UNIVERSIDADE DE SÃO PAULO
Escola de Engenharia de São Carlos
Programa de Pós-Graduação em Engenharia Elétrica

Matheus Giovanni Pires

**Abordagem Neuro-Genética Para Mapeamento
de Problemas de Conexão em Otimização
Combinatória**

São Carlos
Mai 2009

Matheus Giovanni Pires

**Abordagem Neuro-Genética Para Mapeamento
de Problemas de Conexão em Otimização
Combinatória**

*Tese apresentada à Escola de Engenharia de
São Carlos da Universidade de São Paulo,
sendo parte dos requisitos para obtenção do
título de Doutor em Engenharia Elétrica.*

Orientador: Prof. Dr. Ivan Nunes da Silva

**São Carlos
Maio 2009**

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTES
TRABALHOS, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO,
PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

P667a

Pires, Matheus Giovanni

Abordagem neuro-genética para mapeamento de problemas
de conexão em otimização combinatória / Matheus Giovanni
Pires ; Ivan Nunes da Silva. -- São Carlos, 2009.

Tese (Doutorado-Programa de Pós-Graduação em
Engenharia Elétrica e Área de Concentração em Sistemas
Dinâmicos) -- Escola de Engenharia de São Carlos da
Universidade de São Paulo, 2009.

1. Redes neurais artificiais. 2. Algoritmos genéticos.
3. Otimização combinatória. 4. Problema das n-rainhas.
5. Problema do emparelhamento bipartido. 6. Problema do
caminho mínimo. I. Título.

FOLHA DE JULGAMENTO

Candidato: Bacharel **MATHEUS GIOVANNI PIRES**

Tese defendida e julgada em 21/05/2009 perante a Comissão Julgadora:

Prof. Associado **IVAN NUNES DA SILVA (Orientador)**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Dr. **ROGÉRIO ANDRADE FLAUZINO**
(Escola de Engenharia de São Carlos/USP)

Aprovado

Prof^ª. Dr^ª. **MARIA DO CARMO NICOLETTI**
(Universidade Federal de São Carlos/UFSCar)

APROVADO

Prof. Associado **JOSÉ REINALDO SILVA**
(Escola Politécnica/USP)

aprovado

Prof^ª. Associada **ROSELI APARECIDA FRANCELIN ROMERO**
(Instituto de Ciências Matemáticas e de Computação/USP)

APROVADA

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e
Presidente da Comissão de Pós-Graduação

*Porque Dele e por meio Dele, e para Ele são todas as coisas.
Glória, pois, a Ele eternamente. Amém.*
Romanos 11, 36

Agradecimentos

Agradeço a DEUS, em nome de Jesus, por tudo o que tem feito por mim e por todas as vitórias alcançadas. Agradeço por mais uma vez me capacitar na realização de um trabalho tão importante para a minha vida, mas acima de qualquer coisa, esta é mais uma obra para a honra e a glória de Seu nome. Amém!

Ao professor Ivan Nunes da Silva, meus sinceros agradecimentos pelos seus valiosos ensinamentos, fundamentais para a realização deste trabalho e para o meu aprendizado. Obrigado por sua paciência e por me aceitar como aluno de doutorado. Desempenhando um papel além de orientador, sempre me incentivou e apoiou nas dificuldades encontradas. Admiro-o não somente como excelente pesquisador que é, mas também pelo caráter e exemplo como pessoa. Valeu Ivan!

À minha esposa Fabiana, a quem amo muito. Uma pessoa que além de amar, eu admiro e tenho como referência de vida. Sempre com grande afinho e fé em Deus, não teme lutar pelos seus objetivos. Sem dúvida nenhuma, ela é o meu porto seguro. Quando estou triste, desanimado, por qualquer motivo, seja tolo ou não, ela sabe como cuidar de mim... Da maneira mais simples e gostosa que possa ser. Obrigado meu Deus, por ter me dado uma pessoa tão especial. Obrigado meu Amor, por tudo. Te Amo!

Agradeço a toda minha família, meus pais Germano e Aparecida e as minhas irmãs, Marina, Marisa e Pâmela. Sei que sempre torceram por mim e continuam torcendo. Mesmo distante, penso em vocês todos os dias. Obrigado por tudo. Amo todos vocês.

Aos pais de minha esposa, Valdir e Vilma, e Helena (avó da Fabiana), agradeço pelas orações e pelo pensamento positivo. Que Deus os abençoe!

Evandro e Rodrigo Faccioli, obrigado pela amizade e pelo companheirismo. Obrigado pelos momentos descontraídos depois de um longo dia de estudos. Sucesso para vocês.

Marcelo Suetake, uma pessoa extremamente competente e modesta. Sempre disposto a ajudar os colegas e compartilhar os seus conhecimentos. Obrigado pelas suas valiosas dicas.

A todos os amigos do LAIPS, da graduação da UNIP, da Universidade Estadual de Feira de Santana, da Igreja Presbiteriana Betel, os quais infelizmente não poderão ser citados um a um neste texto, que direta ou indiretamente contribuíram para a realização deste trabalho... Muito obrigado.

Resumo

PIRES, M. G. (2009). Abordagem Neuro-Genética para Mapeamento de Problemas de Conexão em Otimização Combinatória. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

Devido a restrições de aplicabilidade presentes nos algoritmos para a solução de problemas de otimização combinatória, os sistemas baseados em redes neurais artificiais e algoritmos genéticos oferecem um método alternativo para solucionar tais problemas eficientemente. Os algoritmos genéticos devem a sua popularidade à possibilidade de percorrer espaços de busca não-lineares e extensos. Já as redes neurais artificiais possuem altas taxas de processamento por utilizarem um número elevado de elementos processadores simples com alta conectividade entre si. Complementarmente, redes neurais com conexões realimentadas fornecem um modelo computacional capaz de resolver vários tipos de problemas de otimização, os quais consistem, geralmente, da otimização de uma função objetivo que pode estar sujeita ou não a um conjunto de restrições. Esta tese apresenta uma abordagem inovadora para resolver problemas de conexão em otimização combinatória utilizando uma arquitetura neuro-genética. Mais especificamente, uma rede neural de Hopfield modificada é associada a um algoritmo genético visando garantir a convergência da rede em direção aos pontos de equilíbrio factíveis que representam as soluções para os problemas de otimização combinatória.

Palavras-Chave: Redes Neurais Artificiais, Algoritmos Genéticos, Otimização Combinatória, Problema das N -Rainhas, Problema do Emparelhamento Bipartido, Problema do Caminho Mínimo.

Abstract

PIRES, M. G. (2009). Neurogenetic Approach for Mapping Connection Problems in Combinatorial Optimization. Thesis (Doctoral Degree) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

Due to applicability constraints involved with the algorithms for solving combinatorial optimization problems, systems based on artificial neural networks and genetic algorithms are alternative methods for solving these problems in an efficient way. The genetic algorithms must its popularity to make possible cover nonlinear and extensive search spaces. On the other hand, artificial neural networks have high processing rates due to the use of a massive number of simple processing elements and the high degree of connectivity between these elements. Additionally, neural networks with feedback connections provide a computing model capable of solving a large class of optimization problems, which refer to optimization of an objective function that can be subject to constraints. This thesis presents a novel approach for solving connection problems in combinatorial optimization using a neurogenetic approach. More specifically, a modified Hopfield neural network is associated with a genetic algorithm in order to guarantee the convergence of the network to the equilibrium points, which represent feasible solutions for the combinatorial optimization problems.

Keywords: Artificial Neural Networks, Genetic Algorithms, Combinatorial Optimization, *N*-Queens Problem, Bipartite Graph Optimization, Shortest Path Problem.

Lista de Figuras

FIGURA 2.1: UM PROBLEMA DE PROGRAMAÇÃO DINÂMICA.	10
FIGURA 2.2: RESULTADO DA APLICAÇÃO DO ALGORITMO DE PROGRAMAÇÃO DINÂMICA.	16
FIGURA 2.3: CAMINHO SIMPLES MAIS LONGO.	20
FIGURA 3.1: A REDE NEURAL DE HOPFIELD CONVENCIONAL.	27
FIGURA 3.2: HARDWARE ANALÓGICO DA REDE DE HOPFIELD.	30
FIGURA 3.3: CRUZAMENTO SIMPLES.	38
FIGURA 3.4: CRUZAMENTO ARITMÉTICO.	38
FIGURA 3.5: CRUZAMENTO HEURÍSTICO.	38
FIGURA 3.6: BLX- α APLICADO A UM ESPAÇO UNIDIMENSIONAL.	39
FIGURA 3.7: BLX- α APLICADO A UM ESPAÇO BIDIMENSIONAL.	39
FIGURA 3.8: MUTAÇÃO UNIFORME.	40
FIGURA 4.1: A REDE DE HOPFIELD GENÉTICA.	47
FIGURA 4.2: FLUXOGRAMA DA REDE DE HOPFIELD GENÉTICA.	49
FIGURA 5.1: PROBLEMA DO CAMINHO MÍNIMO.	60
FIGURA 5.2: UM SIMPLES GRAFO COM 6 NÓS E 10 ARCOS.	66
FIGURA 5.3: CODIFICAÇÃO BASEADA EM PRIORIDADE.	66
FIGURA 5.4: REPRESENTAÇÃO GENÉTICA PARA O PROBLEMA DO CAMINHO MÍNIMO.	76
FIGURA 5.5: O PROBLEMA DAS N-RAINHAS, PARA $N=4$	79
FIGURA 5.6: REPRESENTAÇÃO GENÉTICA PARA O PROBLEMA DAS N-RAINHAS.	81
FIGURA 5.7: EXEMPLO DE UM GRAFO.	82
FIGURA 5.8: EXEMPLO DE UM GRAFO BIPARTIDO.	83
FIGURA 5.9: GRAFO BIPARTIDO.	84
FIGURA 5.10: REPRESENTAÇÃO GENÉTICA PARA O PROBLEMA DO EMPARELHAMENTO BIPARTIDO.	85
FIGURA 6.1: EVOLUÇÃO DA MATRIZ V PARA O PCM ($N=8$ E $M=16$).	93
FIGURA 6.2: COMPORTAMENTO DA FUNÇÃO OBJETIVO PARA O PCM ($N=8$ E $M=16$).	94
FIGURA 6.3: EVOLUÇÃO DA MATRIZ V PARA O PROBLEMA DAS N-RAINHAS COM $N=5$	96
FIGURA 6.4: EVOLUÇÃO DA MATRIZ V PARA O PROBLEMA DAS N-RAINHAS COM $N=8$	96
FIGURA 6.5: EVOLUÇÃO DA MATRIZ V PARA O PROBLEMA DO EMPARELHAMENTO BIPARTIDO ($N=5$).	98
FIGURA 6.6: EVOLUÇÃO DA MATRIZ V PARA O PROBLEMA DO EMPARELHAMENTO BIPARTIDO ($N=8$).	99

Lista de Tabelas

TABELA 5.1: RESUMO DOS TRABALHOS APLICADOS AO PCM USANDO RNA.	65
TABELA 5.2: RESUMO DOS TRABALHOS APLICADOS AO PCM USANDO AG.	73
TABELA 6.1: RESULTADOS DO COMPRIMENTO MÉDIO NORMALIZADO PARA O PCM.....	89
TABELA 6.2: RESULTADOS DOS TEMPOS DE PROCESSAMENTO (EM SEGUNDOS).	91
TABELA 6.3: SOLUÇÕES PARA O PROBLEMA DAS N -RAINHAS ($N=5$).	94
TABELA 6.4: SOLUÇÕES PARA O PROBLEMA DAS N -RAINHAS ($N=8$).	95

Lista de Algoritmos

ALGORITMO 2.1: ALGORITMO DE PROGRAMAÇÃO DINÂMICA (CORMEM ET AL., 2002).	15
ALGORITMO 2.2: ALGORITMO DE IMPRESSÃO DA COLOCAÇÃO ÓTIMA DOS PARÊNTESES NA MULTIPLICAÇÃO DE MATRIZES.....	17
ALGORITMO 3.1: ESTRUTURA DE UM ALGORITMO GENÉTICO.	32
ALGORITMO 3.2: SELEÇÃO PELA ROLETA (MITCHELL, 1996).....	35
ALGORITMO 3.3: SELEÇÃO POR TORNEIO (MITCHELL, 1996).....	36

Lista de Abreviaturas e Siglas

AG	Algoritmos Genéticos
BT	Busca Tabu
CF	Computação Flexível
PCM	Problema do Caminho Mínimo
EB	Emparelhamento Bipartido
PD	Programação Dinâmica
RHG	Rede de Hopfield Genética
RNA	Redes Neurais Artificiais
MA	Memórias Associativas

Sumário

RESUMO	III
ABSTRACT	V
LISTA DE FIGURAS	VII
LISTA DE TABELAS.....	IX
LISTA DE ALGORITMOS.....	XI
LISTA DE ABREVIATURAS E SIGLAS	XIII
CAPÍTULO 1 INTRODUÇÃO	1
1.1 MOTIVAÇÃO	1
1.2 OBJETIVOS E JUSTIFICATIVA.....	2
1.3 ORGANIZAÇÃO DA TESE	3
CAPÍTULO 2 OTIMIZAÇÃO DE SISTEMAS	5
2.1 PROBLEMAS E TÉCNICAS DE OTIMIZAÇÃO.....	5
2.2 OTIMIZAÇÃO COMBINATÓRIA	6
2.2.1 <i>Programação Matemática</i>	9
2.2.1.1 Programação Dinâmica	10
2.2.1.2 Aplicabilidade da Programação Dinâmica.....	17
2.2.2 <i>Métodos Heurísticos</i>	22
2.2.3 <i>Métodos de Inteligência Artificial</i>	23
2.3 CONSIDERAÇÕES FINAIS	23
CAPÍTULO 3 REDES NEURAIS ARTIFICIAIS E ALGORITMOS GENÉTICOS.....	25
3.1 REDES NEURAIS ARTIFICIAIS	25
3.2 ALGORITMOS GENÉTICOS.....	31
3.2.1 <i>Princípios Básicos de Algoritmos Genéticos</i>	31
3.2.2 <i>Parâmetros de um Algoritmo Genético</i>	33
3.2.2.1 Representação Genética.....	33
3.2.2.2 População Inicial	33
3.2.2.3 Função de Avaliação	34
3.2.2.4 Método de Seleção	34
3.2.2.5 Operadores Genéticos.....	36
3.2.2.6 Critério de Parada.....	40
3.3 CONSIDERAÇÕES FINAIS	41
CAPÍTULO 4 ABORDAGEM NEURO-GENÉTICA	43
4.1 ABORDAGEM NEURAL E SUAS LIMITAÇÕES.....	43
4.2 ABORDAGEM DO SUBESPAÇO-VÁLIDO DE SOLUÇÕES	46

4.3 A REDE DE HOPFIELD GENÉTICA	47
4.4 DINÂMICA DA REDE DE HOPFIELD GENÉTICA.....	49
4.5 CONSIDERAÇÕES FINAIS	53
CAPÍTULO 5 APLICAÇÃO DA RHG EM PROBLEMAS DE OTIMIZAÇÃO COMBINATÓRIA.....	55
5.1 NOTAÇÕES, DEFINIÇÕES E PROPRIEDADES	56
5.2 PROBLEMAS DE PROGRAMAÇÃO DINÂMICA: O PROBLEMA DO CAMINHO MÍNIMO	59
5.2.1 <i>Redes Neurais Artificiais Aplicadas ao Problema do Caminho Mínimo</i>	60
5.2.2 <i>Algoritmos Genéticos Aplicados ao Problema do Caminho Mínimo</i>	66
5.2.3 <i>Modelamento do Problema do Caminho Mínimo através da Rede de Hopfield Genética</i>	74
5.3 O PROBLEMA DAS <i>N</i> -RAINHAS	77
5.4 O PROBLEMA DO EMPARELHAMENTO BIPARTIDO	82
5.5 CONSIDERAÇÕES FINAIS	86
CAPÍTULO 6 EXPERIMENTOS E RESULTADOS	87
6.1 EXPERIMENTOS: PROBLEMA DO CAMINHO MÍNIMO	87
6.2 EXPERIMENTOS: PROBLEMA DAS <i>N</i> -RAINHAS	94
6.3 EXPERIMENTOS: PROBLEMA DO EMPARELHAMENTO BIPARTIDO	97
CAPÍTULO 7 CONCLUSÕES E TRABALHOS FUTUROS	101
7.1 CONCLUSÕES	101
7.2 TRABALHOS FUTUROS	103
REFERÊNCIAS BIBLIOGRÁFICAS.....	105

Capítulo 1

Introdução

1.1 Motivação

Um problema de otimização consiste em, dada uma função objetivo equacionada usando um conjunto de variáveis, encontrar valores para estas variáveis que correspondem a um valor ótimo para uma função objetivo e que satisfaçam um conjunto de restrições associadas à função objetivo dada. Um problema desta natureza é resolvido formulando um modelo matemático equivalente e utilizando um método de solução adequado ao tipo de problema. Embora existam na literatura diversos métodos matemáticos tradicionais que possam ser aplicados na resolução de problemas de otimização, há a necessidade crescente de investigar métodos alternativos que exploram arquiteturas de processamento paralelas e adaptativas, como forma de melhorar a aplicabilidade e o desempenho em relação aos métodos existentes. Assim sendo, este trabalho propõe o desenvolvimento de uma abordagem híbrida, a qual une metodologias de redes neurais artificiais e algoritmos genéticos. Entre as principais vantagens em se utilizar redes neurais artificiais em otimização, destacam-se a capacidade intrínseca de operação em paralelo, a simplicidade de implementação e o alcance de altas taxas de computação por intermédio dos neurônios artificiais. Já os algoritmos genéticos merecem destaque em função de sua capacidade de exploração do espaço de busca aliada ao uso do paralelismo, permitindo que várias possibilidades de solução sejam exploradas simultaneamente.

Tal combinação neuro-genética possui grande aceitação na comunidade científica, pois adere ao princípio de balanceamento de vantagens da inteligência computacional (Jang et al.,

1997; Rezende, 2003), onde metodologias diferentes colaboram entre si potencializando a utilidade e aplicabilidade dos sistemas resultantes.

1.2 Objetivos e Justificativa

O objetivo deste trabalho é o desenvolvimento de uma abordagem híbrida, a qual é constituída por uma rede neural artificial e por um algoritmo genético, para a solução de problemas de conexão em otimização combinatória. Em outras palavras, é proposta uma alteração na Rede de Hopfield convencional, resultando em uma nova rede neural denominada Rede de Hopfield Genética (RHG). Nesta arquitetura híbrida, a rede neural possui a função de satisfazer as restrições presentes nos problemas de otimização e o algoritmo genético é responsável por minimizar a função objetivo do problema.

Vários problemas práticos de engenharia elétrica, engenharia aeroespacial, economia e pesquisa operacional podem ser modelados usando grafos (Ziviani, 2005), e constituem-se em problemas de otimização combinatória. Tais problemas podem ser agrupados em cinco classes (Goldberg e Luna, 2005): i) problemas de conexão, ii) problemas de fluxo em redes, iii) problema do caixeiro viajante, iv) problemas de roteamento e v) problemas de cobertura e particionamento

Os problemas de conexão estão associados às alternativas de conexão entre vértices de um grafo que modela algum tipo de problema. Para este tipo de problema, o tomador de decisão precisa ser capaz de exibir uma topologia adequada ao modelo, de organizar as configurações desejáveis dentro desta topologia e de estabelecer critérios de escolhas destas configurações.

Atualmente, utilizam-se arquiteturas de redes neurais diferentes, sistemáticas de modelamento distintas ou a combinação de diferentes metodologias, com o intuito de modelar problemas de otimização combinatória. Algumas destas arquiteturas ainda apresentam dificuldades no modelamento destes problemas, resultando-se em soluções algumas vezes inviáveis, ou seja, a não satisfação das restrições presentes nos problemas de otimização combinatória. No Capítulo 2 serão detalhados alguns trabalhos encontrados na literatura com o intuito de apresentar estas dificuldades.

As vantagens na utilização da abordagem neuro-genética proposta nesta tese em relação às demais propostas encontradas na literatura são as seguintes:

- a) Modelamento de problemas de otimização de naturezas diferentes utilizando uma mesma metodologia;
- b) Viabilização do tratamento das restrições envolvidas com os problemas;
- c) Não necessidade de definição de parâmetros de controle ou ponderação para sua inicialização.
- d) Facilidade de implementação em *hardware*.

Diante das vantagens destacadas pela abordagem proposta e da dificuldade presente nos demais trabalhos, a RHG torna-se uma ferramenta promissora para a solução de problemas de conexão em otimização combinatória.

1.3 Organização da tese

Este trabalho está dividido em sete capítulos, os quais são organizados como se segue.

No Capítulo 2 são apresentados conceitos envolvendo otimização combinatória, os tipos de problemas existentes e quais as técnicas comumente utilizadas para resolvê-los.

O Capítulo 3 apresenta as metodologias de inteligência computacional utilizadas no desenvolvimento deste trabalho, redes neurais artificiais e algoritmos genéticos, descrevendo os conceitos, parâmetros e respectivas aplicabilidades.

No Capítulo 4 é formulada a abordagem neuro-genética desenvolvida, descrevendo a rede neural utilizada, a qual incorpora a técnica do subespaço-válido de soluções para confinar as restrições envolvidas com os problemas de otimização bem como o algoritmo genético, aplicado na otimização das funções objetivo destes problemas.

O Capítulo 5 apresenta a aplicação da abordagem neuro-genética para a solução de problemas de conexão em otimização combinatória. Mostra-se também um panorama das abordagens neurais e genéticas disponíveis na literatura, as quais são utilizadas na solução de problemas de otimização combinatória, mais especificamente para o problema do caminho mínimo.

Os experimentos e resultados obtidos da aplicação da abordagem neuro-genética aos problemas do caminho mínimo, das N -rainhas e do emparelhamento bipartido são relatados no Capítulo 6.

Finalmente, no Capítulo 7, apresentam-se as conclusões e os trabalhos futuros.

Capítulo 2

Otimização de Sistemas

Neste capítulo são apresentados os problemas de otimização combinatória e os principais métodos para suas soluções. Na Seção 2.1 são apresentados a definição e os componentes de um problema geral de otimização. A Seção 2.2 descreve conceitos e dificuldades envolvendo otimização combinatória, sua classificação e metodologias aplicáveis em sua resolução.

2.1 Problemas e Técnicas de Otimização

Otimização é o processo de buscar a melhor solução (ou solução ótima) dentre um conjunto de soluções disponíveis para um problema. A otimização pode ser dividida em duas classes: global e local, sendo que a otimização global encontra a melhor solução do conjunto entre todas as soluções possíveis e, a otimização local, encontra uma solução que não é a melhor de todas, mas se apresenta como a melhor dentro de um subconjunto do conjunto universo (Bazaraa et al., 2006).

Tipicamente, um problema de otimização possui três elementos constituintes: variáveis de decisão (parâmetros cujos valores definem uma solução para o problema), função objetivo (uma função das variáveis de decisão a ser minimizada ou maximizada) e restrições (um conjunto de funções que define o espaço de soluções factíveis). Matematicamente, um problema de otimização pode ser enunciado como se segue:

$$\begin{array}{lll}
\text{Minimizar ou Maximizar} & f(x) & x \in R^n \\
\text{Sujeito a:} & c_i(x) = 0 & i = 1..k \\
& c_i(x) \leq 0 & i = k+1..m \\
& z^{mim} \leq x \leq z^{max}
\end{array}$$

onde x é o vetor de variáveis sobre o qual são impostos os limites mínimos e máximos (z^{mim} e z^{max} delimitam o domínio dos valores que as variáveis podem assumir), $f(x)$ é a função objetivo a ser minimizada ou maximizada, e as funções $c_i(x)$ são as restrições impostas pelo problema. A função objetivo e as restrições podem ser de natureza convexa ou não-convexa, de igualdade ou desigualdade, linear ou não-linear. As variáveis de decisão podem assumir valores reais ou inteiros. Um ponto que satisfaça todas as restrições é chamado de ponto factível e o espaço ou região, que contém todos os pontos que satisfaçam todas as restrições, é conhecido como região factível.

Como exemplo da utilização destes elementos, considera-se um sistema de produção. As variáveis de decisão podem representar as quantidades produzidas de determinados objetos; a função objetivo pode simbolizar o interesse em minimizar os custos na produção destes objetos; e as restrições podem estar relacionadas às limitações operacionais do processo de produção ou até mesmo às limitações físicas e tecnológicas.

Existe uma classe especial de problemas de otimização, comum nas áreas de pesquisa operacional e de engenharia, a qual possui como tarefa encontrar uma permutação ótima de algumas variáveis de decisão ou controle. Os problemas pertencentes a esta classe são conhecidos como problemas de otimização combinatória e são detalhados na Seção 2.2.

2.2 Otimização Combinatória

Um problema de otimização combinatória tem por objetivo determinar valores a um conjunto de variáveis de decisão, de tal modo que uma função dessas variáveis (função-objetivo) seja otimizada na presença de um conjunto de restrições. Formalmente, um problema de otimização combinatória é definido por meio de um conjunto finito $N=\{1,...,n\}$, com pesos c_j associados a cada $j \in N$, e um conjunto F formado por subconjuntos viáveis de N . Deseja-se determinar elementos de F , tais que o somatório dos pesos associados sejam ótimos. Uma solução é viável quando os valores atribuídos às variáveis não violam nenhuma restrição. O conjunto F é também chamado de espaço de busca de soluções (Wolsey, 1998).

Problemas práticos de engenharia elétrica, engenharia aeroespacial, economia e pesquisa operacional podem ser modelados usando grafos (Ziviani, 2005), e constituem-se em problemas de otimização combinatória. Tais problemas podem ser agrupados em cinco classes (Goldbarg e Luna, 2005). Maiores detalhes sobre cada um dos problemas mencionados podem ser encontrados em (Goldbarg e Luna, 2005).

