontram-se no apêndice B). Os resultados obtidos nestes casos foram lançados aqui uma vez que não devem ser utilizados para avaliar o desempenho da rede já que a mesma não foi treinada para atingir estes objetivos.

Nas tabelas à seguir, encontram-se resultados obtidos para os seguintes objetivos:

- Desvio Médio (L_{avg});
- Desvio Máximo (L_{max});
- Desvio Total (L_{tot});
- Número de ordens em atraso (N_{tot})
- Atraso Máximo (T_{max});
- Atraso Total (T_{tot});
- Atraso Ponderado (T_{wt}).

Objetivo: Reduzir Desvio Médio (L_{avg})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	-26,8	331,05	291,25	-11,9	14,28
SPT	-29,6	208,80	237,00	-23,1	-5,28
SOFM	-28,5	209,90	238,80	-14,9	4,48
Melhor	SPT	SPT	SPT	SPT	SPT
Pior	EDD	EDD	EDD	EDD	EDD

Objetivo: Reduzir Desvio Máximo (L_{max})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	47	689	920	47	121
SPT	64	1756	1099	62	127
SOFM	48	1618	1070	47	127
Melhor	EDD	EDD	EDD	EDD / SOFM	EDD
Pior	SPT	SPT	SPT	SPT	SPT / SOFM

Objetivo: Reduzir Número Total de Ordens Atrasadas (N_{tot})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	2	20	15	11	19
SPT	2	13	12	6	14
SOFM	2	12	12	8	16
Melhor	Indiferente	SOFM	SOFM / SPT	SPT	SPT
Pior	Indiferente	EDD	EDD	EDD	EDD

Objetivo: Reduzir Número Ponderado de Ordens Atrasadas (N_{wt})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	0,125	1,000	0,755	0,348	0,467
SPT	0,125	0,633	0,633	0,217	0,367
SOFM	0,125	0,561	0,633	0,261	0,400
Melhor	Indiferente	SOFM	SOFM / SPT	SPT	SPT
Pior	Indiferente	EDD	EDD	EDD	EDD

Objetivo: Reduzir Atraso Máximo (T_{max})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	16	689	920	41	121
SPT	17	1756	1099	46	127
SOFM	17	1618	1070	44	127
Melhor	EDD	EDD	EDD	EDD	EDD
Pior	SPT / SOFM	SPT	SPT	SPT	SPT / SOFM

Objetivo: Reduzir Atraso Total (T_{tot})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	22	6621	6803	251	1235
SPT	23	7440	7271	137	737
SOFM	24	7417	7304	211	886
Melhor	EDD	EDD	EDD	SPT	SPT
Pior	SOFM	SPT	SOFM	EDD	EDD

Objetivo: Reduzir Atraso Ponderado (T_{wt})					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
EDD	1,38	343,69	345,60	7,84	31,09
SPT	1,44	313,15	401,04	5,67	20,82
SOFM	1,50	314,75	399,98	6,68	22,22
Melhor	EDD	SPT	EDD	SPT	SPT
Pior	SOFM	EDD	SPT	EDD	EDD

Resultados Alcançados pela Rede Neural em Comparação com EDD e SPT					
Problema	**1** **N = 20**	**4** **N = 20**	**5** **N = 20**	**2** **N = 30**	**3** **N = 40**
Lavg	Intermediário	Intermediário	Intermediário	Intermediário	Intermediário
Lmax	Intermediário	Intermediário	Intermediário	= EDD	= SPT
Ntot	= EDD e SPT	Melhor	= SPT	Intermediário	Intermediário
Nwt	= EDD e SPT	Melhor	= SPT	Intermediário	Intermediário
Tmax	= SPT	Intermediário	Intermediário	Intermediário	= SPT
Ttot	Pior	Intermediário	Pior	Intermediário	Intermediário
Twt	Pior	Intermediário	Intermediário	Intermediário	Intermediário

APÊNDICE B – SEQÜÊNCIAS OBTIDAS COM A REGRA SPT

Problema 1 – Baseado em dados reais - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,063
2	1	1	2	48	0	0,063
3	2	1	3	48	0	0,063
4	3	1	4	48	0	0,021
5	4	1	5	48	0	0,021
6	5	1	6	48	0	0,042
7	6	1	7	48	0	0,042
13	7	1	8	72	0	0,063
8	8	2	10	48	0	0,063
9	10	2	12	48	0	0,063
11	12	2	14	48	0	0,021
12	14	2	16	48	0	0,042
14	16	3	19	48	0	0,063
15	19	4	23	48	0	0,063
10	23	5	28	48	0	0,063
17	28	5	33	48	0	0,021
18	33	5	38	48	0	0,042
16	38	6	44	48	0	0,063
20	44	10	54	48	6	0,063
19	54	11	65	48	17	0,063
Regra SPT ; N_{tot} = 2 ; N_{wt} = 0,125 ; T_{max} = 17 ; T_{tot} = 23 ; T_{wt} = 1,44						