1. Problemas de Conexão – Árvores, Caminhos e Emparelhamento: em inúmeras situações, o problema de otimização estará associado às alternativas de conexão entre vértices. Para este tipo de problema, o tomador de decisão precisa ser capaz de exibir uma topologia adequada ao modelo, de organizar as configurações desejáveis dentro desta topologia e de estabelecer critérios de escolhas destas configurações. Entre os problemas desta classe destacam-se os problemas da árvore geradora mínima, das N -rainhas, do caminho mínimo e do emparelhamento bipartido;
2. Problemas de Fluxo em Redes: abordam o processo de otimização da distribuição de “itens” originados em um vértice e consumidos em outros vértices, dentro de uma rede de interligações. Um processo de distribuição de produtos, por exemplo, não é realizado obrigatoriamente de um ponto de produção a um ponto de demanda, permitindo-se que pontos intermediários sejam utilizados. As interligações podem possuir restrições de capacidade de tráfego e custos variados. Alguns exemplos desta classe são os problemas de transporte, de alocação, problema do transbordo, etc.;
3. Problema do Caixeiro Viajante: dado um grafo qualquer, o objetivo é encontrar o caminho de menor custo que se inicie e termine em um vértice, passando por todos os demais vértices, sem repetição;
4. Problemas de Roteamento: têm por objetivo atender demandas localizadas nas arestas ou nos vértices de uma rede. Basicamente, determinam a sequência de visitas aos vértices, atendendo a uma função-objetivo. Tradicionalmente aplicado a questões operacionais, também pode ser usado para apoiar decisões estratégicas, por exemplo, a localização de fábricas ou depósitos, tipos de veículos, e decisões táticas, tais como número de rotas e de veículos, regime de trabalho e nível de estoque. Dentre os problemas desta classe destacam-se o problema do carteiro chinês, o problema de roteamento de veículos e o problema de escala da tripulação;
5. Problemas de Cobertura e Particionamento: o problema de cobertura pode estar relacionado aos vértices ou às arestas. É um caso de cobertura de vértices quando se

procura o menor conjunto de vértices tal que toda aresta do grafo incida em um destes vértices; sendo a cobertura de arestas quando se trata de obter o menor número de arestas onde todo vértice é incidido por ao menos uma destas arestas. O particionamento é um caso particular dos problemas de Cobertura, sendo que a diferença está na natureza das restrições. Um problema típico desta classe é o de localização de facilidades, tal como a localização de antenas e radares.

Uma característica comum dos problemas de otimização combinatória é o fato de que o conjunto de soluções viáveis, embora seja finito, pode ser muito grande. Tomando por exemplo o problema do Caixeiro Viajante, citado anteriormente, à medida que cresce o número de vértices, ou de cidades, a complexidade do problema aumenta exponencialmente. Para encontrar o número $R(n)$ de rotas para o caso de n cidades, basta fazer um raciocínio combinatório simples. No caso de 4 cidades, a primeira e última posição são fixas, de modo que elas não afetam o cálculo; na segunda posição, coloca-se qualquer uma das três cidades restantes, e uma vez escolhida uma delas, escolhe-se qualquer uma das duas restantes na terceira posição; na quarta posição não há nenhuma escolha, pois restou apenas uma cidade. Conseqüentemente, o número de rotas é $3 \times 2 \times 1 = 6$. Generalizando para o caso de n cidades, como a primeira é fixa, o número total de escolhas é $(n-1) \times (n-2) \times \dots \times 2 \times 1$, ou seja, $R(n) = (n-1)!$. Assim, se fossem atribuídos a n os valores 5, 10 e 25, ter-se-ia $R(n)$ igual a 24, 362.880 e $6,2 \times 10^{23}$ rotas, respectivamente. A partir destes valores é possível perceber o aumento exponencial das alternativas a serem avaliadas no problema do Caixeiro Viajante, e conseqüentemente, exigindo mais recursos computacionais, tais como memória e tempo de processamento.

Situações como esta inviabilizam a utilização de estratégias de força bruta, ou seja, testar todos os valores de solução possíveis, sendo necessário o desenvolvimento de métodos mais rápidos. Quando a estrutura do problema é complexa ou existe uma grande variedade de possíveis soluções, geralmente não há uma solução simples e diretamente calculável para o problema, havendo então a necessidade do uso de técnicas de otimização.

Para algumas classes de problemas de otimização combinatória existem métodos exatos de resolução, chamados de Programação Matemática. Outras classes levaram à necessidade de métodos não-exatos, tais como os Heurísticos e os baseados em Inteligência Artificial, uma vez que sua resolução exata é computacionalmente intratável (Ziviani, 2005). Além disso, muitas aplicações não exigem uma solução exata, tornando aceitável o uso de

métodos de solução de problemas que encontrem soluções aproximadas, mas com menor custo computacional.

As Seções 2.2.1, 2.2.2 e 2.2.3 descrevem de forma breve os métodos de Programação Matemática, Heurísticos e baseados em Inteligência Artificial, respectivamente.

2.2.1 Programação Matemática

A Programação Matemática é definida como o planejamento de atividades para obter um resultado ótimo entre as alternativas viáveis (Hillier e Lieberman, 2001), e visa criar e solucionar modelos quantitativos que podem ser expressos matematicamente e que representam algum processo. Os modelos de Programação Matemática são estruturados de uma forma lógica e amparados no ferramental matemático de representação, objetivando a determinação das melhores condições de funcionamento para os sistemas representados (Goldbarg e Luna, 2005). Assim, o termo “Programação”, constituinte do nome deste método, deve ser entendido no sentido de planejamento, o qual é adequado para expressar as atividades genéricas de programação de ações ou atividades.

Os métodos de Programação Matemática são agrupados de acordo com os tipos de problemas que resolvem. Alguns destes grupos são:

1. Programação Linear: trata dos casos em que as variáveis assumem valores contínuos e apresentam comportamento linear, tanto nas restrições quanto na função-objetivo;
2. Programação Não-Linear: aplicada em problemas nos quais as variáveis assumem valores contínuos e apresentam não-linearidades na função-objetivo ou em alguma das restrições;
3. Programação Linear Inteira: caso de Programação Linear em que as variáveis de decisão não podem assumir valores contínuos, devido à características do próprio problema;
4. Programação Dinâmica: aplicável a problemas de otimização discreta ou contínua, nos quais a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada de subproblemas que, sobrepostos, compõem o problema original. Este método e sua aplicabilidade serão detalhados a seguir, em função de terem sido aplicados a um dos estudos de caso realizados neste trabalho, e por terem sido também utilizados para comparação de resultados no Capítulo 6.

2.2.1.1 Programação Dinâmica

Programação dinâmica (PD) é uma técnica de construção de algoritmos para a resolução de problemas de otimização, em especial os de otimização combinatória, sendo utilizada para a otimização de processos de decisão multiestágios. Um *processo de decisão multiestágios* é aquele que pode ser desdobrado segundo um certo número de etapas sequenciais ou estágios. As alternativas incluídas na conclusão de um estágio são denominadas *decisões*. A condição do processo dentro de cada estágio é denominada *estado*. Cada estágio inclui a tomada de uma decisão que pode ou não alterar o estado do processo, mas que, obrigatoriamente, representa uma transição entre o estado corrente e o estado futuro do processo (Goldbarg e Luna, 2005).

Baseado nesta definição, um problema de programação dinâmica típico pode ser modelado como um conjunto constituído de um nó fonte e um nó destino com n -estágios intermediários, e m -estados em cada estágio como ilustrado na Figura 2.1.

Figura 2.1: Um problema de programação dinâmica.

Em outras palavras, a programação dinâmica resolve problemas dividindo-os em subproblemas (estágios), combinando suas soluções e aproveitando soluções anteriormente obtidas. Desta forma, um algoritmo de programação dinâmica resolve cada subproblema uma única vez, e então grava sua resposta em uma tabela, evitando assim o trabalho de recalculá-la toda vez que o subproblema é encontrado.

Considere o problema de multiplicação de cadeias de matrizes descrito em (Cormen et al., 2002), sendo este um exemplo clássico em que um algoritmo de programação dinâmica

pode ser aplicado. Neste problema, existe uma seqüência $\langle A_1, A_2, \dots, A_n \rangle$ de n matrizes a serem multiplicadas, e deseja-se calcular o produto $A_1 A_2 \dots A_n$. Tal produto pode ser efetuado usando um algoritmo padrão para multiplicação de pares de matrizes, uma vez que os parênteses entre as matrizes tenham sido colocados, solucionando todas as ambigüidades no modo como as matrizes são multiplicadas entre si. A multiplicação de matrizes é associativa, e sendo assim, todas as colocações entre parênteses resultam no mesmo produto. Por exemplo, se a cadeia de matrizes é $\langle A_1, A_2, A_3, A_4 \rangle$, o produto $A_1 A_2 A_3 A_4$ pode ser colocado entre parênteses de cinco modos distintos: $(A_1(A_2(A_3 A_4)))$, $(A_1((A_2 A_3) A_4))$, $((A_1 A_2)(A_3 A_4))$, $((A_1(A_2 A_3)) A_4)$ e $((A_1 A_2) A_3) A_4$.

A forma como uma cadeia de matrizes é colocada entre parênteses tem um impacto significativo sobre o custo computacional resultante do produto destas matrizes. É possível multiplicar duas matrizes A e B somente se elas forem compatíveis, ou seja, se o número de colunas de A for igual ao número de linhas de B . Além disso, se A é uma matriz de dimensão $p \times q$ e B é uma matriz de dimensão $q \times r$, a matriz resultante C é uma matriz de dimensão $p \times r$. O custo para calcular C é definido por $p \cdot q \cdot r$, que denomina o número de multiplicações escalares necessárias para multiplicar A por B .

Para exemplificar os diferentes custos resultantes da colocação dos parênteses de um produto de matrizes de maneiras distintas, considere o problema de uma cadeia $\langle A_1, A_2, A_3 \rangle$. Suponha que as dimensões das matrizes sejam 10×100 , 100×5 e 5×50 , respectivamente. A multiplicação das matrizes de acordo com a colocação dos parênteses $((A_1 A_2) A_3)$, executa $10 \cdot 100 \cdot 5 = 5000$ multiplicações escalares para calcular o produto entre as matrizes $A_1 A_2$, resultando em uma matriz resposta A_{r1} de dimensão 10×5 . Multiplicando A_{r1} por A_3 , executam-se $10 \cdot 5 \cdot 50 = 2500$ multiplicações escalares, produzindo-se um total de 7500 multiplicações escalares. Se, em vez disso, a escolha pela colocação dos parênteses tivesse sido feita $(A_1(A_2 A_3))$, seriam executadas $100 \cdot 5 \cdot 50 = 25000$ multiplicações escalares para calcular o produto entre as matrizes A_2 e A_3 , resultando uma matriz A_{r2} de dimensão 100×50 , mais outras $10 \cdot 100 \cdot 50 = 50000$ multiplicações escalares para multiplicar A_1 por A_{r2} , totalizando-se de 75000 multiplicações escalares. Desse modo, o cálculo do produto de acordo com a primeira colocação dos parênteses é 10 vezes mais rápido.

Observa-se que no problema de multiplicação de cadeias de matrizes a preocupação não está em realmente multiplicar matrizes, mas sim em determinar uma ordem para multiplicar as matrizes de forma a obter um custo computacional mais baixo, ou seja, o tempo

investido para determinar essa ordem ótima é compensado pelo tempo economizado posteriormente, quando as multiplicações de matrizes são executadas de fato.

O desenvolvimento de um algoritmo de programação dinâmica pode ser desmembrado em uma sequência de quatro etapas (Cormem et al., 2002):

1. Caracterizar a estrutura de uma solução ótima;
2. Definir recursivamente o valor de uma solução ótima;
3. Calcular o valor de uma solução ótima em um processo de baixo para cima (*bottom-up*);
4. Construir uma solução ótima a partir de informações calculadas.

Para ilustrar o desenvolvimento de um algoritmo de programação dinâmica, a seguir serão detalhadas cada uma das quatro etapas, considerando o problema de multiplicação de cadeias de matrizes descrito em Cormem et al. (2002), apresentado anteriormente.

Algoritmo de PD - Etapa 1 - Estrutura de Colocação Ótima dos Parênteses

A primeira etapa do paradigma de programação dinâmica é encontrar a subestrutura ótima, e depois usá-la para construir uma solução ótima para o problema a partir de soluções ótimas para subproblemas. Por conveniência, adota-se a notação $A_{i..j}$, onde $i \leq j$ para a matriz que resulta do produto $A_i A_{i+1} \dots A_j$. Nota-se que, se a solução não é trivial, isto é, se $i < j$, qualquer colocação ótima dos parênteses do produto $A_i A_{i+1} \dots A_j$ deve dividir o produto em A_k e A_{k+1} para algum inteiro k no intervalo $i \leq k < j$. Ou seja, para algum k , primeiro calculam-se as matrizes $A_{i..k}$ e $A_{k+1..j}$, e depois multiplicam-se os dois resultados para gerar o produto final $A_{i..j}$. O custo desta colocação dos parênteses é, portanto, o custo de calcular a matriz $A_{i..k}$, mais o custo de calcular $A_{k+1..j}$, mais o custo de multiplicá-las uma pela outra.

A subestrutura ótima deste problema é dada como se segue. Suponha que uma colocação dos parênteses de $A_i A_{i+1} \dots A_j$ divida o produto em A_k e A_{k+1} . Então, a colocação dos parênteses na subcadeia $A_i A_{i+1} \dots A_k$ dentro dessa colocação ótima dos parênteses de $A_i A_{i+1} \dots A_j$, deve ser uma colocação ótima dos parênteses de $A_i A_{i+1} \dots A_k$. A mesma observação é válida para a colocação dos parênteses da subcadeia $A_{k+1} A_{k+2} \dots A_j$. Desta maneira, pode-se construir uma solução ótima para uma instância do problema de multiplicação de cadeias de matrizes dividindo o problema em dois subproblemas, por meio da colocação ótima dos parênteses de

$A_i A_{i+1} \dots A_k$ e $A_{k+1} A_{k+2} \dots A_j$, encontrando-se soluções ótimas para instâncias de subproblemas, e depois combinando essas soluções ótimas de subproblemas. Quando se procura pelo lugar correto para dividir o produto, é preciso assegurar-se de que todas as posições estão sendo consideradas, de forma a se ter a garantia de ter examinado a opção ótima. Este princípio, de que em uma seqüência ótima de escolhas ou decisões, cada subsequência deve ser ótima, é chamado de *princípio da otimalidade* e foi proposto em Bellman (1957).

De acordo com o princípio da otimalidade, dado um estado atual, uma política ótima para as etapas restantes é independente da política adotada nas etapas anteriores. Para os problemas de programação dinâmica em geral, o conhecimento do estado atual abrange toda a informação, sobre seu comportamento anterior, necessária para determinar a política ótima doravante (Hillier e Lieberman, 2001). Em outras palavras, o princípio da otimalidade tem a propriedade de qualquer que seja o estado inicial e a decisão inicial tomada, as decisões subsequêntes devem constituir uma política ótima em direção ao estado final a partir da primeira decisão (Bellman, 1957).

Algoritmo de PD - Etapa 2 - Uma Solução Recursiva

Na segunda etapa, deve-se definir recursivamente o custo de uma solução ótima em termos das soluções ótimas para subproblemas. Para isso, os subproblemas são escolhidos, e neste caso, se tratam dos subproblemas de determinar o custo mínimo de uma colocação dos parênteses de $A_i A_{i+1} \dots A_j$, para $1 \leq i \leq j \leq n$. Seja $m[i, j]$ o número mínimo de multiplicações escalares necessárias para calcular a matriz $A_{i..j}$; para o problema completo, o custo de um caminho “mais econômico” para calcular $A_{i..n}$ seria portanto, $m[1, n]$.

Na definição recursiva de $m[i, j]$, se $i=j$, o problema é trivial, ou seja, a cadeia consiste em apenas uma matriz $A_{i..i} = A_i$, e assim, nenhuma multiplicação escalar é necessária para calcular o produto. Desse modo, $m[i, i] = 0$ para $i = 1, 2, \dots, n$. Para calcular $m[i, j]$ quando $i < j$, aproveita-se a estrutura de uma solução ótima da *Etapa 1*. Supõe-se que a colocação ótima dos parênteses divida o produto $A_i A_{i+1} \dots A_j$ em A_k e A_{k+1} , onde $i \leq k < j$. Então, $m[i, j]$ é igual ao custo mínimo para calcular os subprodutos $A_{i..k}$ e $A_{k+1..j}$, mais o custo de multiplicar essas duas matrizes. Considerando que cada matriz A_i tem dimensão $p_{i-1} p_i$, tem-se que o cálculo do produto de matrizes de $A_{i..k}$ e $A_{k+1..j}$ exige $p_{i-1} p_k p_j$ multiplicações escalares. Desta forma, obtém-se que $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$. Essa equação recursiva pressupõe que o valor de k é conhecido, o que não é verdade. Porém, existe apenas $j-i$ valores possíveis para k ,

isto é, $k = i, i+1, \dots, j-1$. Como a colocação ótima dos parênteses deve usar um destes valores para k , é preciso verificar cada um deles para encontrar o melhor. Assim, uma definição recursiva para o custo mínimo de colocar entre parênteses o produto $A_i A_{i+1} \dots A_j$ se torna:

$$m[i, j] = \begin{cases} 0, & \text{se } i = j, \\ \min_{i \leq k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\}, & \text{se } i < j. \end{cases} \quad (2.1)$$

Os valores de $m[i, j]$ fornecem os custos de soluções ótimas para subproblemas. Para auxiliar a construção de uma solução ótima, uma matriz $s[i, j]$ é definida, armazenando valores de k nos quais é possível dividir o produto $A_i A_{i+1} \dots A_j$ para obter uma colocação ótima dos parênteses.

Algoritmo de PD - Etapa 3 - Como Calcular os Custos Ótimos

Nesta etapa, o cálculo do custo ótimo deve ser realizado utilizando uma abordagem de baixo para cima (*bottom-up*). O Algoritmo 2.1 (Cormem et al., 2002) pressupõe que a matriz A_i tem dimensões $p_{i-1} p_i$, para $i = 1, 2, \dots, n$. A entrada é uma seqüência $p = \langle p_0, p_1, \dots, p_n \rangle$, com $\text{comprimento}[p] = n + 1$. O procedimento utiliza uma tabela principal $m[1..n, 1..n]$ para armazenar os custos de $m[i, j]$ e uma tabela auxiliar $s[1..n, 1..n]$ que registra qual índice de k alcançou o custo ótimo no cálculo de $m[i, j]$. Esta tabela auxiliar será usada posteriormente para a construção de uma solução ótima.

Para implementar corretamente a abordagem de baixo para cima, as entradas da tabela que são usadas para calcular $m[i, j]$ devem ser determinadas. A Equação (2.1) mostra que o custo $m[i, j]$ de calcular um produto de cadeias de matrizes para $j-i+1$ matrizes depende apenas dos custos de se calcular os produtos de cadeias de matrizes de menos de $j-i+1$ matrizes. Ou seja, para $k = i, i+1, \dots, j-1$, a matriz $A_{i..k}$ é um produto de $k-i+1 < j-i+1$ matrizes, e a matriz $A_{k+1..j}$ é um produto de $j-k < j-i+1$ matrizes. Assim, o algoritmo deve preencher a tabela $m[i, j]$ de uma forma que corresponda a resolver o problema de colocação dos parênteses em cadeias de matrizes de comprimento crescente.

```

ORDEN_MATRIZES (p)
1    $n \leftarrow \text{comprimento}[p] - 1$ 
2   para  $i \leftarrow 1$  até  $n$ 
3       faça  $m[i, i] \leftarrow 0$ 
4   para  $t \leftarrow 2$  até  $n$ 
5       faça para  $i \leftarrow 1$  até  $n-t+1$ 

```

```

6          faça  $j \leftarrow i+t-1$ 
7           $M[i,j] \leftarrow \infty$ 
8          para  $k \leftarrow i$  até  $j-1$ 
9              faça  $q \leftarrow m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$ 
10             se  $q < m[i,j]$ 
11                 então  $m[i,j] \leftarrow q$ 
12                  $s[i,j] \leftarrow k$ 
13 retorna  $m$  e  $s$ 

```

Algoritmo 2.1: Algoritmo de programação dinâmica (Cormem et al., 2002).

O Algoritmo 2.1 calcula primeiro $m[i, i] \leftarrow 0$ para $i = 1, 2, \dots, n$ (os custos mínimos para cadeias de comprimento 1) nas linhas 2 e 3. Em seguida, o mesmo usa a recorrência da Equação (2.1) para calcular $m[i, i+1]$ para $i = 1, 2, \dots, n-1$ (custos mínimos para cadeias de comprimento $t = 2$), durante a primeira execução do *loop* nas linhas 4 a 12. Na segunda passagem através do *loop*, ele calcula $m[i, i+2]$ para $i = 1, 2, \dots, n-2$ (custos mínimos para cadeias de comprimento $t = 3$), e assim por diante. Em cada etapa, o custo $m[i, j]$ calculado nas linhas 9 a 12 depende apenas das entradas $m[i, k]$ e $m[k+1, j]$ já calculadas na tabela.

A Figura 2.2 mostra o resultado da aplicação do algoritmo em uma cadeia de $n = 6$ matrizes $(A_1, A_2, \dots, A_6)$, com dimensões dadas respectivamente por 30x35, 35x15, 15x5, 5x10, 10x20 e 20x25. Tendo em vista que $m[i, j]$ é definida somente para $i < j$, apenas a parte da tabela m estritamente acima da diagonal principal é usada. O algoritmo preenche a tabela $m[i, j]$ da diagonal principal para as diagonais superiores a ela, de cima para baixo e da esquerda para a direita.

Para ilustrar o funcionamento do algoritmo ao preencher a tabela e para mostrar a utilização de uma solução previamente obtida, o cálculo do termo $m[1,3]$ será apresentado. Para obter este termo, as variáveis t, i, j e k estão com os respectivos valores 3, 1, 3, e 1. Assim, a equação expressa na linha 9 pode ser construída substituindo as variáveis por seus valores, como segue:

$$q = m[1,3] \leftarrow m[1,1] + m[2,3] + 30.35.5$$

onde,

$m[1,3]$ é o custo de multiplicar as matrizes $A_1..A_3$;

$m[1,1]$ é uma cadeia composta apenas pela matriz A_1 (possui custo zero, ou seja, nenhuma multiplicação escalar é necessária);

$m[2,3]$ é o custo de multiplicar as matrizes $A_2..A_3$, e já havia sido calculado e armazenado na tabela em iteração anterior, não sendo necessário recalculá-lo;

$30.35.5 = p_0.p_1.p_3$ é o custo de multiplicar a matriz A_1 pela matriz resposta resultante de (A_2A_3) .

		j					
i	m	1	2	3	4	5	6
	1	0	15750	7875	9375	11875	15125
	2		0	2625	4375	7125	10500
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

		j					
i	s	1	2	3	4	5	6
	1		1	1	3	3	3
	2			2	3	3	3
	3				3	3	3
	4					4	5
	5						5
	6						

Figura 2.2: Resultado da aplicação do algoritmo de programação dinâmica.

Algoritmo de PD - Etapa 4 - A Construção de Uma Solução Ótima

Embora o Algoritmo 2.1 determine o número ótimo de multiplicações escalares para calcular um produto de cadeias de matrizes, ele não mostra diretamente a ordem como as matrizes devem ser multiplicadas. Para determinar mais facilmente esta ordem, a matriz $s[i,j]$ deve ser utilizada. Cada entrada da matriz s registra o valor de k tal que a colocação ótima dos parênteses de $A_iA_{i+1}...A_j$ divide o produto entre A_k e A_{k+1} . Deste modo, a multiplicação final de matrizes no cálculo ótimo de $A_1..A_n$ é $A_{1..s[1,n]}A_{s[1,n]+1..n}$. As multiplicações de matrizes anteriores podem ser calculadas recursivamente, pois $s[1,s[1,n]]$ determina a última multiplicação de matrizes no cálculo de $A_{1..s[1,n]}$, e $s[s[1,n]+1,n]$ determina a última multiplicação de matrizes no cálculo de $A_{s[1,n]+1..n}$.

O Algoritmo 2.2 a seguir, retirado de (Cormem et al., 2002), imprime uma colocação ótima dos parênteses de $\langle A_i, A_{i+1}, \dots, A_j \rangle$, dada a tabela s calculada pelo Algoritmo 2.1 e os índices i e j .

```

IMPRIME_PARENTESES ( $s, i, j$ )
1   se  $i=j$ 
2       então imprime " $A$ " $i$ 
3       senão imprime "("
4           IMPRIME_PARENTESES ( $s, i, s[i, j]$ )
5           IMPRIME_PARENTESES ( $s, s[i, j]+1, j$ )
6       imprime ")"

```

Algoritmo 2.2: Algoritmo de impressão da colocação ótima dos parênteses na multiplicação de matrizes.

No exemplo da Figura 2.2, a chamada IMPRIME_PARENTESES ($s, 1, 6$) imprime a seguinte colocação dos parênteses: $((A_1(A_2A_3))((A_4A_5)A_6))$.

2.2.1.2 Aplicabilidade da Programação Dinâmica

Para que um problema de otimização combinatória possa ser resolvido por programação dinâmica, ele deve possuir duas características: subestrutura ótima (*princípio da otimalidade*) e superposição de subproblemas (Cormem et al., 2002).

Subestrutura Ótima

Um problema apresenta uma subestrutura ótima se uma solução para o problema contém em seu interior soluções ótimas para subproblemas. Sempre que um problema apresenta subestrutura ótima, esse fato é uma boa indicação de que a programação dinâmica poderia ser aplicada (Brassard e Bradley, 1996). No exemplo de multiplicação de cadeias de matrizes apresentado na Seção 2.2.1.1, foi possível observar que a colocação ótima dos parênteses de $A_iA_{i+1}\dots A_j$ que divide o produto em A_k e A_{k+1} contém dentro dela soluções ótimas para os problemas de colocação dos parênteses de $A_iA_{i+1}\dots A_k$ e $A_{k+1}A_{k+2}\dots A_j$.

Existe um padrão comum a ser seguido na descoberta de uma subestrutura ótima (Cormem et al., 2002), ou seja:

1. Mostrar que uma solução para o problema consiste em fazer uma escolha, como a escolha de um índice no qual será dividida a cadeia de matrizes. Esta escolha deixa um ou mais subproblemas a serem resolvidos;

2. Supor que, para um dado problema, existe uma escolha que conduz a uma solução ótima;
3. Dada esta escolha, determinar quais subproblemas resultam dela e como caracterizar melhor o espaço de subproblemas resultante;
4. Mostrar que as soluções para os subproblemas usados dentro da solução ótima para o problema devem elas próprias ser ótimas, usando uma técnica de “recortar e colar”. Isso é feito supondo-se que cada uma das soluções de subproblemas não é ótima, e então, derivando uma contradição. Em particular, “recortando” a solução não ótima para o subproblema e “colando” a solução ótima, mostra-se que é possível conseguir uma solução melhor para o problema original, contradizendo a hipótese de que a solução já era ótima.