Problema 2 – Baseado em dados reais – 30 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,043
2	1	1	2	48	0	0,043
3	2	1	3	48	0	0,043
4	3	1	4	48	0	0,043
5	4	1	5	48	0	0,043
6	5	1	6	48	0	0,014
7	6	1	7	48	0	0,014
8	7	1	8	48	0	0,029
9	8	1	9	48	0	0,029
15	9	1	10	72	0	0,043
23	10	1	11	72	0	0,014
24	11	1	12	72	0	0,029
10	12	2	14	48	0	0,043
11	14	2	16	48	0	0,043
13	16	2	18	48	0	0,014
14	18	2	20	48	0	0,029
30	20	2	22	72	0	0,043
16	22	3	25	48	0	0,043
25	25	3	28	48	0	0,014
26	28	3	31	48	0	0,029
17	31	4	35	48	0	0,043
27	35	4	39	48	0	0,014
28	39	4	43	48	0	0,029
12	43	5	48	48	0	0,043
19	48	5	53	48	5	0,014
20	53	5	58	48	10	0,029
18	58	6	64	48	16	0,043
29	64	9	73	48	25	0,043
22	73	10	83	48	35	0,043
21	83	11	94	48	46	0,043
Regra SPT : $N_{tot} = 6$; $N_{wt} = 0,217$; $T_{max} = 46$; $T_{tot} = 137$; $T_{wt} = 5,67$						

Problema 3 – Baseado em dados reais – 40 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	1	1	48	0	0,033
2	1	1	2	48	0	0,033
3	2	1	3	48	0	0,033
4	3	1	4	48	0	0,033
5	4	1	5	48	0	0,033
6	5	1	6	48	0	0,011
7	6	1	7	48	0	0,011
8	7	1	8	48	0	0,011
9	8	1	9	48	0	0,022
10	9	1	10	48	0	0,022
11	10	1	11	48	0	0,022
17	11	1	12	72	0	0,033
25	12	1	13	72	0	0,011
26	13	1	14	72	0	0,022
37	14	1	15	96	0	0,033
12	15	2	17	48	0	0,033
13	17	2	19	48	0	0,033
15	19	2	21	48	0	0,011
16	21	2	23	48	0	0,022
32	23	2	25	72	0	0,033
18	25	3	28	48	0	0,033
27	28	3	31	48	0	0,011
28	31	3	34	48	0	0,022
19	34	4	38	48	0	0,033
29	38	4	42	48	0	0,011
30	42	4	46	48	0	0,022
14	46	5	51	48	3	0,033
21	51	5	56	48	8	0,011
22	56	5	61	48	13	0,022
20	61	6	67	48	19	0,033
33	67	6	73	48	25	0,011
34	73	6	79	48	31	0,022
31	79	9	88	48	40	0,033
24	88	10	98	48	50	0,033
23	98	11	109	48	61	0,033
38	109	11	120	48	72	0,011
39	120	11	131	48	83	0,022
35	131	12	143	48	95	0,033
36	143	15	158	48	110	0,033
40	158	17	175	48	127	0,033
Regra SPT : N_{tot} = 14 ; N_{wt} = 0,367 ; T_{max} = 127 ; T_{tot} = 737 ; T_{wt} = 20,82						

Problema 4 – Baseado em BAKER (1997) - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	55	55	109	0	0,051
2	55	68	123	169	0	0,046
3	123	70	193	1039	0	0,036
4	193	73	266	1158	0	0,031
5	266	77	343	1107	0	0,066
6	343	78	421	0	421	0,046
7	421	85	506	767	0	0,061
8	506	86	592	993	0	0,077
9	592	89	681	643	38	0,071
10	681	89	770	667	103	0,051
11	770	92	862	75	787	0,031
12	862	93	955	612	343	0,041
13	955	94	1049	780	269	0,061
14	1049	94	1143	816	327	0,066
15	1143	98	1241	721	520	0,061
16	1241	108	1349	555	794	0,026
17	1349	126	1475	1166	309	0,066
18	1475	138	1613	529	1084	0,041
19	1613	143	1756	0	1756	0,041
20	1756	170	1926	1237	689	0,031
Regra SPT : $N_{tot} = 13$; $N_{wt} = 0,633$; $T_{max} = 1756$; $T_{tot} = 7440$; $T_{wt} = 313,15$						

Problema 5 – Baseado em BAKER (1997) - 20 Ordens de Produção						
id	**s**	**p**	**c**	**d**	**T**	**w**
1	0	68	68	437	0	0,037
2	68	79	147	521	0	0,027
3	147	80	227	678	0	0,069
4	227	86	313	841	0	0,037
5	313	89	402	746	0	0,032
6	402	94	496	520	0	0,037
7	496	96	592	610	0	0,064
8	592	97	689	1112	0	0,064
9	689	100	789	772	17	0,064
10	789	105	894	566	328	0,043
11	894	106	1000	928	72	0,043
12	1000	109	1109	472	637	0,043
13	1109	109	1218	910	308	0,064
14	1218	112	1330	499	831	0,048
15	1330	118	1448	498	950	0,080
16	1448	119	1567	1084	483	0,032
17	1567	120	1687	617	1070	0,064
18	1687	124	1811	1153	658	0,027
19	1811	127	1938	1120	818	0,074
20	1938	135	2073	974	1099	0,053
Regra SPT : $N_{tot} = 12$; $N_{wt} = 0,633$; $T_{max} = 1099$; $T_{tot} = 7271$; $T_{wt} = 401,04$						