Para caracterizar o espaço de subproblemas, uma boa regra prática é tentar manter o espaço tão simples quanto possível, e depois expandi-lo conforme necessário. Por exemplo, a tentativa de restringir o espaço de subproblemas para a multiplicação de cadeias de matrizes a produtos de matrizes da forma $A_1A_2\dots A_j$. Como antes, uma colocação ótima dos parênteses deve dividir esse produto em A_k e A_{k+1} para algum $1 \leq k \leq j$. A menos que fosse possível garantir que k sempre é igual a $j-1$, descobrir-se-ia que haveria problemas da forma $A_1A_2\dots A_k$ e $A_{k+1}A_{k+2}\dots A_j$, e que o último subproblema não tem a forma $A_1A_2\dots A_j$. Para esse problema, seria necessário permitir que os subproblemas variassem em ambas as extremidades, isto é, permitir que tanto i quanto j variassem no subproblema $A_iA_{i+1}\dots A_j$.

A subestrutura ótima varia no domínio de subproblemas de duas maneiras:

1. O número de subproblemas que são usados em uma solução ótima para o problema original;
2. O número de opções que se tem na determinação de quais subproblemas usar em uma solução ótima.

A multiplicação de cadeias de matrizes para a subcadeia $A_iA_{i+1}\dots A_j$ serve como um exemplo com dois subproblemas e $j-i$ opções. Para uma dada matriz A_k em que o produto é dividido, têm-se dois subproblemas: (1) a colocação dos parênteses de $A_iA_{i+1}\dots A_k$, e (2) a colocação dos parênteses de $A_{k+1}A_{k+2}\dots A_j$; e ambos devem ser resolvidos de forma ótima. Uma vez determinadas as soluções ótimas para subproblemas, escolhe-se entre $j-i$ candidatos para o índice k .

Deve-se ter cuidado para não presumir que a subestrutura ótima é aplicável quando ela não é. Considere os dois problemas a seguir, nos quais existe um grafo orientado $G = (V, E)$ e vértices $u, v \in V$.

Problema 1 - Caminho mais curto não-ponderado: encontrar um caminho de u para v consistindo no menor número de arestas. Tal caminho deve ser simples, pois a remoção de um ciclo de um caminho produz um caminho com menos arestas. Nesta tese, um caminho é considerado *simples* se em uma seqüência de vértices de um grafo não há repetição de vértices.

Problema 2 - Caminho simples mais longo não-ponderado: encontrar um caminho simples de u para v consistindo no maior número de arestas. É preciso incluir o requisito de simplicidade porque, do contrário, seria possível percorrer um ciclo tantas vezes quanto desejar para criar caminhos com um número arbitrariamente grande de arestas.

O Problema 1, do caminho mais curto não-ponderado, apresenta subestrutura ótima, como mostrado a seguir. Suponha $u \neq v$, e assim o problema é não trivial. Então, qualquer caminho p de u para v deve conter um vértice intermediário, como por exemplo, w , sendo que w pode ser u ou v . Deste modo, pode-se decompor o caminho $u \xrightarrow{p} v$ em subcaminhos $u \xrightarrow{p_1} w \xrightarrow{p_2} v$. É claro que o número de arestas em p é igual à soma do número de arestas em p_1 e do número de arestas em p_2 . Se p é um caminho ótimo, ou seja, o caminho mais curto de u para v , então p_1 deve ser um caminho mais curto de u para w . Para se mostrar isso, usa-se o argumento de “recortar e colar”: se existisse outro caminho, digamos p_1' , de u para w com menos arestas que p_1 , então seria possível recortar p_1 e colar em p_1' para produzir um caminho $u \xrightarrow{p_1'} w \xrightarrow{p_2} v$ com menos arestas que p , contradizendo deste modo o caráter ótimo de p . Simetricamente, p_2 deve ser um caminho mais curto de w para v . Assim, pode-se encontrar um caminho mais curto de u para v considerando todos os vértices intermediários w , encontrando um caminho mais curto de u para w e um caminho mais curto de w para v , e escolhendo um vértice intermediário w que produza o caminho mais curto global.

Já o Problema 2, do caminho simples mais longo não-ponderado, não possui subestrutura ótima. Existe uma tendência em se pensar que fazendo a decomposição de um caminho simples mais longo $u \xrightarrow{p} v$ em subcaminhos $u \xrightarrow{p_1} w \xrightarrow{p_2} v$, então p_1 deve ser um caminho simples mais longo de u para w e p_2 deve ser um caminho simples mais longo

de u para v , mas isso não é verdade. Para ilustrar o equívoco, observe o exemplo dado pela Figura 2.3 (Cormen et al., 2002).

Figura 2.3: Caminho simples mais longo.

Esse exemplo mostra que encontrar um caminho simples mais longo em um grafo orientado não-ponderado não tem subestrutura ótima, pois o caminho $q \longrightarrow r \longrightarrow t$ é um caminho simples mais longo de q para t , mas o subcaminho $q \longrightarrow r$ não é um caminho simples mais longo de q para r , nem o subcaminho $r \longrightarrow t$ é um caminho mais longo de r para t . Além disso, para caminhos simples mais longos, não apenas falta uma subestrutura ótima, mas também não é possível montar uma solução válida para o problema a partir de soluções para subproblemas. Combinando os caminhos simples mais longos $q \longrightarrow s \longrightarrow t \longrightarrow r$ e $r \longrightarrow q \longrightarrow s \longrightarrow t$, obtém-se o caminho $q \longrightarrow s \longrightarrow t \longrightarrow r \longrightarrow q \longrightarrow s \longrightarrow t$, que não é simples.

Embora sejam usados dois subproblemas em uma solução para um problema tanto de caminhos mais longos quanto de caminhos mais curtos, os subproblemas na localização do caminho simples mais longo não são independentes, enquanto para caminhos mais curtos sim. Subproblemas independentes significam que a solução para um subproblema não afeta a solução para outro subproblema do mesmo problema. No exemplo da Figura 2.3, existe o problema de encontrar um caminho simples mais longo de q para t com dois subproblemas: encontrar caminhos simples mais longos de q para r e de r para t . Para o primeiro destes subproblemas, escolhe-se o caminho $q \longrightarrow s \longrightarrow t \longrightarrow r$, e assim também os vértices s e t são usados. Não se pode mais usar estes vértices no segundo subproblema, pois a combinação das duas soluções para subproblemas produziria um caminho que não é simples. Se não é permitido usar o vértice t no segundo subproblema, então não é possível resolvê-lo, pois t tem que estar no caminho encontrado, e ele não é o vértice em que as soluções de subproblemas estão sendo reunidas (esse vértice é o r). O uso dos vértices s e t em uma solução de subproblema impede que eles sejam usados na solução de outro subproblema. Porém, deve ser usado pelo menos um deles para resolver o outro subproblema, e é preciso

utilizar ambos para resolvê-lo de forma ótima. Desta maneira, estes subproblemas não são independentes. Visto de outro modo, o uso de recursos na resolução de um subproblema (sendo esses recursos os vértices) tornou-os indisponíveis para o outro subproblema.

Já no caso do problema do caminho mais curto, os subproblemas não compartilham recursos. Se um vértice w está em um caminho mais curto p de u para v , então se pode reunir qualquer caminho mais curto $u \xrightarrow{p_1} w$ e qualquer caminho $w \xrightarrow{p_2} v$ para produzir um caminho mais curto de u para v . É assegurado que, além de w , nenhum vértice pode aparecer em ambos os caminhos p_1 e p_2 . Esta afirmação pode ser explicada supondo que algum vértice $x \neq w$ apareça tanto em p_1 quanto em p_2 , de forma que seja possível decompor p_1 como $u \xrightarrow{p_{ux}} x \longrightarrow w$ e p_2 como $w \longrightarrow x \xrightarrow{p_{xv}} v$. Pela subestrutura ótima desse problema, o caminho p tem tantas arestas quanto p_1 e p_2 juntos; suponha que p tem e arestas. Constrói-se, então, um caminho $u \xrightarrow{p_{ux}} x \xrightarrow{p_{xv}} v$ de u para v . Esse caminho tem no máximo $e-2$ arestas, o que contradiz a hipótese de que p é um caminho mais curto. Deste modo, é possível garantir que os subproblemas para o problema do caminho mais curto são independentes.

No exemplo da multiplicação de cadeias de matrizes, os subproblemas são multiplicar subcadeias de $A_i A_{i+1} \dots A_k$ e $A_{k+1} A_{k+2} \dots A_j$. Essas subcadeias são disjuntas, de forma que nenhuma matriz teria possibilidade de ser incluída em ambas.

Superposição de Subproblemas

A segunda característica que um problema de otimização combinatória deve ter para que a programação dinâmica seja aplicável é que o espaço de subproblemas seja pequeno, no sentido de que um algoritmo recursivo para o problema resolva os mesmos subproblemas repetidas vezes, em lugar de sempre gerar novos subproblemas. Quando um algoritmo recursivo reexamina o mesmo problema inúmeras vezes, diz-se que o problema de otimização tem subproblemas superpostos. Com isso, os algoritmos de programação dinâmica tiram proveito desta superposição, resolvendo cada subproblema uma única vez, e depois armazenando a solução em uma tabela, na qual ela possa ser consultada quando necessário.

Para ilustrar a propriedade de subproblemas superpostos, o exemplo da multiplicação de cadeias de matrizes será retomado. Consultando novamente a Figura 2.2, observa-se que o Algoritmo 2.1 procura repetidamente a solução para subproblemas em diagonais inferiores quando resolve subproblemas de diagonais superiores. Por exemplo, a entrada $m[3,4]$ é

referenciada quatro vezes: durante os cálculos de $m[2,4]$, $m[1,4]$, $m[3,5]$ e $m[3,6]$. Se $m[3,4]$ fosse recalculado a cada vez, em vez de apenas ser pesquisado, haveria um aumento significativo no tempo de execução.

2.2.2 Métodos Heurísticos

Conforme já descrito na Seção 2.1, a otimização é o processo de buscar a melhor solução dentre um conjunto de soluções disponíveis para um determinado problema. Ao contrário das estratégias de busca cega, tais como a busca em largura e em profundidade, que não consideram as informações de onde o alvo (ou objetivo) da busca está, e também não conseguem “ver” para onde estão indo (Poole et al., 1998), os métodos heurísticos utilizam informações heurísticas, dada por uma função heurística (Poole et al., 1998), para auxiliá-los na busca, ou seja, esta função heurística tem por objetivo indicar a direção mais promissora para alcançar o alvo. Como as heurísticas são informações particulares para cada tipo de problema, é preciso ter conhecimento específico do mesmo, sendo que o sucesso da busca pela solução do problema está fortemente baseado na experiência de um especialista na área do problema.

Os métodos heurísticos podem ser divididos em três classes que diferem basicamente na forma como exploram o espaço de soluções dos problemas (Arroyo, 2002).

A primeira classe de heurísticas é chamada Construtiva. Estas heurísticas são especializadas para um dado problema e constroem uma solução pela adição de componentes da mesma através de regras específicas associadas com a estrutura do problema (Arroyo, 2002).

A segunda classe de heurísticas é chamada Busca Local ou Busca em Vizinhança. Estas heurísticas iniciam com uma solução completa do problema, e constroem uma vizinhança desta solução que contém todas as soluções alcançáveis através de uma regra de movimento que modifica a solução inicial. Dessa vizinhança, escolhe-se uma solução que possua uma avaliação melhor que a solução inicial. A solução escolhida torna-se a nova solução inicial e o processo continua até encontrar um ótimo local. Claramente, a eficiência das heurísticas de busca local depende da escolha da solução inicial e da definição de uma vizinhança que estabeleça uma relação entre as soluções no espaço de decisões. Uma vez tendo chegado ao ótimo local, essas heurísticas param e não são capazes de escapar da otimalidade local e explorar novas regiões do espaço de busca (Arroyo, 2002).

A terceira classe de heurísticas é chamada Meta-heurística, que são métodos flexíveis, pois possuem uma estrutura com componentes genéricos que são adaptados ao problema que se quer resolver. Estes métodos possuem certa facilidade em incorporar novas situações e exploram o espaço de soluções permitindo a escolha estratégica de soluções piores que as já encontradas, na tentativa de superar a otimalidade local. Existem várias meta-heurísticas que apresentam princípios e estratégias distintas, destacando-se a Busca Tabu e o *Simulated Annealing*. Tais métodos exploram uma vizinhança a cada iteração de acordo com suas estratégias e escolhem apenas um elemento dessa vizinhança a cada passo. Esse tipo de varredura do espaço de busca gera uma trajetória de soluções obtida pela transição de uma solução para outra de acordo com os movimentos permitidos pelo método (Arroyo, 2002).

2.2.3 Métodos de Inteligência Artificial

Inteligência Artificial pode ser definida como o ramo da ciência da computação que se ocupa da automação do comportamento inteligente (Luger, 2004). Em outras palavras, pode-se dizer que ao tentar automatizar um comportamento inteligente por meio de um programa de computador, este programa é considerado inteligente somente quando realiza uma tarefa, sendo que se a mesma fosse feita por um ser humano, seria também considerada inteligente. No entanto, como o termo “inteligência” ainda é motivo para grandes debates, no que se refere, por exemplo, como definir a inteligência humana, como um sistema computacional pode ser considerado inteligente, entre outros aspectos relacionados à construção de sistemas computacionais inteligentes, optou-se então aqui por não aprofundar nestas questões, pelo motivo de não ser o foco principal deste trabalho.

Diante dos vários paradigmas existentes atualmente que são utilizados no desenvolvimento de sistemas possivelmente inteligentes, somente as Redes Neurais Artificiais e os Algoritmos Genéticos serão abordados com maiores detalhes no Capítulo 3, pelo fato de que a abordagem híbrida proposta nesta tese é constituída por uma rede neural artificial e por um algoritmo genético.

2.3 Considerações Finais

Neste capítulo foram apresentados os problemas de otimização combinatória, os quais foram classificados em cinco grupos de acordo com as características específicas de cada tipo de problema, ou seja: i) problemas de conexão, ii) problemas de fluxo em redes, iii) problema

do caixeiro viajante, iv) problemas de roteamento e v) problemas de cobertura e particionamento.

Para a resolução de problemas de otimização, uma descrição sucinta dos métodos heurísticos e de programação matemática foi realizada. A técnica de programação dinâmica foi descrita com mais detalhes, pois um dos problemas considerados nesta tese é modelado como um problema de programação dinâmica.

Capítulo 3

Redes Neurais Artificiais e Algoritmos Genéticos

Este capítulo apresenta os fundamentos básicos de dois paradigmas amplamente usados em Sistemas Inteligentes, que são as Redes Neurais Artificiais e os Algoritmos Genéticos. Na Seção 3.1 é apresentada uma introdução às Redes Neurais Artificiais, sendo que o enfoque é dado à rede recorrente de Hopfield, pois esta é parte da abordagem neuro-genética proposta nesta tese. A Seção 3.2 descreve os princípios básicos e parâmetros de Algoritmos Genéticos.

3.1 Redes Neurais Artificiais

Uma rede neural artificial (RNA) é um processador paralelamente distribuído, constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso (Haykin, 1999). A estrutura das redes neurais foi desenvolvida a partir de modelos conhecidos de sistemas nervosos biológicos e do próprio cérebro humano. As unidades de processamento simples, denominadas neurônios artificiais, correspondem aos nós da rede e são modelos simplificados dos neurônios biológicos. Tais modelos foram obtidos a partir da análise da geração e propagação de impulsos elétricos pela membrana celular dos neurônios (Hodgkin e Huxley, 1952).

Os neurônios utilizados nos modelos de redes neurais artificiais realizam funções simples, como coletar os sinais existentes em suas entradas, agregá-los de acordo com sua função de entrada e produzir uma saída por intermédio de sua função de saída (ativação) inerente. O modelo de neurônio mais simples e que engloba as principais características de uma rede neural biológica, paralelismo e alta conectividade, foi proposto por McCulloch e Pitts (1943). Este modelo realiza a soma algébrica ponderada das entradas de um neurônio que, em seguida, serve como entrada para a função de ativação, determinando a saída da rede (Haykin, 1999).

Uma RNA é classificada pelo algoritmo de aprendizagem utilizado, pelo tipo ou função do neurônio presente e por sua arquitetura. O algoritmo de aprendizagem deve especificar as condições iniciais da rede e a metodologia de ajuste de seus parâmetros internos para se obter o desempenho desejado. A função de ativação determina o comportamento individual interno a cada neurônio. Se todos os neurônios têm o mesmo comportamento, a rede é dita Homogênea, caso contrário é chamada Heterogênea. Em relação à arquitetura, existem algumas que merecem destaque, tais como as redes neurais em camadas, as redes recorrentes e as redes de estrutura *lattice* ou reticulada, cada qual mais adaptada a determinadas categorias de problemas.

As redes neurais em camadas, como o próprio nome já diz, são estruturadas em camadas, cada uma contendo certo número de neurônios, sendo que os neurônios de uma camada têm suas entradas conectadas às saídas da camada anterior e suas saídas conectadas às entradas dos neurônios da camada posterior. A rede neural é dita totalmente conectada quando todos os neurônios de cada camada são conectados a todos os outros neurônios da camada adjacente posterior. Se, contudo, alguma conexão não existe, a rede neural é dita parcialmente conectada (Haykin, 1999). O algoritmo de treinamento propaga a informação da camada de entrada para a camada de saída, e depois realiza o caminho inverso para a atualização dos pesos sinápticos. Dentre as redes deste tipo destacam-se as redes *Perceptron* e Função de Base Radial. Tais redes são geralmente aplicadas à aproximação de funções, reconhecimento de padrões, identificação e controle.

Redes recorrentes são redes que possuem realimentação, ou seja, a saída da rede (ou mesmo de qualquer um de seus neurônios) é utilizada como entrada para ela mesma. Tais redes são utilizadas para modelar sistemas dinâmicos não-lineares, sendo em geral adequadas para tarefas como modelamento de dados de entrada-saída, memória associativa e previsão de séries temporais.

Por fim, as redes de estrutura *lattice* ou reticulada são constituídas por neurônios dispostos em vizinhanças geralmente unidimensionais ou bidimensionais, capazes de detectar características comuns nas entradas e agrupá-las segundo estas características. Estas redes realizam um modelamento entre um espaço n -dimensional (a dimensão do problema original) e o espaço de vizinhança da rede (unidimensional ou bidimensional). De forma simplificada, para cada entrada fornecida, o algoritmo de aprendizagem determina o neurônio mais próximo a ela, aproximando este neurônio e seus vizinhos da entrada em questão. Após certo número de iterações, os neurônios tendem a agrupar-se em torno das entradas fornecidas, sendo possível distinguir conjuntos de dados com características semelhantes. As redes de Kohonen destacam-se dentro deste tipo de rede neural, e são frequentemente empregadas em aplicações de agrupamento (*clustering*).

Como a arquitetura de rede neural utilizada neste trabalho foi a recorrente, apenas ela será apresentada com mais detalhes. Informações adicionais sobre as demais arquiteturas podem ser encontradas em Haykin (1999) e Braga et al. (2007).

Redes Recorrentes de Hopfield

Como definida em Hopfield (1984) e em Hopfield e Tank (1985), esta rede apresenta normalmente uma única camada com conexões realimentadas entre os nós, conforme mostrado na Figura 3.1. Na maioria dos casos, os nós (neurônios) são completamente interconectados, ou seja, todos os neurônios da rede são conectados a todos os outros e a si próprio. O processo de associação e retenção da informação é simulado por um comportamento dinâmico de um sistema altamente interconectado de elementos (neurônios) não-lineares com habilidades computacionais coletivas (Hopfield, 1982).

Figura 3.1: A rede neural de Hopfield convencional.

A equação que descreve a dinâmica para a rede neural de Hopfield contínua no tempo com N -neurônios é dada por:

$$\dot{u}_i(t) = \frac{\partial u_i}{\partial t} = -\eta \cdot u_i(t) + \sum_{j=1}^N T_{ij}(t) \cdot v_j(t) + i_i^b \quad (3.1)$$

$$v_i(t) = g_i(u_i(t)) \quad (3.2)$$

sendo que:

$u_i(t)$ é o estado corrente do i -ésimo neurônio;

$\dot{u}_i(t)$ é a derivada de u_i em relação ao tempo (sistema dinâmico);

$v_i(t)$ é a saída do i -ésimo neurônio;

T_{ij} é o peso da conexão entre o i -ésimo neurônio e o j -ésimo neurônio;

i_i^b é a entrada do i -ésimo neurônio (*input bias*);

$\eta \cdot u_i(t)$ é um termo de decaimento passivo;

$g_i(u_i(t))$ é a função de ativação do i -ésimo neurônio.

Desta forma, as respostas da rede de Hopfield são dinâmicas, isto é, aplica-se um conjunto de entradas; em seguida, as saídas v são calculadas e retro-alimentadas às entradas. A saída é então recalculada e o processo é repetido iterativamente. Essas iterações produzem mudanças nas saídas cada vez menores, até que todas as saídas não mais apresentem alterações (caso o sistema seja estável).

Na equação (3.2), $g_i(u_i(t))$ é uma função de ativação, monótona crescente, que limita a saída de cada neurônio em um intervalo pré-definido. Os três tipos básicos de função de ativação, geralmente empregados nas redes de Hopfield, são as seguintes: função *threshold* (sinal), função *piecewise-linear* (rampa-simétrica) e função *sigmoid* (logística ou tangente hiperbólica).

Uma versão discreta da rede neural de Hopfield é obtida quando $\eta=0$ e $T_{ij} = T_{ji}$ (Hopfield, 1984). Neste caso, os pontos de equilíbrio da rede correspondem aos valores de $v(t)$ que minimizam a função de energia da rede (função de Lyapunov). Se os neurônios forem alterados de forma assíncrona, a função de energia associada à rede de Hopfield é definida por:

$$E(t) = -\frac{1}{2} v(t)^T \cdot T \cdot v(t) - v(t)^T \cdot i^b \quad (3.3)$$

Assim sendo, esta energia é uma função objetivo a ser minimizada pela rede, ou seja, a atualização nos pesos feita pela rede de Hopfield até atingir um ponto de equilíbrio pode ser entendida como um procedimento que procura tornar a energia da rede cada vez menor. Tecnicamente, a rede de Hopfield aplicada à otimização manipula um sistema dinâmico, no qual sua função de energia (ou função de Lyapunov) caracteriza o comportamento da rede e representa o problema a ser resolvido (Wen et al., 2008). Adicionalmente, em Hopfield (1982) foi demonstrado que a condição suficiente para a estabilidade da rede é que a matriz de pesos T seja simétrica, ou seja, $T_{ij}=T_{ji}$ com $T_{ii}=0$, e que o sistema tenha uma função de Lyapunov. Mais detalhes sobre a estabilidade da rede de Hopfield podem ser encontrados em Haykin (2008).

Uma das razões principais para a popularidade da rede de Hopfield é a possibilidade de implementá-la em *hardware*. Quando Hopfield e Tank propuseram a rede contínua, eles também demonstraram que existe um *hardware* analógico equivalente muito simples (ver Figura 3.2). Neste *hardware*, os neurônios são modelados através de amplificadores operacionais com relação entrada-saída determinada pela função de ativação neuronal.

Cada neurônio possui um resistor R_i^{in} e um capacitor C_i^{in} de entrada que definem uma constante de tempo dos neurônios e estabelece a soma analógica integrativa das correntes de entrada vindas de outros neurônios da rede. Uma conexão entre dois neurônios é definida por uma condutância T_{ij} que conecta uma das duas saídas do amplificador j a entrada do amplificador i . Esta conexão é feita com um resistor de valor $R_{ij} = 1 / |T_{ij}|$. Se a sinapse (conexão) é excitatória ($T_{ij} > 0$), este resistor é conectado a saída normal (+) do amplificador j . Para uma sinapse inibitória ($T_{ij} < 0$), é conectada a saída inversa (−) do amplificador j . A matriz T define então a conectividade entre os amplificadores operacionais (neurônios). A corrente de entrada em qualquer neurônio i é a soma das correntes fluindo através do conjunto de resistores que conecta suas entradas às saídas dos outros neurônios.

Como ilustrado na Figura 3.2, este circuito também inclui uma corrente de entrada I_i fornecida externamente para cada neurônio. As equações que descrevem a evolução deste circuito em relação ao tempo são definidas por:

$$C_i \left(\frac{du_i}{dt} \right) = \sum_{j=1}^N T_{ij} V_j - \frac{u_i}{R_i} + I_i \quad (3.4)$$

sendo que $V_j = g_j(u_j)$.

Figura 3.2: Hardware analógico da rede de Hopfield.

Uma outra razão que amplamente atrai o uso da rede de Hopfield é a sua utilização como Memória Associativa (MA) ou Memória Endereçável pelo Conteúdo (*Content-Addressable Memory – CAM*).

A função primordial de uma MA é recuperar um padrão previamente armazenado na memória, em resposta à apresentação de uma versão incompleta ou ruidosa daquele padrão (Haykin, 2008). Para ilustrar o significado desta afirmativa, em Hopfield (1982) é apresentado a seguinte afirmação:

Suponha que um item armazenado na memória seja “H. A. Kramers & G. H. Wannier Physi Ver.60, 252 (1941)”. Uma MA deve ser capaz de recuperar este item inteiro da memória com base em uma informação parcial. A entrada “& Wannier (1941)” deve ser suficiente. Uma memória ideal pode lidar com erros e recuperar esta referência mesmo a partir da entrada “Wannier, (1941)”.

Uma propriedade importante de uma MA é, portanto, a habilidade de recuperar um padrão armazenado, dado um conjunto razoável do conteúdo de informação daquele padrão. Mais ainda, uma MA permite correção de erro no sentido de que ela pode ignorar informações inconsistentes nas sugestões apresentadas a ela.

3.2 Algoritmos Genéticos

Muitos problemas computacionais requerem a busca de uma solução considerando um alto número de possíveis soluções, chamado de espaço de busca. O termo espaço de busca refere-se a uma coleção de soluções candidatas para um determinado problema (Mitchell, 1996). Um exemplo é encontrar por meio de um algoritmo, entre um vasto número de possíveis seqüências de aminoácidos, a estrutura de uma proteína com propriedades específicas. Outro exemplo é buscar uma forma para construir um conjunto de regras ou equações que predizem os altos e baixos de um mercado financeiro, tal como o valor de uma moeda estrangeira.

O uso do paralelismo para a resolução de tais problemas de busca pode ser uma boa opção, pois várias possibilidades são exploradas simultaneamente. Por exemplo, no problema da busca da proteína com propriedades específicas, ao invés de analisar uma seqüência de aminoácidos de cada vez, é muito mais rápido avaliar várias ao mesmo tempo. No entanto, para obter a eficiência do paralelismo, deve-se adotar uma estratégia inteligente para escolher qual será o próximo conjunto de seqüências de aminoácidos que será avaliado. Além do paralelismo, muitos problemas computacionais necessitam de algoritmos adaptativos, para serem aplicados em ambientes em constante alteração. Este é um problema típico de robótica, onde um robô precisa ser controlado para executar tarefas em tais ambientes. Outros problemas requerem algoritmos inovadores, que construam algo realmente novo e original (Mitchell, 1996).

Os Algoritmos Genéticos (AG) se mostram particularmente apropriados para lidar com problemas de busca porque incorporam todas estas características (busca da solução, paralelismo, adaptação e inovação). A aplicabilidade de algoritmos genéticos para a solução de problemas de otimização combinatória torna-se, portanto, muito atrativa, pois estes podem ser modelados como um problema de busca em um determinado espaço.

3.2.1 Princípios Básicos de Algoritmos Genéticos

Os Algoritmos Genéticos foram introduzidos por John Holland nos anos 60 e desenvolvidos por ele, seus alunos e colegas da Universidade de Michigan nas décadas de 60 e 70. Em contraste com as estratégias evolutivas (Back et al., 1991) e a programação evolutiva (Fogel et al., 1966), o objetivo original de Holland não era projetar algoritmos para

solucionar problemas específicos, mas estudar o fenômeno da adaptação na natureza e desenvolver formas onde os mecanismos da adaptação natural pudessem ser simulados em computadores. Em seu livro, Holland (1975) apresentou os algoritmos genéticos como uma abstração da evolução biológica. O algoritmo genético introduzido por Holland (1975) é um método que retira cromossomos (cadeias de símbolos) de uma população de cromossomos e insere-os em uma nova população, usando um tipo de seleção natural juntamente com os operadores de cruzamento e mutação inspirados na genética. Cada cromossomo consiste de genes, e cada gene representa uma instância de um alelo particular (valores que o gene pode assumir). O operador de seleção escolhe, dentre os cromossomos da população, aqueles que irão se reproduzir, usando um algoritmo de seleção. Os cromossomos selecionados podem sofrer modificações em suas características através dos operadores de cruzamento e mutação, originando descendentes para a próxima geração.

Em se tratando de propósitos computacionais, um algoritmo genético simples possui uma estrutura conforme o pseudocódigo a seguir (Michalewicz, 1996):

```

procedimento algoritmo genético
início
     $t \leftarrow 0$ 
    inicializar  $P(t)$ 
    avaliar  $P(t)$ 
    enquanto não (condição_ término) faça
        início
             $t \leftarrow t + 1$ 
            selecionar  $P(t)$  de  $P(t - 1)$ 
            alterar  $P(t)$ 
            avaliar  $P(t)$ 
        fim
    fim

```

Algoritmo 3.1: Estrutura de um algoritmo genético.

Durante a iteração t , o algoritmo genético mantém uma população $P(t)$ de soluções candidatas (cromossomos). Cada cromossomo é avaliada para medir seu *fitness* (ou aptidão), ou seja, a qualidade da solução do problema representada por este cromossomo. Então, uma nova população (iteração $t + 1$) é formada pela seleção dos indivíduos mais aptos da população $P(t)$. Alguns membros desta nova população sofrerão alterações devido à ação dos operadores genéticos de cruzamento e mutação, enquanto outros permanecerão intactos. O cruzamento combina as características de dois cromossomos pais para formar dois cromossomos filhos. O objetivo da aplicação do cruzamento é trocar informações entre soluções em potencial. A mutação altera aleatoriamente um ou mais genes de um

cromossomo selecionado, com o intuito de introduzir informação extra para a população. O algoritmo genético termina quando um critério de parada é satisfeito, como por exemplo, um número máximo de gerações predefinido que o algoritmo genético deve executar.

3.2.2 Parâmetros de um Algoritmo Genético

Segundo Michalewicz (1996), para a implementação de um algoritmo genético, deve-se primeiramente definir alguns aspectos importantes, tais como, uma representação genética para as soluções do problema, uma forma de criar uma população inicial das soluções, definição do tamanho desta população, uma função de avaliação que desempenha o papel do ambiente, métodos de seleção de indivíduos, operadores genéticos que alteram a composição dos filhos e um critério de parada. A influência de cada um destes aspectos no desempenho do algoritmo depende da classe de problemas que está sendo tratada.

3.2.2.1 Representação Genética

Existem alguns tipos de representação genética para as soluções de um problema, dentre as quais se destacam as codificações binária e real. A representação binária foi amplamente usada em algoritmos genéticos, uma vez que é de fácil utilização e manipulação, além de simples de analisar teoricamente. Contudo, por apresentar desvantagens, como por exemplo, gerar um complexo espaço de busca resultante da codificação de cromossomos de grande tamanho, quando aplicada a problemas multidimensionais e a problemas numéricos de alta precisão (Michalewicz, 1996), a representação binária deixou de ser utilizada, sendo que as codificações real ou inteira são mais aptas nestes casos por possuir um maior grau de representatividade.

3.2.2.2 População Inicial

Uma busca rápida e uma boa cobertura do espaço de busca são essenciais para a resolução de problemas utilizando-se algoritmos genéticos. Sendo assim, a geração da população inicial torna-se um aspecto importante no processo genético.

Com o objetivo de acelerar o processo de busca, pode-se inserir na população inicial algum conhecimento prévio que indique regiões promissoras do espaço de busca (Lacerda, 2003). Em relação à cobertura do espaço, a escolha de uma população inicial maior que a

população a ser utilizada nas gerações subseqüentes pode melhorar a representação do espaço de busca (Lacerda e Carvalho, 1999). Se uma população inicial pequena for gerada aleatoriamente, provavelmente algumas regiões do espaço de busca não serão representadas. Geralmente, utiliza-se um tamanho de população proporcional ao tamanho do cromossomo, isto é, quanto maior for o cromossomo maior deverá ser o tamanho da população a fim de manter uma diversidade razoável.

3.2.2.3 *Função de Avaliação*

A qualidade dos indivíduos para a solução de um determinado problema é dada por meio dos seus valores de *fitness* (ou aptidão). Este valor é calculado por uma função de avaliação, a qual assume de forma análoga, o papel do ambiente, ou seja, os indivíduos mais bem adaptados a um determinado meio terão valores de aptidão melhores do que os indivíduos menos adaptados.

Segundo Lacerda e Carvalho (1999), alguns cuidados podem ser tomados para que indivíduos idênticos não sejam avaliados mais de uma vez, tais como: evitar gerar cromossomos idênticos na população inicial, verificar se o filho é igual a um dos pais e, antes de avaliar um filho, verificar se um cromossomo idêntico já existe na população.

3.2.2.4 *Método de Seleção*

Dada uma população em que a cada indivíduo foi atribuído um valor de aptidão, existem vários métodos para selecionar os indivíduos sobre os quais serão aplicados os operadores genéticos de cruzamento e mutação. Estes indivíduos selecionados formarão uma população, que é conhecida como população intermediária. A maioria dos métodos de seleção é projetada para escolher preferencialmente indivíduos com maiores ou menores valores de aptidão, embora não exclusivamente, a fim de manter a diversidade da população. Com base em Michalewicz (1996), alguns destes métodos são apresentados a seguir.

Seleção pela Roleta

O método de seleção da Roleta é um método de seleção muito utilizado, onde os indivíduos de uma geração são escolhidos para fazer parte da população intermediária através de um sorteio de roleta. Neste método, cada indivíduo da população é representado na roleta

proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa é dada uma porção relativamente menor. Finalmente, a roleta é girada um determinado número de vezes dependendo do tamanho da população e são escolhidos como indivíduos que participarão da população intermediária aqueles sorteados na roleta.

procedimento roleta

início

$T \leftarrow$ soma dos valores de aptidão de todos os indivíduos da população

repita

$r \leftarrow$ valor aleatório entre 0 e T

Percorra sequencialmente os indivíduos da população, acumulando em S o valor de aptidão dos indivíduos já percorridos

se $S \geq r$ **então**

selecione o indivíduo corrente

até (n indivíduos terem sido selecionados)

fim

Algoritmo 3.2: Seleção pela Roleta (Mitchell, 1996).

Seleção por Amostragem Universal Estocástica

Para visualizar este método, considere um círculo dividido em n regiões (tamanho da população), onde a área de cada região é proporcional à aptidão do indivíduo. Colocam-se sobre este círculo uma "roleta" com n cursores, igualmente espaçados. Após um giro da roleta, a posição dos cursores indica os indivíduos selecionados. Evidentemente, os indivíduos cujas regiões possuem maior área, terão maior probabilidade de serem selecionados, mais de uma vez inclusive. Como consequência, a seleção de indivíduos pode conter várias cópias de um mesmo indivíduo enquanto outros podem desaparecer. Este tipo de seleção é mais rápido que a Roleta, pois em uma só rodada já são selecionados vários indivíduos.

Seleção por Torneio

Este método escolhe aleatoriamente dois indivíduos da população. Um número aleatório r é escolhido entre zero e um. Se $r < k$ (onde k é um valor predefinido entre zero e um, por exemplo, 0,75), o melhor indivíduo é selecionado; caso contrário, o pior será selecionado. Os dois cromossomos retornam para a população original e podem ser selecionados novamente. Este processo se repete até selecionar N indivíduos, onde N é igual ao tamanho da população. O pseudocódigo deste método é descrito pelo Algoritmo 3.3.

```

procedimento torneio ( $k$ )
início
     $N \leftarrow 1$ 
    repita
        selecione 2 indivíduos da população
         $r \leftarrow [0,1]$ 
        se ( $r < k$ ) então
            o melhor indivíduo é selecionado
        else
            o pior indivíduo é selecionado
         $N \leftarrow N + 1$ 
    até ( $N = \text{tamanho da população}$ )
fim

```

Algoritmo 3.3: Seleção por Torneio (Mitchell, 1996).

Seleção Elitista ou Elitismo

A seleção elitista ou elitismo, introduzido por De Jong (1975), é um método de seleção que força o algoritmo genético a reter um determinado número de melhores indivíduos em cada geração. A aplicação do elitismo previne que tais indivíduos sejam destruídos pela aplicação dos operadores de cruzamento ou mutação (Mitchell, 1996). Devido ao seu comportamento, a seleção elitista é sempre utilizada em conjunto com algum outro método de seleção.

3.2.2.5 Operadores Genéticos

Os operadores genéticos de cruzamento e mutação provocam alterações em uma população. O operador de cruzamento é aplicado aos cromossomos de acordo com uma probabilidade conhecida como taxa de cruzamento. Quanto maior for esta taxa, mais rapidamente novos indivíduos serão introduzidos na população. Mas se esta for muito alta, a maior parte da população será substituída e poderá ocorrer a perda de indivíduos com alta aptidão. Com um valor baixo, o algoritmo pode se tornar muito lento. A mutação é necessária para a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais indivíduos, fornecendo assim, meios para a introdução de novos indivíduos na população. O operador de mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação. Uma baixa taxa de mutação pode assegurar a diversidade na população, mas o contrário pode destruir toda a informação contida no indivíduo que foi adquirida durante as gerações passadas.

Assim, as taxas de cruzamento e mutação devem ser definidas empiricamente para cada domínio de problema, levando em conta as considerações mencionadas. A maior parte da literatura recomenda uma taxa de cruzamento entre 75% e 95% (Mitchell, 1996). A literatura ainda recomenda uma taxa de mutação entre 0,5% e 1% (Mitchell, 1996).

A escolha dos operadores genéticos está intimamente ligada à codificação adotada para a representação genética (Delgado, 2002), ou seja, existe uma variação no comportamento dos operadores, o qual está relacionado à codificação empregada nos cromossomos. Em outras palavras, há operadores genéticos que foram concebidos para o uso com codificação binária e operadores genéticos para o uso com codificação real ou inteira.

A seguir, serão apresentados apenas os operadores genéticos para codificação real ou inteira, uma vez que a codificação real foi utilizada neste trabalho.

Operadores Genéticos para Codificação Real ou Inteira

A seguir serão descritos os operadores genéticos de cruzamento e mutação para codificação real ou inteira, de acordo com Michalewicz (1996) e Lacerda (2003), mas antes é necessário definir o conceito de domínio. Domínio define o intervalo de possíveis valores que os genes dos cromossomos podem assumir. Desta forma, o domínio para os genes dos cromossomos com codificação binária abrange os valores zero ou um, contudo, para os cromossomos com codificação real ou inteira, isto dependerá do problema que será tratado.

Cruzamento Simples

O operador genético cruzamento simples é definido da seguinte forma: se os cromossomos $C_1=(x_1,...,x_q)$ e $C_2=(y_1,...,y_q)$ são cruzados a partir da k -ésima posição, os descendentes serão: $C'_1=(x_1,...,x_k,y_{k+1},...,y_q)$ e $C'_2=(y_1,...,y_k,x_{k+1},...,x_q)$. Tal operador pode produzir descendentes fora do domínio em questão. Para evitar isto, usa-se a propriedade do espaço convexo, onde $a \in [0,1]$ tal que:

$$C'_1 = (x_1,..., x_k, y_{k+1} \cdot a + x_{k+1} \cdot (1 - a),..., y_q \cdot a + x_q \cdot (1 - a)), \quad (3.5)$$

$$C'_2 = (y_1,..., y_k, x_{k+1} \cdot a + y_{k+1} \cdot (1 - a),..., x_q \cdot a + y_q \cdot (1 - a)). \quad (3.6)$$

Na Figura 3.3 é ilustrado um exemplo do funcionamento de um cruzamento simples, sendo $a = 0,7$.

Figura 3.3: Cruzamento simples.

Cruzamento Aritmético

O cruzamento aritmético é definido como uma combinação linear de dois vetores. Se x_1 e x_2 são cruzados, os descendentes serão $x'_1 = a.x_1 + (1-a).x_2$ e $x'_2 = a.x_2 + (1-a).x_1$. Este operador usa um valor aleatório $a \in [0,1]$, para garantir que x'_1 e x'_2 estejam dentro do domínio de x_1 e x_2 . Na Figura 3.4 é ilustrado o funcionamento deste operador, onde $a = 0,7$.

Figura 3.4: Cruzamento aritmético.

Cruzamento Heurístico

O cruzamento heurístico utiliza valores da função de *fitness* na determinação da direção da busca, gerando apenas um ou nenhum descendente. O cromossomo filho x_3 é gerado a partir de dois cromossomos pais x_1 e x_2 , de acordo com a seguinte regra: $x_3 = r(x_2 - x_1) + x_2$, onde r é um número aleatório entre zero e um. Na Figura 3.5 é ilustrado o comportamento deste operador, onde $r = 0,8$.

Figura 3.5: Cruzamento heurístico.

Cruzamento BLX- α

O cruzamento BLX- α consiste em gerar um novo cromossomo a partir da seguinte expressão:

$$c = p_1 + r(p_2 - p_1) \quad (3.7)$$

sendo que c é o cromossomo filho gerado, p_1 e p_2 são os cromossomos pais e r é um vetor aleatório entre $[-\alpha, 1+\alpha]$. A aplicação do cruzamento BLX- α a um espaço unidimensional é ilustrada na Figura 3.6 e a Figura 3.7 mostra para o caso bidimensional. O termo α é um pequeno valor que estende os limites para a definição de c . Para o espaço unidimensional, se $\alpha = 0$, os cromossomos filhos estarão dentro do segmento de linha L que une p_1 e p_2 . Se $\alpha > 0$, o segmento de linha L é estendido. Para o espaço bidimensional, o cruzamento BLX- α gera filhos aleatoriamente dentro de um hiper-retângulo definido pelos valores dos pais. Usualmente, define-se $\alpha = 0.5$ (estende o segmento L em $0.5L$ para cada extremo), possibilitando que os filhos estejam dentro ou fora do segmento de linha L com a mesma probabilidade.

Figura 3.6: BLX- α aplicado a um espaço unidimensional.

Figura 3.7: BLX- α aplicado a um espaço bidimensional.

Mutação Uniforme

Baseado em Michalewicz (1996), este operador seleciona um gene aleatório $k \in \{1, 2, \dots, q\}$ do cromossomo $C = (x_1, \dots, x_k, \dots, x_q)$ e produz $C' = (x_1, \dots, x'_k, \dots, x_q)$, onde x'_k é um valor aleatório (distribuição probabilística uniforme) do intervalo $[\text{esquerda}(k), \text{direita}(k)]$, onde $\text{esquerda}(k)$ é o limite inferior e $\text{direita}(k)$ é o limite superior do intervalo que representa o domínio do gene x_k . Um exemplo é ilustrado na Figura 3.8, onde $q=5$ e $k=4$.

Figura 3.8: Mutação uniforme.

Mutação Não-Uniforme

Seja um cromossomo pai $C = (x_1, \dots, x_k, \dots, x_q)$. Se o elemento x_k for selecionado para mutação o resultado será $C' = (x_1, \dots, x'_k, \dots, x_q)$, onde:

$$x'_k = \begin{cases} x_k + \Delta(t, \text{direita}(k) - x_k) & \text{se o dígito binário é 0} \\ x_k + \Delta(t, x_k - \text{esquerda}(k)) & \text{se o dígito binário é 1} \end{cases} \quad (3.8)$$

A função $\Delta(t, y)$ retorna um valor no intervalo $[0, y]$ tal que, quando a probabilidade de $\Delta(t, y)$ se aproxima de zero, ela é incrementada de acordo com o aumento de t (número de gerações). Esta propriedade faz o operador buscar uniformemente pelo espaço (quando t é pequeno), e localmente em estágios posteriores. A função $\Delta(t, y)$ é definida pela expressão $\Delta(t, y) = y \cdot r \cdot (1 - t/T)^b$, onde r é um número aleatório de $[0, 1]$, T é o número máximo de gerações e b é um parâmetro do sistema que determina o grau de não-uniformidade.

3.2.2.6 Critério de Parada

Os critérios de parada de um algoritmo genético variam de acordo com a aplicação em questão. A finalização da execução pode ocorrer após um número de gerações pré-fixado, pode ser baseada na repetição sucessiva do melhor indivíduo durante as gerações, pode ser efetuada através da observância de que a média ou o desvio padrão do valor de aptidão não

sofre alteração no decorrer das gerações, ou simplesmente parar ao encontrar a solução ótima, caso ela seja conhecida.

3.3 Considerações Finais

Este capítulo apresentou os fundamentos básicos das redes neurais artificiais, mais especificamente sobre as redes recorrentes de Hopfield, e dos algoritmos genéticos, com o propósito de fornecer os fundamentos teóricos necessários para o entendimento da abordagem híbrida proposta nesta tese, a qual é composta por uma rede neural artificial e por um algoritmo genético, e que será detalhada no próximo capítulo.

Capítulo 4

Abordagem Neuro-Genética

Este capítulo descreve a abordagem Neuro-Genética desenvolvida. A Seção 4.1 apresenta algumas dificuldades das redes neurais em resolver problemas de otimização combinatória quando aplicadas isoladamente, ou seja, quando não associadas a outro método. Na Seção 4.2 é apresentada a técnica do subespaço-válido de soluções, utilizada para projetar as soluções obtidas para uma região factível do espaço de busca do problema. A Seção 4.3 descreve a Rede de Hopfield Genética, abordagem esta que associa uma rede de Hopfield a um algoritmo genético, e que será utilizada para a resolução de problemas de conexão em otimização combinatória. Por fim, na Seção 4.4, a dinâmica da Rede de Hopfield Genética é detalhada.

4.1 Abordagem Neural e Suas Limitações

Grande parte das abordagens neurais aplicadas em otimização de sistemas utilizam redes recorrentes (Wen et al., 2008), sendo que a rede neural de Hopfield é provavelmente o melhor exemplo conhecido deste tipo de rede (Hush e Horne, 1993), devido à sua facilidade de implementação em *hardware*, assim como a sua utilização em outros tipos de aplicação, como por exemplo, memórias associativas, comentadas na Seção 3.1.

O modelamento de problemas de otimização combinatória utilizando uma rede neural de Hopfield consiste em determinar, independente da natureza diferente de cada problema, a matriz de pesos T (simétrica) e o vetor de entradas i^b associados à função de energia da rede,

descrita na equação (3.3). Esta função, por sua vez, deve ser associada à função objetivo e às restrições descrevendo o problema a ser resolvido.

A dificuldade em mapear problemas de otimização variados por meio de uma rede neural de Hopfield se deve à necessidade de satisfazer as várias restrições que são impostas por cada tipo de problema (Wen et al., 2008). Neste caso, a função de energia associada à rede deve levar em conta não só a função objetivo do problema, mas também todas as restrições existentes. A rede atua com o propósito de minimizar simultaneamente uma função de energia $E^{ot}(t)$ correspondente à função objetivo e um conjunto de funções (E_k^{rest}) descrevendo as k restrições envolvidas no problema. Se qualquer uma destas restrições for violada, a solução é considerada inválida (Gee e Prager, 1991). Uma técnica simples de modelamento é a que utiliza Multiplicadores de Lagrange (Bazaraa et al., 2006), a qual consiste em incluir as restrições como termos na função de energia, que são minimizados quando as restrições são satisfeitas, ou seja:

$$E(t) = E^{ot}(t) + c_1 \cdot E_1^{rest}(t) + c_2 \cdot E_2^{rest} + \dots + c_k \cdot E_k^{rest} \quad (4.1)$$

sendo que c_i são constantes positivas que ponderam cada uma das restrições.

Os primeiros resultados publicados relativos à utilização de redes neurais artificiais em problemas de otimização foram fornecidos por Hopfield e Tank (1985). Em tal trabalho são descritos os resultados obtidos, em simulações por computador, da rede neural de Hopfield para resolver o clássico problema do caixeiro viajante. Entretanto, a maioria das soluções obtidas pela rede eram consideradas inválidas, pois os caminhos encontrados pela rede eram ineficazes. As principais razões para este problema de convergência, como discutido posteriormente por alguns autores, são as seguintes:

- i) Dificuldade de se obter os valores corretos para as constantes arbitrárias ($c_1, c_2, \dots, c_k$) que ponderam os termos de energia relativos às restrições do problema a ser resolvido (Kamgar-Parsi e Kamgar-Parsi, 1990; Hegde et al., 1998).
- ii) Influência dos termos de restrições no termo de otimização, dificultando a convergência da rede (Wilson e Pawley, 1988; Peterson e Soderberg, 1989).
- iii) A qualidade da solução final é afetada pelos valores das constantes de ponderação.

Em resumo, no processo de otimização com uma rede neural de Hopfield, cuja função objetivo é dada por (4.1), a multiplicidade de termos de restrições nesta equação afeta

consideravelmente o valor da solução final. Como resultado, as soluções obtidas no final do processo de otimização podem ser infactíveis (caso alguma restrição não seja satisfeita); além disso, o desempenho da rede é sensível aos valores dos parâmetros c_i .

Para contornar estes problemas, realiza-se em Aiyyer et al. (1990) uma análise dos autovetores da rede durante a sua convergência, o que permite verificar como $E^{ot}(t)$ e os termos de restrições em (4.1) podem ser efetivamente separados dentro de subespaços diferentes, de modo a tornar o problema factível. O subespaço que agrupa todas as restrições impostas pelo problema é denominado subespaço-válido de soluções (Seção 4.2). Neste caso, a equação definida por (4.1) é simplificada para:

$$E(t) = E^{ot}(t) + c_0 \cdot E^{conf}(t) \quad (4.2)$$

sendo que o termo $E^{conf}(t)$ satisfaz todas as restrições dadas na equação (4.1).

Dado que $E(t)$ foi dividido em um termo de confinamento ($E^{conf}(t)$) e em um termo de otimização ($E^{ot}(t)$) que pode ser representativo da função custo do problema, tal que a minimização de $E^{conf}(t)$ da rede confina v dentro de um subespaço-válido e a minimização de $E^{ot}(t)$ da rede move v em direção a uma solução ótima, então, similarmente à equação (3.3), os termos da função de energia em (4.2) podem ser escritos como:

$$E^{ot}(t) = -\frac{1}{2} v(t)^T \cdot T^{ot} \cdot v(t) - v(t)^T \cdot i^{ot} \quad (4.3)$$

$$E^{conf}(t) = -\frac{1}{2} v(t)^T \cdot T^{conf} \cdot v(t) - v(t)^T \cdot i^{conf} \quad (4.4)$$

Devido à necessidade de assegurar que $E^{conf}(t)$ seja dominante (satisfação das restrições) sobre $E^{ot}(t)$, deve-se atribuir um valor elevado para a constante c_0 em (4.2). Esta condição torna a simulação da rede ineficiente, visto que a maior parte do tempo será despendida em garantir o confinamento de v no subespaço-válido. Para evitar este problema, propõe-se na Seção 4.2 uma estratégia para obtenção dos parâmetros T^{conf} e i^{conf} do termo $E^{conf}(t)$ definido na Equação (4.4), que diretamente garante a validade das restrições agrupadas pelo mesmo, dispensando a utilização da constante de ponderação c_0 .

4.2 Abordagem do Subespaço-Válido de Soluções

Para uma grande variedade de problemas com restrições que podem ser resolvidos pela rede neural de Hopfield, através da decomposição da função de energia da rede (3.3) nas duas parcelas dadas em (4.3) e (4.4), verifica-se que os pontos de equilíbrio da rede, correspondentes aos valores de v que minimizam a função de energia E^{conf} dada pela equação (4.4), pertencem todos a um mesmo subespaço comum (Aiyer et al., 1990). Este subespaço comum, denominado subespaço-válido de soluções, possui equação definida por:

$$v(t+1) = T^{val}.v(t) + s \quad (4.5)$$

sendo que:

- a) T^{val} é uma matriz projeção (isto é: $T^{val}.T^{val} = T^{val}$) que projeta o vetor v no subespaço-válido ($v^{val} = T^{val}.v$), onde v^{val} é a componente de v projetada sobre o subespaço-válido);
- b) s é um vetor que está relacionado com as restrições do problema a ser resolvido, sendo o mesmo ortogonal ao subespaço-válido ($T^{val}.s = 0$).

A partir da abordagem do subespaço-válido, o funcionamento da rede de Hopfield é realizado em dois passos principais:

- 1) A rede é inicializada em um estado aleatório, tal que v seja limitado pela função de ativação dos neurônios;
- 2) A minimização de $E^{conf}(t)$, por intermédio da projeção de v no subespaço-válido, move v em direção a um dos pontos de equilíbrio da rede que satisfaz todas as restrições impostas pelo problema.

Portanto, a função principal da projeção no subespaço-válido é confinar todas as restrições impostas pelo problema dentro de um único subespaço. As soluções que são factíveis a todas as restrições estão contidas neste subespaço-válido. Um estudo mais abrangente, tratando da abordagem de subespaço-válido, é apresentado em Silva (1995).

4.3 A Rede de Hopfield Genética

Analogamente à Equação (4.2), a função de energia $E(t)$ adotada neste trabalho será composta de dois termos, sem se ter a necessidade de presença da constante de ponderação c_0 , ou seja:

$$E(t) = E^{ot}(t) + E^{conf}(t) \quad (4.6)$$

sendo que $E^{conf}(t)$ é um termo de confinamento que agrupa as restrições estruturais impostas pelo problema, e $E^{ot}(t)$ é a minimização da função objetivo do problema em questão, a qual é realizada por meio de um algoritmo genético, com o objetivo de conduzir a saída da rede para os pontos de equilíbrio que representam as soluções do problema de otimização considerado.

Como descrito anteriormente, o modelo convencional da rede de Hopfield necessita de um esforço computacional adicional para forçar o confinamento de v no subespaço-válido, gastando tempo para isso. Com o propósito de contornar tais problemas, propõe-se uma Rede de Hopfield Genética (RHG), a qual possui a dinâmica explicitada por meio dos três passos que compõem o esquema da Figura 4.1.

Figura 4.1: A rede de Hopfield Genética.

(I) Minimização de E^{conf} : corresponde à projeção de v no subespaço-válido gerado por todas as restrições impostas pelo problema:

$$v(t+1) = T^{conf} \cdot v(t) + i^{conf} \quad (4.7)$$

Dado que T^{conf} é uma matriz projeção, qualquer componente que seja ortogonal a $T^{conf} \cdot v$ é desprezada (Seção 4.4). Esta operação realiza uma minimização indireta de E^{conf} , sendo que $T^{conf} = T^{val}$ e $i^{conf} = s$.

(II) Aplicação de uma função de ativação do tipo ‘rampa-simétrica’, restringindo v dentro de um hipercubo inicial:

$$g_i(v_i) = \begin{cases} 0 & \text{se } 0 > v_i \\ v_i & \text{se } 0 < v_i < 1 \\ 1 & \text{se } v_i > 1 \end{cases} \quad (4.8)$$

onde $v_i \in [0,1]$. Este passo assegura que os elementos do vetor v de saída da rede estejam sempre entre $[0,1]$.

Após a convergência do loop interno, o qual é realizado por meio das aplicações sucessivas dos passos (I) e (II), a RHG está pronta para executar o terceiro passo de otimização, dado a seguir.

(III) Minimização de $E^{ot}(t)$: Com base nos valores obtidos para v , após seu confinamento para um subespaço-válido através da aplicação dos passos (I) e (II) ilustrados na Figura 4.1, aplica-se um algoritmo genético para a alteração de v em direção a uma solução ótima frente à função objetivo do problema.

Como observado na Figura 4.1, as aplicações sucessivas dos passos (I) e (II), seguidas pela execução do passo (III), levam a saída da RHG para um ponto de equilíbrio que corresponde à solução ótima do problema de otimização. Neste caso, a iteração representada pelos passos (I), (II) e (III) mencionados tem dois estágios distintos. No primeiro estágio, v é projetado diretamente no subespaço-válido a fim de satisfazer as restrições impostas para o problema. Este primeiro estágio é um processo iterativo, em que v é ortogonalmente projetado no subespaço-válido por meio da aplicação do Passo (I), seguido pela aplicação do Passo (II), a fim de limitar seus elementos no domínio definido por $[0,1]$. No segundo estágio, aplicado após a execução dos Passos (I) e (II), v é alterado em direção a uma solução ótima para o problema utilizando um algoritmo genético (Passo (III)).

A aplicação dos passos (I), (II) e (III) é repetida sucessivamente, enquanto os valores de v devolvidos pela rede de Hopfield (v^{rede}) e pelo algoritmo genético (v^{ag}) não forem bem similares. O processo de convergência é concluído quando os valores de v^{rede} e v^{ag} estão bem próximo um do outro entre duas iterações sucessivas, considerando para tanto a tolerância (precisão ε) requerida para a comparação dos mesmos.

A Rede de Hopfield Genética pode ser implementada por meio dos passos do fluxograma descrito na Figura 4.2.

Figura 4.2: Fluxograma da rede de Hopfield Genética.

A próxima seção discute o processo de convergência da Rede de Hopfield Genética.

4.4 Dinâmica da Rede de Hopfield Genética

Nesta seção, analisa-se a convergência da Rede de Hopfield Genética cuja dinâmica de operação é implementada através dos passos descritos na Figura 4.2. Em particular, considera-se que a região de operação na qual o vetor v está contido é limitada pelo hipercubo definido

pela função de ativação ‘rampa-simétrica’ (4.8). A equação nodal descrevendo o comportamento dinâmico desta rede é obtida a partir de (3.1) para $\eta = 0$ e $v(t) = u(t)$, ou seja:

$$\dot{v}_i(t) = \frac{\partial v_i}{\partial t} = \sum_{j=1}^N T_{ij} \cdot v_j(t) + i_i^b \quad (4.9)$$

A análise dinâmica da rede é, portanto, inicializada com a obtenção de uma equação para $\dot{v}^{val}$, componente de $\dot{v}$ que pertence ao subespaço-válido.

Como na Rede de Hopfield Genética, v é constantemente confinado ao subespaço-válido pela aplicação da operação (I), isto é, $v = T^{val} \cdot v + s$, qualquer componente de $\dot{v}$ ortogonal a $\dot{v}^{val}$ é continuamente suprimida. Logo, a componente $\dot{v}^{val}$ (não $\dot{v}$) caracteriza melhor a dinâmica global da rede, ou seja:

$$\begin{aligned} \dot{v}^{val} &= T^{val} \cdot \dot{v} = T^{val}(T^{ot} \cdot v + i^{ot}) \\ &= T^{val}(T^{ot}(T^{val} \cdot v + s) + i^{ot}) \\ &= T^{val}(T^{ot}T^{val} \cdot v + T^{ot} \cdot s + i^{ot}) \\ &= T^{val}T^{ot}T^{val} \cdot v + T^{val}(T^{ot} \cdot s + i^{ot}) \end{aligned} \quad (4.10)$$

Da equação (4.10), verifica-se que $\dot{v}^{val}$ é composta de duas partes: um termo independente de v , $T^{val}(T^{ot} \cdot s + i^{ot})$, e um termo que depende de v , $T^{val}T^{ot}T^{val} \cdot v$. Para simplificar, estes termos passam a ser definidos por:

$$T^{val}T^{ot}T^{val} = A \quad (4.11)$$

$$T^{val}(T^{ot} \cdot s + i^{ot}) = b \quad (4.12)$$

Assim, a equação (4.10) torna-se:

$$\dot{v}^{val} = A \cdot v + b = A \cdot v^{val} + b \quad (4.13)$$

sendo que $v^{val} = T^{val} \cdot v$ e $T^{val} \cdot T^{val} = T^{val}$.

Para sistemas invariantes no tempo (autônomos), a solução geral de (4.13) pode ser descrita por meio de uma equação exponencial matricial (D' azzo e Houppis, 1975); (Rosenbrook e Storey, 1970; Vidyasagar, 1993):

$$v^{val}(t) = e^{At} v_0^{val} + \int_0^t e^{A(t-\tau)} b \cdot d\tau \quad (4.14)$$

sendo que v_0^{val} é o valor de v^{val} inicializado no tempo $t = 0$. Este valor é geralmente escolhido como um vetor com elementos aleatórios pequenos.

Dado que a expansão em série da exponencial $e^{A\zeta}$ é dada por:

$$e^{A\zeta} = \sum_{k=0}^{\infty} \frac{(A\zeta)^k}{k!} \quad (4.15)$$

a equação (4.14) torna-se:

$$\begin{aligned} v^{val}(t) &= \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k v_0^{val} + \int_0^t \sum_{k=0}^{\infty} \frac{(t-\tau)^k}{k!} A^k b \cdot d\tau \\ &= \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k v_0^{val} + \sum_{k=0}^{\infty} \frac{A^k b}{k!} \int_0^t (t-\tau)^k \cdot d\tau \\ &= \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k v_0^{val} + \sum_{k=0}^{\infty} \frac{A^k b}{k!} \left(\frac{t^{k+1}}{k+1} \right) \\ &= \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k v_0^{val} + \sum_{k=0}^{\infty} \frac{t^{k+1}}{(k+1)!} A^k b \end{aligned} \quad (4.16)$$

Para se analisar o comportamento de v^{val} durante o processo de convergência da rede, deve-se escrever os vetores v^{val} , v_0^{val} e b em termos de suas componentes expressas no espaço coordenado gerado pelos autovetores normalizados da matriz A . Para isto, considere $\lambda_1, \lambda_2, \dots, \lambda_N$ os autovalores de A , com autovetores normalizados $u_1, u_2, \dots, u_N$. Para se distinguir entre os autovalores nulos e não-nulos de A , define-se o conjunto Z , tal que $\lambda_i = 0$ para $i \in Z$, e $\lambda_i \neq 0$ para $i \notin Z$. A decomposição de v^{val} , v_0^{val} e b na direção dos autovetores de A gera:

$$v^{val} = \sum_{i=1}^N v_i u_i, \quad v_0^{val} = \sum_{i=1}^N o_i u_i, \quad b = \sum_{i=1}^N b_i u_i \quad (4.17)$$

sendo que v_i , o_i e b_i são, respectivamente, os valores da i -ésima componente dos vetores v^{val} , v_0^{val} e b , representadas no espaço coordenado gerado pelos autovetores de A .

A partir de (4.17), obtém-se:

$$A^k v_0^{val} = \sum_{i=1}^N o_i \lambda_i^k u_i, \quad A^k b = \sum_{i=1}^N b_i \lambda_i^k u_i \quad (4.18)$$

Com a utilização de (4.18), a equação (4.16) torna-se:

$$\begin{aligned}
v^{val}(t) &= \sum_{k=0}^{\infty} \frac{t^k}{k!} \sum_{i=1}^N o_i \lambda_i^k u_i + \sum_{k=0}^{\infty} \frac{t^{k+1}}{(k+1)!} \sum_{i=1}^N b_i \lambda_i^k u_i \\
&= \sum_{i=1}^N o_i u_i \underbrace{\sum_{k=0}^{\infty} \frac{t^k \lambda_i^k}{k!}}_{e^{\lambda_i t}} + \sum_{i \notin Z} \frac{b_i u_i}{\lambda_i} \sum_{k=0}^{\infty} \frac{t^{k+1} \lambda_i^{k+1}}{(k+1)!} + \sum_{i \in Z} b_i u_i \sum_{k=0}^{\infty} \frac{t^{k+1} 0^k}{(k+1)!} \\
&= \sum_{i=1}^N e^{\lambda_i t} o_i u_i + \sum_{i \notin Z} \frac{b_i u_i}{\lambda_i} \left(\underbrace{\sum_{k=0}^{\infty} \frac{t^k \lambda_i^k}{k!}}_{e^{\lambda_i t}} - 1 \right) + \sum_{i \in Z} b_i u_i t \\
&= \sum_{i=1}^N e^{\lambda_i t} o_i u_i + \sum_{i \notin Z} \frac{b_i u_i}{\lambda_i} (e^{\lambda_i t} - 1) + \sum_{i \in Z} b_i u_i t
\end{aligned} \tag{4.19}$$

A equação (4.19) é válida para valores arbitrários de A , b e v_0^{val} . Entretanto, pode-se simplificar esta equação a partir das definições de A e b dadas nas equações (4.11) e (4.12). Desta análise, verifica-se que b permanece integralmente no subespaço-válido durante toda convergência, de modo que $b_i = 0$ para $i \in Z$. Assim, a equação (4.19) torna-se:

$$v^{val}(t) = \sum_{i=1}^N e^{\lambda_i t} o_i u_i + \sum_{i \notin Z} \frac{b_i u_i}{\lambda_i} (e^{\lambda_i t} - 1) \tag{4.20}$$

Para t pequeno, tem-se que $e^{\lambda_i t} \approx 1 + \lambda_i t$. Logo, a equação (4.20) torna-se:

$$v^{val}(t) \approx \sum_{i=1}^N [o_i(1 + \lambda_i t) + b_i t] u_i \tag{4.21}$$

Como v_0^{val} é escolhido aleatoriamente pequeno, os termos o_i são também pequenos em comparação com os b_i . Portanto, a equação (4.21) transforma-se em:

$$v^{val}(t) \approx t \cdot \sum_{i=1}^N b_i u_i = t \cdot b \tag{4.22}$$

Verifica-se então que os valores de v^{val} inicialmente partem na direção do vetor b . No limite, para t grande, e visto que v é limitado pelo hipercubo definido pela função de ativação rampa-simétrica dada em (4.8), a equação (4.20) indica que v^{val} tenderá à direção dos autovetores de A correspondente ao maior autovalor positivo, sendo que u^{max} denota estes autovetores (e λ^{max} o autovalor correspondente) (Vidyasagar, 1993). Assim, b e u^{max} são os vetores que mais influenciam a dinâmica de v^{val} . Quando v converge para um ponto de equilíbrio válido, o v^{val} associado contém invariavelmente uma componente significativa na direção de u^{max} .

Mostra-se, assim, que a Rede de Hopfield Genética, inicializada em um ponto aleatório, convergirá para um ponto de equilíbrio válido. Este ponto de equilíbrio está contido dentro do hipercubo definido pela função de ativação ‘rampa-simétrica’.

4.5 Considerações Finais

Com o objetivo de fornecer uma nova metodologia para modelamento de problemas de conexão em otimização combinatória, uma abordagem neuro-genética foi apresentada neste capítulo. Trata-se de uma arquitetura híbrida, constituída por uma rede neural de Hopfield e por um algoritmo genético, denominada Rede de Hopfield Genética.

A Rede de Hopfield Genética utiliza a técnica do subespaço-válido de soluções para confinar todas as restrições impostas pelo problema de otimização dentro de um único subespaço, pois todas as soluções que são factíveis a todas as restrições estão contidas neste subespaço-válido. Após o confinamento de todas as restrições, um algoritmo genético é utilizado para minimizar a função objetivo do problema. Portanto, a resolução de problemas de conexão em otimização combinatória usando a Rede de Hopfield Genética é executada em dois estágios distintos: 1) satisfação das restrições impostas pelo problema de otimização e 2) otimização da função objetivo.

As principais vantagens na utilização da abordagem neuro-genética proposta nesta tese são: i) modelamento de problemas de otimização de naturezas diferentes utilizando uma mesma metodologia; ii) viabilização do tratamento das restrições envolvidas com os problemas, sem comprometer a otimização da função objetivo; iii) não necessidade de definição de parâmetros de controle ou ponderação para sua inicialização e iv) a facilidade de implementação em *hardware*.

O próximo capítulo apresenta a metodologia utilizada no modelamento de problemas de conexão em otimização combinatória através da Rede de Hopfield Genética.

Capítulo 5

Aplicação da RHG em Problemas de Otimização Combinatória

Neste capítulo são descritas as formas de modelamento de problemas de otimização combinatória utilizando a Rede de Hopfield Genética, assim como as notações, definições e propriedades necessárias à compreensão destes modelamentos.

Como esta tese objetiva utilizar uma abordagem neuro-genética na solução de problemas de otimização combinatória, e como existem classes diferentes deste tipo de problema, conforme apresentado no Capítulo 2, a classe selecionada para a aplicação da abordagem desenvolvida foi a de problemas de conexão. Para tanto, foram escolhidos três problemas para ilustrar a aplicabilidade e eficiência da abordagem proposta: dois problemas de conexão clássicos, representados pelos problemas das N -rainhas e do emparelhamento bipartido, e outro com características que tornam sua solução possível por programação dinâmica, representado pelo problema do caminho mínimo. Por questões de simplificação, deste ponto em diante, os “problemas solucionáveis por programação dinâmica” serão chamados de “problemas de programação dinâmica”.

Em face da importância e abrangência do problema do caminho mínimo, o qual possui inúmeras aplicações práticas, tais como, roteamento de pacotes em redes de computadores (Thomopoulos et al., 1991), roteamento de veículos em sistemas de transporte (Bodin et al., 1983), roteamento de informações em redes de telecomunicações (Topkis, 1988; Ephremides

e Verdu, 1989; Antonio et al., 1992) e planejamento de rotas para robôs autônomos (Jun e Shin, 1991), este foi selecionado como o principal problema a ser abordado nesta tese. Os outros dois problemas, com menos relevância quando comparado com o caminho mínimo, foram incorporados como exemplos de que outros tipos de problemas de conexão em otimização combinatória podem ser tratados usando a mesma sistemática de modelamento empregada no problema do caminho mínimo.

5.1 Notações, Definições e Propriedades

Uma característica fundamental dos problemas que são tratados neste capítulo é a existência de um conjunto finito de n -soluções, sendo possível representar cada solução por um inteiro ou até mesmo um conjunto de inteiros. Além disso, os vetores que representam cada uma destas soluções possuem tipicamente elementos com valores zero ou um. Assim, dado um vetor $p \in \mathfrak{R}^n$, representando cada uma destas soluções, cujos elementos são inteiros no intervalo $\{1, \dots, m\}$, tem-se:

$$p_i \in \{1, \dots, m\} \text{ onde } i \in \{1, \dots, n\} \quad (5.1)$$

Pode-se representar o vetor p por um vetor v (saída da rede), com elementos definidos por zero ou um, utilizando produtos de Kronecker (produto tensor). Na notação utilizando produtos de Kronecker, têm-se as seguintes definições (Graham, 1981):

- δ é uma matriz de dimensão $n \times n$ ($\delta \in \mathfrak{R}^{n \times n}$) definida por:

$$\delta_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases} \quad (5.2)$$

e $\delta(k) \in \mathfrak{R}^n$ é o vetor coluna correspondente à k -ésima coluna de δ .

- $v(p)$ é um vetor de dimensão $n.m$ ($v(p) \in \mathfrak{R}^{n.m}$) que representa a forma final do vetor de saída da rede, v , correspondente à solução do problema representado por p . O vetor $v(p)$ é definido por:

$$v(p) = \begin{bmatrix} \delta(p_1) \\ \delta(p_2) \\ \vdots \\ \delta(p_n) \end{bmatrix} \quad (5.3)$$

- $V(p)$ é uma matriz de dimensão $n.m$ ($V(p) \in \mathbb{R}^{n.m}$) que corresponde a uma forma alternativa de se representar o vetor $v(p)$ de uma forma particionada. Esta matriz é definida por:

$$V(p) = \begin{bmatrix} \delta(p_1)^T \\ \delta(p_2)^T \\ \vdots \\ \delta(p_n)^T \end{bmatrix} \quad (5.4)$$

onde $[V(p)]_{ij} = [\delta(p_i)]_j$.

- $P \otimes Q$ denota o produto de Kronecker de duas matrizes. Se P for uma matriz de dimensão $n \times n$, e Q for uma matriz de dimensão $m \times m$, então $(P \otimes Q)$ será uma matriz de dimensão $(n.m) \times (n.m)$ dada por:

$$P \otimes Q = \begin{bmatrix} P_{11}Q & P_{12}Q & \cdots & P_{1n}Q \\ P_{21}Q & P_{22}Q & \cdots & P_{2n}Q \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1}Q & P_{n2}Q & \cdots & P_{nn}Q \end{bmatrix} \quad (5.5)$$

- $w \otimes h$ denota o produto de Kronecker de dois vetores. Se w for um vetor de dimensão n , e h for um vetor de dimensão m , então $(w \otimes h)$ será um vetor de dimensão $n.m$ dado por:

$$w \otimes h = \begin{bmatrix} w_1 \cdot h \\ w_2 \cdot h \\ \vdots \\ w_n \cdot h \end{bmatrix} \quad (5.6)$$

- $vec(U)$ é uma função que mapeia uma matriz U de dimensão $m \times n$ para um vetor v de dimensão $n.m$. Esta função é definida por:

$$v = vec(U) = [U_{11} \ U_{21} \dots U_{m1} \ U_{12} \ U_{22} \dots U_{m2} \ U_{1n} \ U_{2n} \dots U_{mn}]^T \quad (5.7)$$

As propriedades dos produtos de Kronecker que serão utilizadas nesta seção são as seguintes (Graham, 1981):

$$(\lambda w \otimes \gamma h) = \lambda \gamma (w \otimes h) \quad (5.8)$$

$$(w \otimes h)^T (x \otimes g) = (w^T x) (h^T g) \quad (5.9)$$

$$(P \otimes Q)(w \otimes h) = (Pw \otimes Qh) \quad (5.10)$$

$$(P^T \otimes Q^T) = (P \otimes Q)^T \quad (5.11)$$

$$(P \otimes Q)(E \otimes F) = (PE \otimes QF) \quad (5.12)$$

$$\text{traço}(V^T.E) = \text{vec}(V)^T.\text{vec}(E) \quad (5.13)$$

$$\text{vec}(Q.V.P^T) = (P \otimes Q).\text{vec}(V) \quad (5.14)$$

$$\text{traço}(V^T.Q.V.P^T) = \text{vec}(V)^T.(P \otimes Q).\text{vec}(V) \quad (5.15)$$

- o^n e O^n são respectivamente um vetor de dimensão n e uma matriz de dimensão $n \times n$ com todos os seus elementos possuindo valor unitário, ou seja:

$$\left. \begin{array}{l} [o]_i = 1 \\ [O]_{ij} = 1 \end{array} \right\} \text{ para } i, j \in \{1, \dots, n\} \quad (5.16)$$

- R^n é uma matriz projeção (i.e. $R^n.R^n = R^n$) de dimensão $n \times n$, tal que a pré-multiplicação de uma matriz por R^n transforma a soma dos elementos de cada coluna dessa matriz para zero, enquanto a pós-multiplicação transforma a soma dos elementos de cada linha para zero. A matriz R^n é definida por:

$$R^n = I^n - \frac{1}{n} O^n \quad (5.17)$$

As propriedades utilizadas no modelamento dos problemas de conexão em otimização combinatória são dadas por:

- Vetor soma de cada uma das colunas de uma matriz V , denotado por c , é dado por:

$$\begin{aligned} c_i &= \sum_{x=1}^n V_{xi} \\ c^T &= o^{nT}.V \end{aligned} \quad (5.18)$$

- Vetor soma de cada uma das linhas de V , denotado por l , é dado por:

$$\begin{aligned} l_i &= \sum_{x=1}^m V_{xi} \\ l^T &= V.o^m \end{aligned} \quad (5.19)$$

- Pré-multiplicação de uma dada matriz $A^{m \times n}$ por R^m transforma a soma dos elementos de cada coluna dessa matriz para zero, ou seja:

$$\begin{aligned}
\sum_{i=1}^m [R^m \cdot a]_i &= \sum_{i=1}^m \left[a - \frac{1}{m} O^m \cdot a \right]_i \\
&= o^{mT} \cdot \left[a - \frac{1}{m} O^m \cdot a \right] \\
&= o^{mT} \cdot a - \frac{1}{m} (o^{mT} \cdot o^m \cdot o^{mT}) \cdot a \\
&= o^{mT} \cdot a - o^{mT} \cdot a \\
&= o^{mT} \cdot a - o^{mT} \cdot a = 0
\end{aligned} \tag{5.20}$$

onde o vetor a corresponde a um dos vetores coluna da matriz A .

- Pós-multiplicação de uma dada matriz $A^{m \times n}$ por R^n transforma a soma dos elementos de cada linha dessa matriz para zero, ou seja:

$$\begin{aligned}
\sum_{i=1}^n [a^T \cdot R^n]_i &= \sum_{i=1}^n \left[a^T - \frac{1}{n} a^T \cdot O^n \right]_i \\
&= \left[a^T - \frac{1}{n} a^T \cdot O^n \right] \cdot o^n \\
&= a^T \cdot o^n - \frac{1}{n} a^T \cdot (o^n \cdot o^{nT} \cdot o^n) \cdot a \\
&= a^T \cdot o^n - a^T \cdot o^n \\
&= a^T \cdot o^n - a^T \cdot o^n = 0
\end{aligned} \tag{5.21}$$

onde o vetor a corresponde a um dos vetores linha da matriz A .

5.2 Problemas de Programação Dinâmica: O Problema do Caminho Mínimo

Conforme apresentado no Capítulo 2, um problema típico de programação dinâmica pode ser modelado como um conjunto constituído de um nó fonte e um nó destino com n -estágios intermediários e m -estados em cada estágio, como ilustrado na Figura 5.1, para $n=m=3$. O custo associado da transição do i -ésimo estado pertencente ao x -ésimo estágio para o j -ésimo estado pertencente ao estágio seguinte, $x+1$, é dado pela variável constituída de índices duplos denotada por $d_{xi,(x+1)j}$, onde x e $(x+1)$ são os índices dos estágios, e i e j são os índices dos estados em cada estágio. Na Figura 5.1, o custo identificado por d é denotado por $d_{11,22}$. O objetivo é encontrar um caminho composto de um e somente um estado em cada estágio, partindo da fonte com direção ao destino (abordagem *forward*), e que satisfaça algum critério.

Figura 5.1: Problema do caminho mínimo.

Portanto, o objetivo do problema de programação dinâmica considerado nesta tese, o problema do caminho mínimo, é encontrar o menor caminho entre todos os caminhos possíveis, que se inicializam no nó fonte e finalizam no nó destino, passando por um único estado em cada estágio. Admite-se que cada nó de estado possa ser convenientemente considerado como um elemento processador do i -ésimo estado pertencente ao x -ésimo estágio.

Vale destacar que o problema do caminho mínimo, em um grafo em camadas, possui as duas características que o tornam solucionável por programação dinâmica: subestrutura ótima e superposição de subproblemas. A subestrutura ótima (a solução para o problema contém em seu interior soluções ótimas para subproblemas, os quais são independentes) pode ser percebida na possibilidade de dividir o caminho total em subcaminhos, onde cada um deles pode ser resolvido de forma independente, e a composição de cada menor caminho nos subcaminhos resulta no menor caminho total. No caso dos subproblemas superpostos (o algoritmo para o problema resolve os mesmos subproblemas repetidas vezes) é possível notar que o algoritmo aplicado para encontrar o menor caminho em cada subcaminho é o mesmo.

5.2.1 Redes Neurais Artificiais Aplicadas ao Problema do Caminho Mínimo

Rauch e Winarske (1988) publicaram o primeiro trabalho que, motivado pelo trabalho de Hopfield e Tank (1985), utilizou redes neurais para determinar a melhor rota para a transmissão de dados em redes de comunicação. Os autores usaram a rede de Hopfield e a projetaram para encontrar o menor caminho entre um par de nós (origem-destino), porém,

para cada novo par origem-destino, a configuração da rede necessitava ser alterada. Outro aspecto relevante é que a rede conseguia encontrar o menor caminho, considerando um determinado par origem-destino, somente quando o número de *links* (conexões) que o caminho possuía era conhecido. Esta solução não é apropriada para casos reais, porque na prática, o custo dos *links* em uma rede de comunicação varia no tempo e, conseqüentemente, as conexões também devem ser alteradas. A definição dos valores dos parâmetros associados aos termos da função de energia (originalmente definida para o problema do caixeiro viajante em Hopfield e Tank (1985)) é outro ponto negativo, pois estes parâmetros exercem grande influência na convergência da rede e na qualidade da solução.

Três anos depois, Thomopoulos et al. (1991) propuseram a solução de algumas limitações presentes no trabalho de Rauch e Winarske (1988). Nesta nova proposta, não era mais necessário conhecer *a priori* o número de conexões entre um par de nós origem-destino para determinar o menor caminho. O algoritmo proposto foi usado para obter rotas para múltiplos pares de nós origem-destino. Com base nos experimentos realizados, os autores identificaram que os resultados variavam de acordo com as condições iniciais da rede, ou seja, conforme a definição dos pesos iniciais dos neurônios da rede. Como em Rauch e Winarske (1988), a definição adequada dos valores dos parâmetros associados aos termos da função de energia também foi necessária.

Em Chiu et al. (1991) foi proposta uma rede na qual os termos de otimização e de restrições eram tratados em um único estágio (como nos trabalhos anteriormente descritos), conseqüentemente, tornou-se necessário especificar os parâmetros de ponderação associados a cada termo de restrição. A tarefa de encontrar os valores adequados para as constantes de ponderação, relativos aos termos de restrições, necessárias para a convergência da rede, tornou-se um fator complicador da abordagem. De acordo com as constantes escolhidas, a interferência entre os termos de restrições e o termo de otimização afetava o número de estados mínimos da rede, refletindo assim na solução obtida.

Ali e Kamoun (1993) apresentaram uma versão modificada da rede neural de Hopfield, a qual objetivava minimizar o tempo de atraso na entrega dos pacotes em um problema de roteamento em redes de computadores. Para realizar tal tarefa, apresentou-se uma modificação na função de energia da rede de Hopfield, a qual possuía cinco parâmetros: o primeiro minimizava o custo total do caminho levando em consideração os *links* (conexão entre dois nós) existentes; o segundo parâmetro prevenia a não existência de *links* no caminho escolhido; o terceiro parâmetro tinha seu valor igual a zero se todos os nós escolhidos

pertencessem a um caminho válido; o quarto parâmetro movia a saída da rede em direção a um ponto de equilíbrio; e por fim, o quinto parâmetro assegurava que o nó fonte e o destino estivessem no caminho. Uma vantagem do modelo proposto era que o modelamento dos dados (custo das conexões e a conectividade dos nós) estava no vetor de entrada (*input bias*) e não nas interconexões neurais, ou seja, a rede possuía a capacidade de adaptar-se a mudanças na topologia da rede e nos custos das conexões. Outras vantagens destacadas pelos autores eram a boa convergência e a baixa complexidade de implementação. A principal desvantagem desta abordagem está na dificuldade de determinar apropriadamente os valores para os parâmetros; adicionalmente, segundo Araújo et al. (2001), a rede proposta por Ali e Kamoun (1993) freqüentemente não converge para uma solução final válida quando o número de nós do grafo é elevado e os resultados são piores quando comparados com as soluções ótimas obtidas pelo algoritmo de Dijkstra (algoritmo específico para solução de problemas de caminho mínimo em grafos).

Alguns anos mais tarde, Park e Choi (1998) publicaram um artigo que propôs uma evolução do método de Ali e Kamoun (1993), o qual conseguia determinar o menor caminho entre um nó origem e vários nós destinos. A abordagem proposta era composta por três processos; primeiro, a rede de Hopfield era utilizada para determinar a ordem de roteamento entre a origem e os múltiplos destinos; em seguida, um procedimento era aplicado para simplificar o problema, ou seja, o número de nós era reduzido para acelerar a busca pelo menor caminho; e por último, um novo algoritmo de roteamento baseado na rede de Hopfield era simultaneamente aplicado ao processo anterior para encontrar todos os caminhos. A função de energia era quase idêntica à função proposta por Ali e Kamoun (1993), somente com uma alteração no último termo, sendo assim, apresentava as mesmas vantagens da abordagem de Ali e Kamoun (1993). Em contrapartida, a dificuldade de determinar os valores dos parâmetros era o fator negativo. Com base nos experimentos realizados pelos autores (um grafo com 20 nós, uma origem e três destinos), a abordagem proposta apresentou menor tempo de convergência (número de épocas) em relação à abordagem de Ali e Kamoun (1993), e a qualidade das soluções foram as mesmas obtidas pelo algoritmo de Dijkstra.

Em Wang (1998), duas redes neurais recorrentes foram propostas para a resolução do problema do caminho mínimo, denominadas redes neurais primais e duais. Em ambos os casos, o problema do caminho mínimo era formulado como um problema de programação linear; sendo assim, o trabalho que atacava diretamente o problema foi descrito em outro artigo do mesmo autor em Wang (1993). Neste artigo, uma rede neural recorrente para a

resolução de problemas de programação linear foi proposta, em que os termos de otimização e de restrições associados com o problema eram tratados em um único estágio, ou seja, definiu-se uma equação para a dinâmica da rede, na qual os dois primeiros termos forçavam a penalização pela violação das restrições e o terceiro termo minimizava a função objetivo. Outra equação (esta possuía um parâmetro) definia a função de ativação, a qual restringia a saída dos neurônios dentro de um hipercubo, definido pelas restrições. Conseqüentemente, tornou-se necessário especificar corretamente os parâmetros de ponderação associados a cada equação. As vantagens destacadas pelo autor eram a facilidade de implementação em *hardware* e a capacidade de resolução de problemas de programação linear em tempo real e de larga escala.

O artigo publicado por Ahn et al. (2001) seguiu a linha dos trabalhos propostos por Ali e Kamoun (1993) e Park e Choi (1998), ou seja, definiu-se uma nova função de energia para a rede, a qual modelava a função objetivo do problema do caminho mínimo e suas restrições através dos termos que a compunham. Neste caso, a função de energia possuía sete termos. Os quatro primeiros termos eram os mesmos propostos por Ali e Kamoun (1993), sendo que o quinto era a alteração proposta por Park e Choi (1998). Os dois últimos termos foram a contribuição do trabalho de Ahn et al. (2001). Os autores relataram que estes novos termos eliminavam a possibilidade de *loops* ou partições no caminho, e também aceleravam e aprimoravam a convergência da rede para uma solução ótima. Segundo os autores, a rede utilizava todas as informações disponíveis nos neurônios periféricos. Além disso, fazia uso do alto conhecimento correlacionado com os neurônios locais. Desta forma, o algoritmo proposto permitia uma convergência rápida e conseguia obter soluções ótimas. O experimento realizado consistiu de 10000 execuções, alterando aleatoriamente os custos das conexões entre os nós de um grafo com 14 nós. Os testes também foram executados utilizando as redes apresentadas em Ali e Kamoun (1993) e Park e Choi (1998). De acordo com os resultados, o desempenho da rede proposta foi 20% superior em relação às demais, e cerca de sete a oito por cento mais precisa, ou seja, 93,82% dos casos obtiveram soluções ótimas. A desvantagem desta abordagem, como em Ali e Kamoun (1993) e Park e Choi (1998), está na determinação dos valores apropriados para os parâmetros associados a cada termo da função de energia da rede.

O trabalho desenvolvido por Araújo et al. (2001), apresentou uma nova rede neural para a solução do problema do caminho mínimo para roteamento de pacotes em redes de computadores. A solução proposta estendeu a arquitetura tradicional da rede de Hopfield,

introduzindo uma segunda camada de neurônios que automaticamente satisfazia o conjunto de restrições, assegurando então a obtenção de uma solução válida. O problema do caminho mínimo era definido como um problema de programação linear inteira, sendo assim, uma camada da rede neural representava as variáveis dependentes e a outra camada representava as variáveis independentes. A arquitetura necessitava de poucos neurônios, sendo que a quantidade era igual ao número de arcos no grafo, ao contrário das abordagens de Ali e Kamoun (1993) e Park e Choi (1998), nas quais o número de neurônios era igual ao número de nós do grafo ao quadrado. Somente no pior caso, onde todos os nós do grafo eram conectados uns aos outros, que o número de neurônios era igual. Por outro lado, o número reduzido de neurônios dificultava a adaptação da rede para mudanças de topologia, tais como, novos nós ou conexões. A rede também era muito sensível a estes dois parâmetros. Contudo, a rede permanecia com a principal vantagem da abordagem apresentada por Ali e Kamoun (1993), que era a habilidade de adaptação às mudanças dos custos dos arcos, pois estas informações eram armazenadas no vetor de entrada (*input bias*). Com base nos experimentos, a rede apresentou melhor desempenho computacional quando comparada com o trabalho de Park e Choi (1998), porém, a qualidade da solução apresentada foi pior.

Em Silva et al. (2007) foi apresentada uma rede recorrente que fornecia resultados satisfatórios para problemas de Programação Dinâmica, mas a convergência em direção aos pontos de equilíbrio era lenta para os casos complexos, sendo que este fato pode comprometer a aplicabilidade efetiva da abordagem proposta. Outro aspecto importante é que a rede utilizava um parâmetro de inicialização obtido experimentalmente.

Um resumo dos trabalhos descritos anteriormente é apresentado na Tabela 5.1, destacando as suas principais ressalvas.

Tabela 5.1: Resumo dos trabalhos aplicados ao PCM usando RNA.

Autores	Descrição	Ressalvas
Rauch e Winarske (1988)	Utilizam a rede de Hopfield, originalmente definida para o problema do caixeiro viajante.	Deve-se conhecer o número de conexões do menor caminho entre um par de nós origem-destino.
Thomopoulos et al. (1991)	Utilizam a rede de Hopfield, originalmente definida para o problema do caixeiro viajante.	Resultados variam de acordo com as condições iniciais da rede.
Chiu et al. (1991)	Alteram a função de energia da Rede de Hopfield.	Deve-se especificar os parâmetros de ponderação associados a cada termo da função de energia.
Ali e Kamoun (1993)	O modelamento dos dados (custo das conexões e a conectividade dos nós) está no vetor de entrada (<i>input bias</i>) e não nas interconexões neurais.	Deve-se especificar os parâmetros de ponderação associados a cada termo da função de energia.
Park e Choi (1998)	Abordagem composta por três processos. Altera a função de energia proposta por Ali e Kamoun (1993).	Deve-se especificar os parâmetros de ponderação associados a cada termo da função de energia.
Wang (1998)	Utiliza uma rede neural recorrente para a resolução de problemas de programação linear, proposta em Wang (1993).	Deve-se especificar os parâmetros de ponderação associados a cada termo da função de energia.
Ahn et al. (2001)	Alteram a função de energia da rede de Hopfield baseado nos trabalhos de Ali e Kamoun (1993) e Park e Choi (1998), e adicionam dois novos termos.	Deve-se especificar os parâmetros de ponderação associados a cada termo da função de energia.
Araújo et al. (2001)	Estendem a arquitetura tradicional da rede de Hopfield, introduzindo uma segunda camada de neurônios.	Dificuldade em adaptar-se a mudanças de topologia. Sensível a parâmetros.
Silva et al. (2007)	Rede recorrente para problemas de Programação Dinâmica.	A convergência em direção aos pontos de equilíbrio é lenta para casos complexos. Utiliza um parâmetro de inicialização obtido de forma experimental.

5.2.2 Algoritmos Genéticos Aplicados ao Problema do Caminho Mínimo

Gen et al. (1997) propõem uma nova metodologia de codificação dos cromossomos para o problema do caminho mínimo, a qual é denominada pelos autores do método de codificação baseada em prioridade (*priority-based encoding method*). As posições do cromossomo são utilizadas como identificador do nó. Os valores dos genes são usados como valores de prioridade dos nós.

Suponha um problema em que existam seis nós: 1 (nó origem), 2, 3, 4, 5 e 6 (nó destino), conforme mostra a Figura 5.2. Neste grafo, o valor da aresta que conecta dois nós representa o custo do caminho entre eles.

Figura 5.2: Um simples grafo com 6 nós e 10 arcos.

Considere um cromossomo gerado aleatoriamente, como ilustrado na Figura 5.3. O nó origem 1, possui prioridade 3, e está ligado a outros três nós, 2, 3 e 4, com prioridades 5, 4 e 6, respectivamente. O algoritmo escolhe o nó com ligação ao nó 1 com maior prioridade, ou seja, entre os nós 2, 3 e 4 o que tem maior prioridade é o nó 4 (prioridade igual a seis). Portanto, o nó 4 é acrescentado ao caminho entre o nó 1 e o nó 6. Em seguida, deve-se escolher o nó com maior prioridade e que tenha ligação ao nó 4. Este processo se repete até que o caminho esteja completo (1, 4,..., 6). Pode-se facilmente notar que qualquer permutação dos genes produzirá um caminho.

1	2	3	4	5	6
3	5	4	6	2	1

Figura 5.3: Codificação baseada em prioridade.

A abordagem foi testada para três diferentes problemas. Cada problema foi executado 400 vezes, com valores iniciais aleatórios. No primeiro teste, com seis nós e 10 arcos, a resposta obtida em 100% dos testes foi 10, que equivale à solução ótima. No segundo teste,

considerando 32 nós e 66 arcos, a melhor solução obtida foi 205, que corresponde à solução apresentada na literatura como ótima. A pior solução encontrada foi 220 e a média dos valores obtidos como resultado foi 205,15. Os autores também relatam que a solução ótima foi encontrada em 98% das execuções. Por fim, o terceiro teste foi realizado com 70 nós e 211 arcos. A melhor, a pior e a solução média obtidas foram respectivamente 2708, 3040 e 2745,53. A solução ótima foi encontrada em 64% das execuções. Vale destacar que, nos testes realizados, os autores fizeram variações no tamanho da população (de 10 a 100 indivíduos) e no número de gerações (de 100 a 3000 gerações) do algoritmo genético. A solução ótima foi encontrada com maior frequência quando os valores dos parâmetros de tamanho de população e número de gerações foram aumentados.

Para evitar que um nó que não leva ao nó destino (nó morto) seja incluído no caminho, a abordagem realizava testes para verificar se um nó é ou não morto, verificando se algum dos caminhos a partir dele leva ao nó destino. Em problemas com grandes espaços de busca, este procedimento elevaria muito o custo computacional da abordagem. Assim, os autores optaram por criar uma conexão “virtual” entre o nó morto e o nó destino, associando a ela um parâmetro de penalidade (custo), mas não deixam claro como o valor deste parâmetro é atribuído. Experimentos com problemas complexos não foram realizados, e os autores mencionam que o sucesso obtido com a abordagem em problemas simples pode ser um indício de que algoritmos genéticos são bons em resolver também problemas complexos.

No artigo publicado por Munemoto et al. (1998), um algoritmo genético foi proposto para encontrar um caminho mínimo em uma rede de computadores com ou sem fio. O algoritmo genético utilizava cromossomos de tamanho variável para codificar o problema. O método de cruzamento proposto era bastante restrito, uma vez que os pontos de cruzamento deviam ser genes que possuíam o mesmo valor e estavam na mesma posição em ambos os cromossomos pais. Assim, poucos cromossomos realizavam cruzamento, dificultando a exploração do espaço de busca e aumentando, ou até mesmo impedindo, a obtenção da solução. O método de mutação selecionava aleatoriamente um ponto de mutação (gene) em um cromossomo. Em seguida, utilizava o algoritmo de Dijkstra para gerar um menor caminho parcial, do nó origem àquele ponto, e outro daquele ponto ao nó destino.

O algoritmo genético requeria uma população relativamente grande para a obtenção de uma solução ótima, devido às restrições sobre o mecanismo de cruzamento. A aplicação do algoritmo de Dijkstra no método de mutação inseriu elevado custo computacional ao método.

Desta forma, o mesmo não podia ser aplicado a problemas extensos ou que exigiam rapidez na solução.

Ahn e Ramakrishna (2002) apresentaram uma abordagem baseada em algoritmos genéticos para resolver problemas de caminho mínimo no roteamento de pacotes em redes de computadores de grande porte ou de tempo-real. Esta abordagem propôs que o tamanho da população fosse definido dinamicamente de acordo com a aplicação em questão, por meio de uma Equação de Tamanho da População, com a finalidade de diminuir o elevado custo computacional presente na maioria das abordagens que utilizavam AG para o mesmo fim. Os autores perceberam que, na tentativa de obter boas soluções, o tamanho da população aumentava exponencialmente com a complexidade do problema. Na população, as posições do cromossomo foram utilizadas como identificador do nó. Os valores dos genes eram números inteiros positivos que representavam a ordem dos nós através de um caminho. A função de aptidão utilizada realizava a soma dos custos associados a cada caminho (representado por um cromossomo), e considerava como o melhor indivíduo aquele que possuísse o menor valor de aptidão.

Os experimentos foram realizados em duas etapas. Na primeira, foi fixada uma topologia de rede com 20 nós, tendo o nó 1 como origem e o nó 20 como destino. A abordagem proposta foi comparada com o algoritmo de Dijkstra (porque o mesmo sempre converge para a solução ótima), e com outra metodologia, proposta por Munemoto et al. (1998), sendo que todas foram testadas com populações de mesmo tamanho. A abordagem proposta obteve o resultado ótimo, com um caminho mínimo de custo igual a 142, também encontrado pelo algoritmo de Dijkstra, porém, com um tempo de processamento menor. O tempo de processamento para a abordagem proposta foi de 0,067 segundos, enquanto que para o Algoritmo de Dijkstra foi de 0,14 segundos. A abordagem de Munemoto et al. (1998) obteve o custo de 187. O tempo de convergência da abordagem proposta também é menor que o tempo de convergência de Munemoto et al. (1998), atingindo o valor ótimo em aproximadamente 9 gerações. Na segunda etapa, diferentes topologias foram utilizadas, com a quantidade de nós variando de 15 a 50. Os melhores resultados, considerando precisão e custo computacional, continuaram sendo obtidos pela abordagem proposta.

Como na maioria dos métodos que utilizam AG, novos operadores de cruzamento e mutação tiveram que ser definidos, visando para tanto garantir a validade dos caminhos (soluções). O método de cruzamento proposto pela abordagem gerava cromossomos representativos de caminhos ineficazes, o que forçou os autores a introduzir uma Função de

Reparo dos cromossomos, a qual deveria ser aplicada a cada nova geração para evitar que caminhos inexistentes ou com ciclo permanecessem na população e gerassem descendentes também infactíveis.

Em Fu et al. (2003), o problema de encontrar o caminho com menor custo de *download* de um endereço *web* a outro foi tratado. Este problema ocorre porque muitos *browsers* fazem o *download* automático de objetos encontrados em documentos HTML, tais como gráficos, Java *applets* e arquivos de áudio e vídeo, muitas vezes até de forma repetida, sendo que há uma quantidade restrita de memória disponível para armazenar tais objetos, o que limita a escolha do caminho.

Primeiramente, foi apresentada como solução uma adaptação no algoritmo de Dijkstra, mas o método se mostrava infactível por utilizar muita memória. Em seguida, uma abordagem baseada em Busca Tabu (BT) foi proposta, mas não encontrou bons resultados em função de sua dificuldade em vasculhar novas regiões do espaço de busca, pois se trata de um algoritmo de busca heurística local. Por fim, um algoritmo genético tradicional foi apresentado, produzindo soluções aceitáveis. Com base nestes experimentos, os autores optaram por propor uma quarta abordagem, combinando a habilidade da Busca Tabu em encontrar um ótimo local com a eficiência dos algoritmos genéticos em explorar o espaço de busca, de forma então a aproveitar as potencialidades de ambas. Observando-se os resultados, percebeu-se que as melhores soluções foram obtidas com esta combinação. Entretanto, os autores não discutiram algumas especificidades que podiam ser percebidas através dos experimentos, as quais serão discutidas a seguir.

Considerando dois dos cinco experimentos apresentados, é possível notar uma diferença pequena no valor do menor caminho, mas uma diferença de tempo bastante grande entre as abordagens. Em um dos experimentos, considerando um total de 100 nós (um nó representa um endereço *web*), uma média de 10 objetos por nó e uma memória de tamanho igual a 50, os métodos de Dijkstra, BT, AG e AG+BT, obtiveram as respectivas soluções: 103, 101, 115 e 99, com tempos de 1, 2, 10, e 214 segundos. Em um segundo experimento, com 1000 nós, uma média de 20 objetos por nó e uma memória de tamanho igual a 100, os métodos de Dijkstra, BT, AG e AG+BT, obtiveram as respectivas soluções: 163, 161, 175 e 155, com tempos de 3, 53, 537, e 76015 segundos. Deste modo, apesar de os melhores resultados terem sido obtidos da combinação de AG+BT, os tempos exigidos por esta abordagem híbrida são muito superiores aos exigidos pelas abordagens individuais. Assim, a decisão sobre qual abordagem é a melhor depende da necessidade da aplicação.

Wu e Ruan (2004) propuseram um algoritmo genético (G-C GA) com restrições sobre os genes para garantir que cada cromossomo represente um caminho factível sem ciclos. Os autores mencionaram que, quando comparado com outros algoritmos genéticos para o mesmo fim, o G-C GA era mais flexível, rápido e preciso, não importando se o grafo era direcionado ou não. O problema em análise nos experimentos era bastante simples e realizado em um grafo não-direcionado de 100 vértices. O G-C GA utilizou uma população de 10 indivíduos e um critério de parada de no máximo 100 gerações, encontrando sempre a solução ótima com um número de gerações inferior ao máximo permitido. Os resultados foram comparados com os obtidos pelo algoritmo de Floyd, comprovando a obtenção da solução ótima apresentada na literatura. Nenhum experimento de maior complexidade foi realizado como forma de justificar a aplicação de um algoritmo genético na solução do problema, ao invés de simplesmente utilizar um método tradicional como o algoritmo de Floyd.

Nas conclusões, os autores declararam que, através dos experimentos, só foi possível demonstrar que o G-C GA era eficiente em resolver problemas simples de busca de caminho mínimo, e que serviria apenas como base para o desenvolvimento de outra abordagem capaz de resolver problemas mais complexos.

No trabalho desenvolvido por Li et al. (2006) foi utilizado um Algoritmo Genético Específico (AGE) para encontrar um menor caminho em um sistema de transporte inteligente. Um cromossomo representava um caminho, onde cada gene descrevia um nó do caminho. O mecanismo de cruzamento utilizado foi o proposto por Wu e Ruan (2004). O operador de mutação foi redefinido, utilizando a mesma metodologia proposta em (Munemoto et al., 1998), com a diferença de que, ao invés de utilizar o algoritmo de Dijkstra, os autores usaram o algoritmo A*. As probabilidades de cruzamento e mutação foram ajustadas de forma adaptativa por um Sistema de Controle *Fuzzy*.

Os experimentos foram realizados sobre cinco grafos, cada um consistindo respectivamente de 16 nós e 24 arcos, 32 nós e 55 arcos, 43 nós e 75 arcos, 62 nós e 111 arcos, e 78 nós e 139 arcos. O tamanho da população variou de 10 a 50 indivíduos, com um total de gerações entre 50 e 200. Os autores relataram que o AGE obteve a solução ótima em cerca de 90% dos testes, com um tempo variando de 2 a 17 segundos, dependendo da quantidade de nós.

A introdução do algoritmo A* no mecanismo de mutação atrasou o processo de busca, uma vez que este algoritmo não encontra caminhos parciais ótimos, apesar de encontrar um caminho com bastante rapidez. A utilização de um Sistema de Controle *Fuzzy* também

retardou o processo de convergência, pois ele era aplicado na obtenção das probabilidades de ambos os operadores genéticos por repetidas vezes no decorrer do processo.

No artigo publicado por Li et al. (2006b) foi apresentada uma versão melhorada do AG proposto por Gen et al. (1997), propondo alterações na codificação cromossômica, na forma de criação da população inicial e no operador de mutação, além de definir uma função de *fitness* semelhante à apresentada em Gen et al. (1997), mas com uma alteração que a torna variável. A abordagem foi dividida em duas fases. Na primeira, uma grande população era considerada para que houvesse uma melhor exploração do espaço de busca, implicando num aumento no tempo de processamento. Na segunda, o tamanho da população era reduzido, pois a busca estava concentrada em uma região específica do espaço. Então, para melhorar a precisão, os autores associaram ao AG o método de Subida da Encosta para realização da busca local.

Os experimentos mostrados em Gen et al. (1997) foram repetidos utilizando a abordagem em questão, percebendo-se redução do número de iterações e aumento da probabilidade de convergência quando comparado com a abordagem proposta em Gen et al. (1997). Entretanto, as alterações propostas para alcançar esta melhora inseriram quatro parâmetros de ponderação: α , β , δ e γ . O termo α era um parâmetro que dinamiza a função de *fitness*, β era um parâmetro associado à divisão do número total de iterações entre as duas fases da abordagem, δ era um fator associado à redução do tamanho da população na segunda fase da abordagem, e γ era um parâmetro relacionado com o número de iterações, no qual o método da Subida da Encosta deveria ser colocado em execução. Estes parâmetros eram definidos experimentalmente de acordo com a aplicação.

Lin et al. (2008) propuseram um algoritmo genético, baseado no trabalho de Ahn e Ramakrishna (2002), para um sistema de indicação de rotas para usuários dentro de seus veículos, por meio da utilização de dispositivos móveis com capacidades de processamento e memória limitadas. Por se tratar de um sistema real, variáveis como velocidade do tráfego, limites de velocidade e capacidade de velocidade do veículo deveriam ser consideradas, além da variável de distância. Assim, ao invés de utilizar o termo “menor caminho”, o trabalho utilizava o termo “menor tempo”, considerando para tanto a aplicação da equação $\text{Tempo} = \text{Distância} / \text{Velocidade}$.

Os experimentos foram realizados, primeiramente, em um ambiente virtual. Foram utilizados mapas definidos por matrizes quadradas de 4x4 (16 nós), 8x8 (64 nós), 16x16 (256

nós) e 32x32 (1024 nós). As distâncias entre os nós foram fixadas em 20 e as velocidades variavam de 2 a 10. O algoritmo genético convergiu com menos de 20 iterações para todas as matrizes quadradas mencionadas, variando o tamanho da população de 20 a 160 indivíduos. Entretanto, mesmo aumentando o tamanho da população, apenas soluções aproximadas foram encontradas, podendo haver uma diferença de mais de 50% entre a solução ótima e a solução aproximada encontrada.

Novos experimentos foram executados em um ambiente real, considerando mapas de uma cidade de Taiwan, com 8039 nós e cinco níveis diferentes de limites de velocidade. O AG convergiu com uma média de 30 iterações, mas também encontrava apenas soluções aproximadas. Considerando uma população de 40 indivíduos, a diferença entre a solução ótima e a solução encontrada foi de 223%. Para conseguir reduzir esta diferença para aproximadamente 50% (a qual ainda permanece grande), o tamanho da população teve que ser aumentado para 1280 indivíduos, o que se tornou um fator crítico para a abordagem, segundo os autores.

Inicialmente, os autores mencionaram que sua abordagem seria comparada ao algoritmo de Dijkstra. No entanto, com o aumento do número de nós, a quantidade de memória requerida pelo algoritmo de Dijkstra ultrapassou o limite de memória do dispositivo móvel. Deste modo, apenas os experimentos com o AG foram apresentados.

Na Tabela 5.2 é relacionada uma síntese dos trabalhos apresentados, destacando as suas principais ressalvas.

Tabela 5.2: Resumo dos trabalhos aplicados ao PCM usando AG.

Autores	Descrição	Ressalvas
Gen et al. (1997)	Propõem uma nova metodologia de codificação dos cromossomos baseada em prioridades (<i>priority-based encoding method</i>).	Um parâmetro de penalidade é utilizado.
Munemoto et al. (1998)	Utilizam cromossomos de tamanho variável; operador de cruzamento é bastante restritivo; usa o algoritmo de Dijkstra na mutação.	Requer uma população relativamente grande para a obtenção de uma solução ótima; aplicação do algoritmo de Dijkstra insere elevado custo computacional.
Ahn e Ramakrishna (2002)	O tamanho da população é definido dinamicamente.	O operador de cruzamento gera caminhos inactíveis; a função de reparo para caminhos inactíveis necessita de ajuste de um parâmetro.
Fu et al. (2003)	Associam o método Busca Tabu ao algoritmo genético.	Apresenta elevados tempos de processamento.
Wu e Ruan (2004)	Propõem algoritmo genético com restrições sobre os genes.	Faltaram experimentos para comprovar a eficiência da abordagem proposta.
Li et al. (2006)	Utilizam cruzamento proposto por Wu e Ruan (2004); usam o algoritmo A* na mutação e ajustam as taxas do cruzamento e mutação por um Sistema de Controle <i>Fuzzy</i> .	A introdução do algoritmo A* na mutação pode atrasar o processo de busca; a utilização do Sistema de Controle <i>Fuzzy</i> pode aumentar o tempo convergência.
Li et al. (2006b)	Versão melhorada da abordagem de Gen et al. (1997); associam o método Subida da Encosta ao algoritmo genético.	Utilização de parâmetros de ponderação.
Lin et al. (2008)	Propõem um algoritmo genético baseado no trabalho de Ahn e Ramakrishna (2002) para uma aplicação real	Encontra apenas soluções próximas das ótimas; o tamanho da população deve ser grande, conseqüentemente, o processo de busca é lento.

5.2.3 Modelamento do Problema do Caminho Mínimo através da Rede de Hopfield Genética

A implementação deste problema pode ser feita a partir das definições apresentadas na Seção 5.1. Deste modo, redefine-se o vetor p como sendo o vetor que contém os índices dos estados nos diversos estágios intermediários. O valor do elemento p_i representa qual estado está ativado no i -ésimo estágio intermediário, ou seja:

$$p_i \in \{1, \dots, m\}, \text{ onde } i \in \{1, \dots, n\}$$

sendo m o número de estados em cada estágio, e n o número total de estágios intermediários. O vetor $v(p)$ e a matriz $V(p)$ são dados por:

$$v(p) = \begin{bmatrix} \delta(p_1) \\ \delta(p_2) \\ \delta(p_3) \\ \dots \\ \delta(p_n) \end{bmatrix}_{n \times m} \quad V(p) = \begin{bmatrix} \delta(p_1)^T \\ \delta(p_2)^T \\ \delta(p_3)^T \\ \dots \\ \delta(p_n)^T \end{bmatrix}_{n \times m}$$

onde $\delta(k) \in \mathbb{R}^n$ é um vetor coluna correspondente à k -ésima coluna de δ .

Como exemplo ilustrativo, para um problema com 5 estágios e com 4 estados por estágios ($n = 5$ e $m = 4$), e tendo a sequência de estados seguintes como solução: estado 3 no estágio 1, estado 2 no estágio 2, estado 4 no estágio 3, estado 4 no estágio 4, e estado 1 no estágio 5; os vetores p e $v(p)$, e a matriz $V(p)$ são dados por:

$$p^T = [3 \ 2 \ 4 \ 4 \ 1]$$

$$v(p)^T = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\delta(p_1)^T \quad \delta(p_2)^T \quad \delta(p_3)^T \quad \delta(p_4)^T \quad \delta(p_5)^T$$

$$V(p) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Deduzem-se, a seguir, as equações para os parâmetros internos da rede de Hopfield Genética.

Dedução das Equações para T^{val} e s

As equações de T^{val} e s do subespaço-válido para o problema de programação dinâmica são desenvolvidas utilizando os conceitos descritos na Seção 5.1. A restrição para o problema em questão mostra que apenas um neurônio por estágio deve estar ativado. Esta restrição significa que apenas um e somente um elemento de cada linha da matriz $V(p)$ deve possuir valor igual a um, ou seja:

$$[V(p)]_{ij} \in \{1, 0\}$$

$$\sum_{j=1}^m [V(p)]_{ij} = 1 \quad (5.22)$$

Deve-se ressaltar que não há nenhuma restrição para as colunas da matriz $V(p)$, visto que os índices que representam os estados podem ser repetidos em vários estágios. Assim, como o vetor soma das linhas é dado por $l = V.o^m$, um subespaço $S \in \mathbb{R}^{n \times m}$ para o problema pode ser representado por:

$$S = V = \frac{1}{m} o^n . o^{mT} \quad (5.23)$$

A equação (5.23) garante que a soma dos elementos de cada linha da matriz V possui valor igual a um. Portanto, o termo $T^{val}.V$ da equação do subespaço-válido deve garantir que a soma dos elementos de cada linha da matriz V seja igual a zero. Este procedimento pode ser realizado com a utilização da matriz R^m , isto é, a pós-multiplicação de V por R^m transforma a soma das linhas de V para zero, ou seja:

$$I^n . V . R^m = T^{val} . V \quad (5.24)$$

$$V . R^m = T^{val} . V$$

Substituindo-se as equações (5.23) e (5.24) na equação do subespaço-válido, tem-se:

$$V = I^n . V . R^m + \frac{1}{m} o^n . o^{mT} \quad (5.25)$$

Aplicando-se em (5.25) o operador $vec(.)$ definido em (5.14), tem-se:

$$vec(V) = vec(I^n . V . R^m) + \frac{1}{m} vec(o^n . o^{mT})$$

$$vec(V) = (I^n \otimes R^m) . vec(V) + \frac{1}{m} (o^n \otimes o^m) \quad (5.26)$$

Substituindo-se $vec(V)$ por v na equação (5.26), tem-se:

$$v = (I^n \otimes R^m).v + \frac{1}{m} (o^n \otimes o^m) \quad (5.27)$$

Assim, os parâmetros T^{val} e s do subespaço-válido para o problema são dados por:

$$T^{val} = (I^n \otimes R^m) \quad (5.28)$$

$$s = \frac{1}{m} (o^n \otimes o^m) \quad (5.29)$$

destacando-se que as equações (5.28) e (5.29) obedecem as propriedades do subespaço-válido, isto é, $T^{val}.T^{val} = T^{val}$ e $T^{val}.s = 0$.

Definição dos Parâmetros do Algoritmo Genético Para Obtenção do Termo E^{ot}

Os parâmetros do algoritmo genético utilizado para a otimização da função objetivo (E^{ot}) foram:

- **Representação Genética:** Para um problema do caminho mínimo constituído por três estágios ($n=3$) e por três estados ($m=3$) em cada estágio, o tamanho dos cromossomos é igual a $n*m$ e cada cromossomo é codificado como um vetor de números reais, no qual a cada m posições do vetor se representam os estados do estágio n , como mostra a Figura 5.4.

Figura 5.4: Representação genética para o problema do caminho mínimo.

- **População Inicial:** Formada pelos valores de v previamente obtidos com a aplicação da rede neural, por meio dos passos (I) e (II) mostrados na Figura 4.2; 10% dos cromossomos foram gerados em torno deste v repassado pela rede neural, objetivando inserir uma tendência a regiões supostamente promissoras do espaço de busca; e os demais cromossomos foram gerados aleatoriamente.
- **Tamanho da População:** O número de indivíduos foi definido empiricamente de acordo com a dimensão de cada problema. Este número se inicia em 50 indivíduos e é incrementado à medida que o número de estágios e estados aumenta.

- Funções de Avaliação ou *Fitness*: O termo de energia E^{ot} da rede de Hopfield Genética para o problema em questão é projetado com o objetivo de que a solução ótima corresponda ao menor percurso entre o nó fonte e o nó destino. Quando o termo de energia E^{ot} for minimizado, a solução ótima desta função de energia corresponderá ao estado de menor energia da rede. A função E^{ot} para o problema é definida por:

$$E^{ot} = \frac{1}{4} \left[\underbrace{\sum_{x=1}^{n-1} \sum_{i=1}^m \sum_{j=1}^m d_{xi,(x+1)j} \cdot v_{xi} \cdot v_{(x+1)j}}_{1o. \text{ Termo}} + \underbrace{\sum_{x=2}^n \sum_{i=1}^m \sum_{j=1}^m d_{(x-1)j,xi} \cdot v_{xi} \cdot v_{(x-1)j}}_{2o. \text{ Termo}} \right] + \left[\underbrace{\sum_{x=1}^1 \sum_{i=1}^m d_{fonte,xi} \cdot v_{xi}}_{3o. \text{ Termo}} + \underbrace{\sum_{x=n}^n \sum_{i=1}^m d_{xi,destino} \cdot v_{xi}}_{4o. \text{ Termo}} \right] \quad (5.30)$$

Nesta equação, o primeiro termo fornece o custo de conexão entre cada neurônio v_{xi} , pertencente ao estágio x , e todos os outros neurônios que estão no estágio posterior $x+1$; enquanto o segundo termo fornece o custo de conexão de cada neurônio v_{xi} com todos os outros neurônios que estão no estágio anterior $x-1$. O terceiro termo fornece o custo de conexão entre o nó fonte e todos os outros neurônios que estão no primeiro estágio, enquanto o quarto termo fornece o custo de conexão do nó destino com todos os outros neurônios que estão no último estágio. Logo, a otimização de E^{ot} corresponde a minimização de cada termo em relação a cada neurônio v_{xi} . Como se trata de um problema de minimização, o indivíduo mais adaptado será o que possuir menor valor de aptidão;

- Método de Seleção: Foi aplicado o método do Torneio.
- Operadores Genéticos: O cruzamento foi realizado aplicando o operador BLX- α , com $\alpha = 0,5$ e taxa de cruzamento igual a 75% (valor usual que se mostrou adequado para os experimentos). A mutação utilizada foi a Uniforme, com $x'_k \in [0,1]$ e taxa de mutação de 2%, estando dentro da faixa recomendada na literatura.
- Critério de Parada: Foi utilizado o desvio padrão entre os valores de aptidão dos indivíduos da população. Quando o desvio padrão atinge uma precisão requerida, definida em 10^{-4} , o algoritmo genético é finalizado.

5.3 O Problema das N -Rainhas

O problema das N -rainhas foi estudado por C. F. Gauss em 1850 e é enunciado a seguir (Wirth, 1976). N rainhas devem ser colocadas em um tabuleiro de xadrez de dimensão

$N \times N$, de modo que nenhuma delas ataque ou seja atacada por qualquer outra. Um modo, pouco viável de se resolver este problema é através da busca exaustiva, isto é, enumeram-se todas as combinações possíveis e utiliza-se um método de tentativa e erro. O número de combinações possíveis ($NCP^{Rainhas}$) é dado por:

$$NCP^{Rainhas} = \frac{N^2 !}{N!(N^2 - N)!} \quad (5.31)$$

e para $N=8$, tem-se $4,426 \times 10^9$ combinações possíveis. A pesquisa de todas estas combinações torna o algoritmo de busca exaustiva pouco atrativo, pois um enorme esforço computacional é exigido.

A partir das regras do xadrez, sabe-se que a rainha pode atacar peças que se encontrem na mesma linha, coluna ou diagonal do tabuleiro, logo, se pode deduzir inicialmente que cada linha e cada coluna devem conter apenas uma rainha. Portanto, dado que cada casa (quadrado) do tabuleiro pode ser representada adequadamente por um neurônio, as restrições estruturais para o problema, minimizadas pelo termo de energia E^{conf} , e que são mapeadas pelo subespaço-válido são dadas por:

- Cada linha deve conter somente uma rainha;
- Cada coluna deve conter somente uma rainha.

A minimização do termo de energia E^{ot} corresponderá, portanto, à minimização do número de rainhas em cada uma das diagonais do tabuleiro de xadrez. Deve-se notar que existem várias soluções válidas (igualmente ótimas) para o problema específico das N -Rainhas, e a determinação de uma destas soluções depende exclusivamente dos valores atribuídos para v na inicialização da rede.

O problema das N -Rainhas não é um problema solucionável por programação dinâmica, por não possuir a característica de subestrutura ótima, ou seja, a solução para o problema não contém em seu interior soluções ótimas para subproblemas independentes. Não existe subestrutura ótima porque cada subproblema, ou seja, cada colocação de uma rainha no tabuleiro depende das colocações anteriores de rainhas.

Modelamento do Problema das N -Rainhas através da Rede de Hopfield Genética

A implementação do problema das N -Rainhas, como definido anteriormente, pode ser feita a partir das definições apresentadas na Seção 5.1. Define-se o vetor p como sendo o vetor que contém os índices das casas onde serão colocadas as rainhas nas diversas linhas do tabuleiro. Portanto, o valor do elemento p_i representa que casa estará ativada (ocupada por uma rainha) na i -ésima linha do tabuleiro, ou seja:

$$p_i \in \{1, \dots, N\}, \text{ onde } i \in \{1, \dots, N\}$$

sendo N o número de rainhas para o problema. O vetor $v(p)$ e $V(p)$ são dados por:

$$v(p) = \begin{bmatrix} \delta(p_1) \\ \delta(p_2) \\ \delta(p_3) \\ \vdots \\ \delta(p_n) \end{bmatrix}_{N \times N} \quad V(p) = \begin{bmatrix} \delta(p_1)^T \\ \delta(p_2)^T \\ \delta(p_3)^T \\ \vdots \\ \delta(p_n)^T \end{bmatrix}_{N \times N}$$

onde $\delta(k) \in \mathbb{R}^N$ é um vetor coluna correspondente a k -ésima coluna de δ .

	♠		
			♠
♠			
		♠	

Figura 5.5: O problema das N -rainhas, para $N=4$.

Como ilustração para um problema das N -rainhas com $N=4$, cuja representação gráfica de uma das soluções é mostrada na Figura 5.5, os vetores p e $v(p)$, e a matriz $V(p)$ são dados por:

$$p^T = [2 \ 4 \ 1 \ 3]$$

$$v(p)^T = [0 \ 1 \ 0 \ 0 \quad 0 \ 0 \ 0 \ 1 \quad 1 \ 0 \ 0 \ 0 \quad 0 \ 0 \ 1 \ 0]$$

$$V(p) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Deduz-se a seguir, as equações para os parâmetros internos (T^{val} e s) da rede de Hopfield Genética para o problema das N -rainhas.

Dedução das Equações Para os Parâmetros T^{val} e s do Termo E^{conf}

As equações de T^{val} e s do subespaço-válido para o problema das N -rainhas são desenvolvidas utilizando os conceitos descritos na Seção 5.1. As restrições estruturais para o problema das N -rainhas mostram que apenas um neurônio por linha ou coluna deve estar ativado. Esta restrição significa que apenas um e somente um elemento de cada linha ou coluna da matriz $V(p)$ deve possuir valor igual a 1, ou seja:

$$\begin{aligned} [V(p)]_{ij} &\in \{1, 0\} \\ \sum_{i=1}^N [V(p)]_{ij} &= \sum_{j=1}^N [V(p)]_{ij} = 1 \quad (i, j \in \{1, \dots, N\}) \end{aligned}$$

Para este problema, dado que $v = \text{vec}(V^T)$ é uma solução válida, um subespaço para o mesmo pode ser representado da seguinte forma:

$$S = V = \frac{1}{N} o^N \cdot o^{N^T} \quad (5.32)$$

Neste caso, para que a solução seja válida durante toda a evolução da rede, a matriz T^{val} do subespaço-válido deve garantir que a soma de cada linha ou coluna da matriz V seja igual a zero. Esta operação é efetuada, utilizando-se as propriedades da matriz R^N dada em (5.17), isto é:

$$R^N \cdot V \cdot R^N = T^{val} \cdot V \quad (5.33)$$

Substituindo-se (5.32) e (5.33) na equação do subespaço-válido, tem-se:

$$V = R^N \cdot V \cdot R^N + \frac{1}{N} o^N \cdot o^{N^T} \quad (5.34)$$

Aplicando-se o operador $\text{vec}(\cdot)$ dado em (5.14) na equação (5.34), obtém-se:

$$\text{vec}(V) = \text{vec}(R^N \cdot V \cdot R^N) + \frac{1}{N} \text{vec}(o^N \cdot o^{N^T})$$

$$\text{vec}(V) = (R^N \otimes R^N) \cdot \text{vec}(V) + \frac{1}{N} (o^N \otimes o^N) \quad (5.35)$$

Substituindo-se $\text{vec}(V)$ por v na equação (5.35), tem-se:

$$v = (R^N \otimes R^N) \cdot v + \frac{1}{N} (o^N \otimes o^N) \quad (5.36)$$

Por identidade com a equação do subespaço-válido ($v = T^{val} \cdot v + s$), resulta que os parâmetros T^{val} e s para o problema das N -rainhas são dados por:

$$T^{val} = (R^N \otimes R^N) \quad (5.37)$$

$$s = \frac{1}{N} (o^N \otimes o^N) \quad (5.38)$$

As equações (5.37) e (5.38) satisfazem as propriedades do subespaço-válido, isto é, $T^{val} \cdot T^{val} = T^{val}$ e $T^{val} \cdot s = 0$.

Definição dos Parâmetros do Algoritmo Genético Para Obtenção do Termo E^{ot}

Para o problema das N -rainhas, considerando $n=3$, o tamanho dos cromossomos é igual a $n \times n$ e cada cromossomo é codificado como um vetor de números reais, no qual a cada n posições do vetor se representam as posições da linha n , como mostra a Figura 5.6.

Figura 5.6: Representação genética para o problema das N-rainhas.

A função de avaliação ou *fitness* é definida com base no termo de energia E^{ot} da rede de Hopfield Genética. Assim, o termo E^{ot} para o problema das N -rainhas é definido por:

$$\begin{aligned}
 E^{ot} = & \underbrace{\frac{1}{2} \sum_{i=2}^N \sum_{j=1}^{i-1} \left[\left(\sum_{\substack{k=i-j+1 \\ k \neq i}}^N v_{k,k-i+j} \right) \cdot v_{ij} \right]}_{1o.Termo} + \underbrace{\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \left[\left(\sum_{\substack{k=1 \\ k \neq i}}^{N+i-j} v_{k,k-i+j} \right) \cdot v_{ij} \right]}_{2o.Termo} + \\
 & \underbrace{\frac{1}{2} \sum_{i=1}^N \sum_{j=N-i+1}^N \left[\left(\sum_{\substack{k=i+j-N \\ k \neq i}}^N v_{k,i+j-k} \right) \cdot v_{ij} \right]}_{3o.Termo} + \underbrace{\frac{1}{2} \sum_{i=1}^{N-1} \sum_{j=1}^{N-i} \left[\left(\sum_{\substack{k=1 \\ k \neq i}}^{i+j-1} v_{k,i+j-k} \right) \cdot v_{ij} \right]}_{4o.Termo}
 \end{aligned} \quad (5.39)$$

Na equação (5.39), o primeiro termo fornece o produto entre cada elemento v_{ij} e todos os outros elementos que estão na diagonal direita inferior, enquanto o segundo termo fornece o produto de cada elemento v_{ij} com todos os outros elementos que estão na diagonal esquerda superior. Por conseguinte, o terceiro termo fornece o produto entre cada elemento v_{ij} e todos os outros elementos que estão na diagonal direita superior, e o quarto termo fornece o produto de cada elemento v_{ij} com todos os outros elementos que estão na diagonal esquerda inferior.

Os demais parâmetros do algoritmo genético, população inicial, tamanho da população, método de seleção, operadores genéticos e critério de parada são os mesmos usados no problema do caminho mínimo, definidos na Seção 5.2.3.

5.4 O Problema do Emparelhamento Bipartido

O problema do emparelhamento bipartido está relacionado intrinsecamente à Teoria de Grafos. Um grafo G é constituído por um par $G = (V, E)$, onde V é um conjunto finito de nodos ou vértices, e o conjunto E tem como elementos os subconjuntos de V de cardinalidade dois, denominado de arcos (Papadimitriou e Steiglitz, 1998). Como exemplo, considere o grafo $G=(V,E)$, onde $V = \{v_1, v_2, v_3, v_4\}$ e $E = \{[v_1, v_2], [v_2, v_3], [v_3, v_4], [v_4, v_1], [v_1, v_3]\}$, ilustrado na Figura 5.7.

Figura 5.7: Exemplo de um grafo.

Segundo Papadimitriou e Steiglitz (1998), um grafo $G=(V,E)$ é denominado bipartido quando o conjunto de vértices V pode ser particionado em dois conjuntos, U e W , de modo que cada arco $\in E$ possui um vértice em U e outro vértice em W , como mostra a Figura 5.8.

A cada arco $[u_i, w_j]$ pode-se associar um valor $P_{ij} \geq 0$, que corresponde ao valor (ou peso) da conexão entre o vértice u_i e w_j . Este valor pode ser interpretado como o custo de sair do vértice u_i e chegar até o w_j .

Figura 5.8: Exemplo de um grafo bipartido.

O problema do emparelhamento bipartido em um grafo consiste em determinar um subconjunto M de arcos, de modo que cada vértice de U e W esteja associado somente a um arco. Nesta tese são considerados apenas os grafos bipartidos completos, ou seja, quando os conjuntos U e W possuem o mesmo número de vértices (Papadimitriou e Steiglitz, 1998).

O problema do emparelhamento bipartido é mínimo se a soma dos valores P_{ij} associado a cada arco de M for o menor possível (Papadimitriou e Steiglitz, 1998). Como ilustração, a Figura 5.9 mostra um grafo bipartido com quatro vértices. Os conjuntos V , E , U e W , e a matriz P (escolhida aleatoriamente) são dados por:

$$V = \{u_1, u_2, w_1, w_2\}$$

$$E = \{[u_1, w_1], [u_1, w_2], [u_2, w_1], [u_2, w_2]\}$$

$$U = \{u_1, u_2\}$$

$$W = \{w_1, w_2\}$$

$$P = \begin{bmatrix} 2.9 & 0.8 \\ 1.3 & 3.5 \end{bmatrix}$$

Neste caso, o emparelhamento bipartido mínimo será escolhido entre o subconjunto $M_1 = \{[u_1, w_1], [u_2, w_2]\}$ e $M_2 = \{[u_1, w_2], [u_2, w_1]\}$. Como a soma dos arcos do subconjunto M_2 é menor que a soma dos arcos do subconjunto definido por M_1 , então os elementos (arcos) do subconjunto M_2 corresponderão ao emparelhamento bipartido mínimo, ou seja, $M = M_2$.

Figura 5.9: Grafo bipartido.

Modelamento do Problema do Emparelhamento Bipartido Através da Rede de Hopfield Genética

A implementação deste problema também pode ser feita a partir das definições apresentadas na Seção 5.1. Sabe-se que a cada elemento pertencente tanto ao conjunto U como ao conjunto W se deve associar um único arco. Portanto, as restrições estruturais para o problema são semelhantes àsquelas do problema das N -rainhas, isto é, cada vértice do conjunto U e W deve ser associado a somente um arco. Redefine-se então o vetor p como sendo o vetor cujos elementos contêm os índices dos vértices pertencentes ao conjunto W . Assim, o termo p_i representa o arco que une o i -ésimo vértice pertencente a U ao respectivo vértice (valor do elemento p_i) pertencente a W , ou seja:

$$p_i \in \{1, \dots, N\}, \text{ onde } i \in \{1, \dots, N\}$$

onde N é o número de elementos do conjunto U ou W . Para o problema do emparelhamento bipartido ilustrado na Figura 5.9 com $N=2$, os vetores p e $v(p)$, e a matriz $V(p)$ são dados por:

$$p^T = [2 \quad 1]$$

$$v(p)^T = \begin{bmatrix} 0 & 1 & 1 & 0 \\ \delta(p_1)^T & \delta(p_2)^T \end{bmatrix}$$

$$V(p) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Deduz-se a seguir, as equações para os parâmetros internos (T^{val} e s) da rede de Hopfield Genética para o problema do emparelhamento bipartido.

Dedução das Equações Para os Parâmetros T^{val} e s do Termo E^{conf}

As equações de T^{val} e s do subespaço-válido para o problema do emparelhamento bipartido são idênticas àsquelas desenvolvidas para o problema das N -rainhas. Assim, as equações de T^{val} e s do subespaço-válido são dadas por:

$$T^{val} = (R^N \otimes R^N) \quad (5.40)$$

$$s = \frac{1}{N} (o^N \otimes o^N) \quad (5.41)$$

Portanto, as equações (5.40) e (5.41) garantem que para cada vértice, $u_j \in U$ e $w_i \in W$, seja associado a um único arco.

Definição dos Parâmetros de Algoritmo Genético Para Obtenção do Termo E^{ot}

Para o problema do emparelhamento bipartido ilustrado na Figura 5.10, o qual é constituído por seis vértices, divididos em dois conjuntos U e W de três vértices cada, sendo u_i (com $i=1..3$) os elementos do conjunto U , e w_j (com $j=1..3$) os elementos do conjunto W . O tamanho dos cromossomos é igual a $i*j$ e cada cromossomo é codificado como um vetor de números reais, no qual a cada $|w|$ posições do vetor se representam as conexões do vértice u_i com cada vértice w_j .

Figura 5.10: Representação genética para o problema do emparelhamento bipartido.

A função de avaliação ou *fitness* também é definida com base no termo de energia E^{ot} da rede de Hopfield Genética. Este termo de energia, para o problema do emparelhamento bipartido, é projetado com o objetivo de que a solução ótima corresponda à menor soma dos valores P_{ij} associados a cada arco pertencente a M . Assim, a soma total (st) referente ao vetor ordenado p é dada por:

$$\begin{aligned}
 E^{ot} = st(p) &= \text{traço}(V(p)^T \cdot P) \\
 &= \text{vec}(V(p)^T) \cdot \text{vec}(P)
 \end{aligned}$$

$$= v(p)^T \cdot \text{vec}(P) \quad (5.42)$$

Os demais parâmetros do algoritmo genético, ou seja, população inicial, tamanho da população, método de seleção, operadores genéticos e critério de parada são os mesmos usados no problema do caminho mínimo, definidos na Seção 5.2.3.

5.5 Considerações Finais

O modelamento dos problemas do caminho mínimo, das N -rainhas e do emparelhamento bipartido utilizando a Rede de Hopfield Genética, assim como as notações, definições e propriedades necessárias para realizar o modelamento foram apresentadas neste capítulo. A partir da descrição do modelamento dos problemas, nota-se que a abordagem neuro-genética proposta possui uma importante vantagem em relação às demais abordagens encontradas na literatura (ver Seções 5.2.1 e 5.2.2), que é a aplicabilidade, isto é, a forma de modelamento dos problemas de otimização, independente da natureza de cada tipo de problema, é feita sempre utilizando a mesma metodologia.

As funções objetivo de cada problema foram definidas e as respectivas representações genéticas (codificação dos cromossomos) utilizadas no algoritmo genético, responsável pela otimização da função objetivo dos problemas de otimização, foram detalhadas.

Outro aspecto relevante da Rede de Hopfield Genética, quando utilizada na resolução de problemas de programação dinâmica, exemplificado nesta tese pelo problema do caminho mínimo, deve-se à otimização da função objetivo de forma paralela. Diferentemente da abordagem tradicional da programação dinâmica, que resolve um problema dividindo-o em subproblemas menores, de forma sequencial (ver exemplo na Seção 2.2.1.1), a RHG se beneficia do paralelismo presente nos algoritmos genéticos, pois o problema original completo é codificado nos cromossomos do algoritmo genético, onde cada cromossomo representa uma possível solução do espaço de busca do problema.

No próximo capítulo, os experimentos e resultados da aplicação da Rede de Hopfield Genética na resolução de problemas de conexão em otimização combinatória são apresentados.

Capítulo 6

Experimentos e Resultados

Neste capítulo são apresentados os experimentos e resultados obtidos a fim de demonstrar a efetividade da Rede de Hopfield Genética quando utilizada na resolução de problemas de conexão em otimização combinatória, mais precisamente para os problemas do caminho mínimo, das N -rainhas e do emparelhamento bipartido.

Todos os experimentos foram realizados utilizando um processador Intel Core 2 Duo de 1.8GHz, memória RAM de 2 Gigabytes e o ambiente de programação MatLab. Estas informações são importantes para que se possa comparar o tempo de convergência da Rede de Hopfield Genética com outras abordagens.

6.1 Experimentos: Problema do Caminho Mínimo

Para mostrar a aplicabilidade e eficiência da abordagem neuro-genética, diferentes configurações para o problema do caminho mínimo foram simuladas e comparadas com as abordagens apresentadas por Chiu et al. (1991), Silva et al. (2007) e por um algoritmo específico para o problema do caminho mínimo, que calcula a distância entre todos os pares de vértices de um grafo qualquer. Este algoritmo foi implementado utilizando a técnica da programação dinâmica, sendo que o seu pseudocódigo está disponível em Toscani e Veloso (2005). Além disso, comparações com o algoritmo de Dijkstra (Dijkstra, 1959) também são apresentadas. Este algoritmo também é específico para solucionar o problema do caminho mínimo em grafos direcionados ou não direcionados com arestas de peso não negativo

(Ziviani, 2005). O algoritmo de programação dinâmica e o algoritmo de Dijkstra são exatos, isto é, sempre encontram a solução ótima.

Para o problema do caminho mínimo (PCM) é importante destacar o fato de que a maioria dos trabalhos encontrados na literatura modela o PCM como uma busca em um grafo qualquer. Por outro lado, a abordagem desenvolvida nesta tese resolve este problema modelando-o como um problema de Programação Dinâmica; sendo assim, a busca da solução é realizada por meio de um grafo em camadas, conforme ilustrado na Figura 2.1. Dos trabalhos encontrados na literatura que resolvem o PCM, apenas os desenvolvidos por Chiu et al. (1991) e Silva et al. (2007) modelam este problema através de uma busca em um grafo em camadas.

Para uma comparação mais sistemática, utiliza-se a mesma metodologia de simulação usada em Chiu et al. (1991). Nesta metodologia, o número de estágios e o número de estados foram incrementados passo a passo. Estes números foram obtidos a partir de valores pertencentes ao conjunto inteiro definido por $\{2, 4, 8, 16, 32, 64\}$. Os pesos das conexões ligando os nós do grafo foram aleatoriamente selecionados a partir de um conjunto de números inteiros definido por $\{1, 3, 5, 7, 9\}$. Para as instâncias com o número de estágios e estados menor que 32, foram executadas 20 simulações com condições iniciais aleatórias. Para os casos com o número de estágios e estados maior ou igual a 32, foram executadas 10 simulações, também com condições iniciais aleatórias. A Tabela 6.1 ilustra os resultados obtidos. Nesta tabela, as colunas n e m descrevem o número de estágios e o número de estados em cada estágio, respectivamente. As colunas D^{NGA} , D^{MHA} , D^{DPN} , D^{DJK} e D^{PD} fornecem os resultados do comprimento médio normalizado computados pela abordagem neuro-genética proposta nesta tese, pelas abordagens de Silva et al. (2007) e Chiu et al. (1991), e pelos algoritmos de Dijkstra e Programação Dinâmica, respectivamente. O comprimento médio normalizado para cada instância é dado por:

$$D = \frac{S_c}{n_s(n+1)} \quad (6.1)$$

onde S_c é a soma dos caminhos selecionados após a convergência da rede, n_s é o número de simulações, e n é o número de estágios.

Tabela 6.1: Resultados do comprimento médio normalizado para o PCM.

Estágios (n)	Estados (m)	D^{NGA}	D^{MHA}	D^{DPN}	D^{DJK}	D^{PD}
2	2	3,67	3,67	3,25	3,67	3,67
2	4	1,67	1,67	indisponível	1,67	1,67
2	8	2,33	2,33	indisponível	2,33	2,33
2	16	1,67	1,67	1,53	1,67	1,67
2	32	1,00	1,00	indisponível	1,00	1,00
2	64	1,00	1,00	indisponível	1,00	1,00
4	2	3,40	3,40	indisponível	3,40	3,40
4	4	1,80	1,80	3,12	1,80	1,80
4	8	1,80	1,80	indisponível	1,80	1,80
4	16	1,40	1,40	1,60	1,40	1,40
4	32	1,00	1,00	indisponível	1,00	1,00
4	64	1,00	1,00	indisponível	1,00	1,00
8	2	2,78	2,78	indisponível	2,78	2,78
8	4	2,78	2,78	indisponível	2,78	2,78
8	8	1,44	1,44	2,00	1,44	1,44
8	16	1,00	1,00	1,76	1,00	1,00
8	32	1,00	1,00	indisponível	1,00	1,00
8	64	1,00	1,00	indisponível	1,00	1,00
16	2	3,47	3,47	3,21	3,47	3,47
16	4	2,18	2,18	2,98	2,18	2,18
16	8	1,29	1,35	1,85	1,24	1,24
16	16	1,07	1,00	1,85	1,00	1,00
16	32	1,04	1,00	1,79	1,00	1,00
16	64	1,02	1,00	indisponível	1,00	1,00
32	2	3,55	3,55	indisponível	3,55	3,55
32	4	2,26	2,15	indisponível	2,15	2,15
32	8	1,56	1,42	indisponível	1,36	1,36
32	16	1,19	1,06	indisponível	1,00	1,00
32	32	1,09	1,00	1,61	1,00	1,00
32	64	1,15	1,01	indisponível	1,00	1,00
64	2	3,37	3,37	indisponível	3,37	3,37
64	4	2,06	1,95	indisponível	1,89	1,89
64	8	1,42	1,38	indisponível	1,18	1,18
64	16	1,39	1,21	indisponível	1,03	1,03
64	32	1,10	1,06	indisponível	1,00	1,00
64	64	1,27	1,00	1,39	1,00	1,00

Como dito anteriormente, os algoritmos de Dijkstra e de Programação Dinâmica são exatos, ou seja, sempre encontram a solução ótima do problema do caminho mínimo. Sendo assim, os resultados de comprimento médio normalizado destes algoritmos servem como referência para averiguar o quão próximo do resultado ótimo estão os resultados das demais abordagens. Em outras palavras, isto significa que os resultados obtidos pelas abordagens devem ser maiores ou igual ao resultado computado pelos algoritmos de Dijkstra e Programação Dinâmica.

Os resultados das simulações mostram que a abordagem neuro-genética é uma alternativa eficiente para a resolução do problema do caminho mínimo, pois todos os resultados obtidos pela abordagem neuro-genética são soluções próximas das ótimas ou ótimas. A qualidade dos resultados reduz a partir do aumento da complexidade do problema, ou seja, com o aumento do número de estágios e estados considerados.

Conforme mencionado, para a realização dos testes, os pesos das conexões foram gerados aleatoriamente. Porém, para cada instância do problema, uma mesma configuração dos pesos entre os nós do grafo foi utilizada para testar todas as abordagens, com o intuito de manter condições iguais entre elas. Um fato importante a ressaltar é em relação a alguns resultados apresentados por Chiu et al. (1991), os quais são menores que os dos algoritmos de Dijkstra e Programação Dinâmica. Isto significa dizer que Chiu et al. (1991) encontraram resultados melhores que aqueles ótimos, os quais mostram que tais soluções são infactíveis, pois provavelmente as restrições estruturais foram violadas. Esta discrepância ocorre porque em Chiu et al. (1991) a configuração dos pesos utilizada não é apresentada, o que impossibilitou a realização de testes nas mesmas condições.

A abordagem proposta por Silva et al. (2007) obteve resultados um pouco melhores do que os obtidos pela abordagem neuro-genética. Apesar disso, esta abordagem apresenta um alto tempo de processamento para os problemas com um alto número de estágios e estados, quando comparado com a abordagem neuro-genética.

A Tabela 6.2 ilustra os tempos de processamento (em segundos) obtidos pela abordagem neuro-genética (T^{NGA}), pela abordagem de Silva et al. (2007) (T^{MHA}), pelo algoritmo de Dijkstra (T^{DJK}) e pelo algoritmo de Programação Dinâmica (T^{PD}).

Tabela 6.2: Resultados dos tempos de processamento (em segundos).

Estágios (n)	Estados (m)	T^{NGA}	T^{MHA}	T^{DJK}	T^{PD}
2	2	0,02	0,00	0,05	0,03
2	4	0,04	0,00	0,01	0,00
2	8	0,08	0,02	0,02	0,04
2	16	0,07	0,10	0,02	0,12
2	32	0,19	0,31	0,03	0,49
2	64	0,41	17,13	0,04	2,14
4	2	0,03	0,00	0,01	0,01
4	4	0,08	0,00	0,06	0,06
4	8	0,11	0,03	0,02	0,13
4	16	0,43	0,56	0,03	0,49
4	32	1,67	11,70	0,05	2,14
4	64	6,22	159,34	0,08	10,40
8	2	0,12	0,00	0,02	0,04
8	4	0,22	0,17	0,02	0,13
8	8	1,47	3,96	0,07	0,52
8	16	4,82	5,51	0,05	2,15
8	32	15,67	49,28	0,08	10,39
8	64	150,73	555,13	0,18	57,41
16	2	0,66	0,01	0,02	0,13
16	4	5,38	0,30	0,03	0,49
16	8	19,93	4,89	0,05	2,14
16	16	62,71	115,82	0,08	10,45
16	32	445,87	1183,80	0,18	57,41
16	64	2024,22	2673,45	0,42	380,72
32	2	1,23	0,02	0,03	0,49
32	4	5,82	4,47	0,10	1,93
32	8	61,06	110,46	0,14	8,50
32	16	777,08	1038,83	0,23	57,98
32	32	3490,37	5092,48	0,42	391,83
32	64	10857,00	15680,90	1,07	1467,08
64	2	9,77	4,14	0,05	1,95
64	4	79,73	24,45	0,08	8,72
64	8	560,79	569,88	0,17	41,43
64	16	2938,34	3852,35	0,42	256,78
64	32	10554,10	11522,90	1,04	1909,16
64	64	21870,50	53274,90	7,09	15674,56

Em todos os problemas com número de estados menor que 8, independente do número de estágios, a abordagem de Silva et al. (2007) teve melhor desempenho. Vale destacar que estes problemas são considerados de baixa complexidade, em função de seu pequeno número de estados em cada estágio. Nos problemas com número de estados igual a 8, independente do número de estágios, os resultados da abordagem neuro-genética foram piores que a abordagem de Silva et al. (2007) e melhores para outros.

Para problemas com número de estados maior ou igual a 16, também independente do número de estágios, a abordagem neuro-genética apresentou tempo de processamento consideravelmente inferior ao obtido por Silva et al. (2007), o que demonstra a eficiência, tanto em precisão quanto em tempo de processamento, da abordagem neuro-genética em resolver problemas com alta dimensionalidade e, portanto, mais complexos.

Portanto, a partir dos resultados obtidos, pode-se concluir que para definir qual sistema inteligente é o mais apropriado para uma determinada situação, torna-se necessário analisar os requisitos da aplicação, ou seja, se o mais importante é a precisão, então a abordagem de Silva et al. (2007) será a mais adequada. No entanto, se resultados rápidos são necessários, considerando instâncias com maior número de estágios e estados, a abordagem neuro-genética deve ser prioritariamente escolhida.

Os tempos de processamento exibidos na Tabela 6.2 mostram que os algoritmos exatos, tanto o algoritmo de Dijkstra quanto o algoritmo de Programação Dinâmica, apresentam melhor desempenho, com grande superioridade em relação à abordagem neuro-genética e à abordagem de Silva et al. (2007). Entretanto, os algoritmos exatos são bastante específicos. O algoritmo de Dijkstra resolve apenas problemas de caminho mínimo em grafos e o algoritmo de Programação Dinâmica soluciona apenas problemas que podem ser modelados segundo as características apresentadas na Seção 2.2.1.2.

A Figura 6.1 mostra, em 3 dimensões, a evolução da matriz V em uma simulação da Rede de Hopfield Genética para um problema com 8 estágios e 16 estados por estágio. Nota-se que a rede começa a convergir para a solução final, $p^T = [10, 1, 4, 14, 13, 11, 8, 12]$, a partir da décima quinta iteração.

Figura 6.1: Evolução da matriz V para o PCM ($n=8$ e $m=16$).

A Figura 6.2 ilustra o comportamento da função objetivo do problema $E^{ot}(v)$, representada por $E(v)$, em função do número de iterações. Após a convergência da rede, o valor da função objetivo para o problema ($n=8$ e $m=16$) é igual a 10,4482.

Figura 6.2: Comportamento da função objetivo para o PCM ($n=8$ e $m=16$).

Em Pires et al. (2008a; b; c) podem ser encontrados os resultados obtidos da aplicação da abordagem neuro-genética para o problema do caminho mínimo.

6.2 Experimentos: Problema das N -Rainhas

A rede de Hopfield Genética foi aplicada na solução do problema das N -rainhas com $N=5$ e com $N=8$. Para $N=5$, a rede converge em cerca de 26 iterações. A Figura 6.3 mostra duas soluções diferentes obtidas pela abordagem neuro-genética. No problema das N -rainhas com $N=8$ (dimensão de um tabuleiro oficial), a abordagem neuro-genética proposta sempre converge para uma solução válida (ponto de equilíbrio) em cerca de 90 iterações em cada simulação. A Tabela 6.4, mostra 14 soluções diferentes obtidas pela abordagem proposta. O vetor de saída da rede, v , foi inicializado, em todas as simulações (independente do valor atribuído a N), com valores aleatórios entre zero e um.

Tabela 6.3: Soluções para o problema das N -rainhas ($N=5$).

p	p_1	p_2	p_3	p_4	p_5
1	2	5	3	1	4
2	4	1	3	5	2

Tabela 6.4: Soluções para o problema das N-rainhas (N=8).

p	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
1	3	6	4	1	8	5	7	2
2	7	4	2	5	8	1	3	6
3	4	2	7	3	6	8	1	5
4	5	8	4	1	3	6	2	7
5	6	3	7	2	8	5	1	4
6	6	3	1	7	5	8	2	4
7	5	3	1	7	2	8	6	4
8	7	2	4	1	8	5	3	6
9	3	6	8	2	4	1	7	5
10	4	7	5	3	1	6	8	2
11	2	6	8	3	1	4	7	5
12	6	4	7	1	8	2	5	3
13	3	5	2	8	1	7	4	6
14	3	5	8	4	1	7	2	6

A Figura 6.3 mostra, em 3 dimensões, a evolução da matriz V (representação matricial do vetor de saída da rede) em uma simulação da Rede de Hopfield Genética para um problema das N -rainhas com $N=5$. Nota-se que a rede começa a convergir para a solução final, $p^T = [4, 1, 3, 5, 2]$, a partir da décima iteração.

Figura 6.3: Evolução da matriz V para o problema das N -rainhas com $N=5$.

A Figura 6.4 ilustra a evolução da matriz V em uma simulação da Rede de Hopfield Genética para um problema das N -rainhas com $N=8$. Observa-se que a rede, a partir da trigésima segunda iteração, começa a convergir para a solução final representada pelo vetor $p^T=[3,5,8,4,1,7,2,6]$.

Figura 6.4: Evolução da matriz V para o problema das N -rainhas com $N=8$.

6.3 Experimentos: Problema do Emparelhamento Bipartido

A abordagem neuro-genética foi utilizada na solução do problema do emparelhamento bipartido com $N=5$ proposto em Papadimitriou e Steiglitz (1998). A matriz P fornecida para o problema, e cujos elementos representam o valor das conexões entre os arcos do grafo bipartido, é dada por:

$$P = \begin{bmatrix} 7 & 2 & 1 & 9 & 4 \\ 9 & 6 & 9 & 5 & 5 \\ 3 & 8 & 3 & 1 & 8 \\ 7 & 9 & 4 & 2 & 2 \\ 8 & 4 & 7 & 4 & 8 \end{bmatrix}$$

Para as simulações realizadas, a rede converge para a solução ótima, em cerca de 34 iterações. Os valores de p , $v(p)$ e $V(p)$, representando a solução ótima obtida após a convergência da rede, são dados por:

$$p^T = [3 \quad 5 \quad 1 \quad 4 \quad 2]$$

$$v(p) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{matrix} \delta(p_1)^T & \delta(p_2)^T & \delta(p_3)^T & \delta(p_4)^T & \delta(p_5)^T \end{matrix}$$

$$V(p) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

A solução ótima obtida (soma dos arcos), com $E^{ot} = 15$, é a mesma encontrada pelo método ‘*Hungarian*’, que é um método baseado no algoritmo primal-dual utilizado em programação linear (Papadimitriou e Steiglitz, 1998). As funções de ativação de todos os neurônios da rede são limitadas entre zero e um.

A Figura 6.5 mostra a saída da rede durante os estágios de convergência. Nota-se nesta figura que a partir da décima segunda iteração, a rede já tende às prováveis soluções finais para o problema do emparelhamento bipartido.

Figura 6.5: Evolução da matriz V para o problema do emparelhamento bipartido ($N=5$).

A Figura 6.6 ilustra, em 3 dimensões, a evolução da matriz V em uma simulação da Rede de Hopfield Genética para um problema do emparelhamento bipartido com $N=8$. Nota-se que a rede começa a convergir para a solução final, $p^T = [7, 4, 3, 1, 6, 2, 5, 8]$, a partir da vigésima nona iteração.

Figura 6.6: Evolução da matriz V para o problema do emparelhamento bipartido ($N=8$).

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

Este trabalho apresentou o desenvolvimento de uma abordagem híbrida, a qual é constituída por uma rede neural artificial e por um algoritmo genético, para a solução de problemas de Conexão em Otimização Combinatória. Mais precisamente, foi proposta uma alteração na Rede de Hopfield convencional, resultando em uma nova rede neural denominada Rede de Hopfield Genética. Nesta arquitetura híbrida, a rede neural possui a função de satisfazer as restrições presentes nos problemas de otimização e o algoritmo genético é responsável por minimizar a função objetivo do problema.

As configurações utilizadas nos experimentos do problema do caminho mínimo visando propósitos de validação, modelados e solucionados por meio da Rede de Hopfield Genética, foram comparados com dois métodos inteligentes {(Chiu et al., 1991) e (Silva et al., 2007)} e também com dois algoritmos exatos {algoritmo de Dijkstra e Programação Dinâmica}. As diversas simulações realizadas confirmam que a abordagem proposta é uma alternativa viável para solucionar problemas de conexão em otimização combinatória, considerando tanto a precisão das soluções obtidas como também o tempo de convergência despendido para o alcance dos pontos de equilíbrio que representam tais soluções.

Com o objetivo de mostrar a aplicabilidade da abordagem neuro-genética na resolução de outros problemas de otimização combinatória, utilizando-se sempre a mesma metodologia

de modelamento dos problemas, experimentos adicionais foram realizados para os problemas das N -rainhas e do emparelhamento bipartido.

Com base em tais experimentos, tornou-se possível verificar que a utilização da abordagem neuro-genética proporciona vantagens quando comparada com os métodos encontrados na literatura. Dentre estes benefícios, destacam-se:

i) Modelamento de problemas de otimização de naturezas diferentes utilizando uma mesma metodologia. Além disso, problemas que podem ser solucionados através da técnica de programação dinâmica, como por exemplo, o problema do caminho mínimo, também são modelados usando a mesma metodologia empregada nos demais problemas de otimização.

ii) Viabilização do tratamento das restrições envolvidas com os problemas, sem interferir no processo de otimização da função objetivo do problema. Como foi comentado na Seção 4.1, o tratamento das restrições impostas pelo problema utilizando o modelo convencional da rede de Hopfield, necessitava de um grande esforço computacional, o que poderia prejudicar no processo de convergência da rede e na qualidade das soluções obtidas. Para contornar este problema, a Rede de Hopfield Genética proposta utilizou a técnica do subespaço-válido. Sendo assim, a resolução de um problema de otimização por meio da RHG é feita em dois estágios distintos: 1) confinamento das restrições impostas pelo problema dentro de um subespaço-válido de soluções e 2) otimização da função objetivo.

iii) Não necessidade de definição de parâmetros de controle ou ponderação para sua inicialização. Este também é um diferencial da RHG em relação aos demais trabalhos encontrados na literatura, pois a maioria deles utiliza vários parâmetros para auxiliar o sistema na busca de soluções. Porém, os valores destes parâmetros são definidos por tentativa e erro, e para cada tipo de problema, novos valores devem ser definidos. Em outras palavras, caso os valores escolhidos não sejam adequados, o sistema pode torna-se inviável computacionalmente (alto tempo de processamento) e/ou obter soluções ineficazes (violação das restrições impostas pelos problemas).

iv) Redução do custo computacional para problemas com alta dimensionalidade, quando comparado com a abordagem proposta em Silva et al. (2007). Os algoritmos exatos apresentaram um desempenho muito superior em relação à RHG, porém, estes algoritmos são específicos para o problema do caminho mínimo. Portanto, com a abordagem neuro-genética proposta se ganha em aplicabilidade e flexibilidade e, por outro lado, perde-se em tempo de processamento.

iv) Uma das razões para a ampla utilização da rede de Hopfield na resolução de problemas de otimização é a sua facilidade de implementação em *hardware*, conforme apresentado na Seção 3.1. Esta é uma possível alternativa para o problema do tempo de processamento apresentado pela RHG, pois a implementação em *hardware* é muito mais rápida de que em *software* (Wen et al., 2008).

Tais benefícios comprovam que o desenvolvimento da abordagem neuro-genética proposta nesta tese é uma ferramenta promissora e vantajosa, capaz de solucionar problemas de conexão em otimização combinatória, além de fornecer um método inovador para solução deste tipo de problema.

7.2 Trabalhos Futuros

Como foi descrito na Seção 5.2, a maioria dos trabalhos encontrados na literatura modela o problema do caminho mínimo como uma busca em um grafo qualquer e, a abordagem proposta nesta tese, realiza a busca em um grafo em camadas. Sendo assim, pretende-se estender a abordagem neuro-genética na resolução do PCM, considerando não apenas a disposição dos nós do grafo em camadas, mas também em qualquer disposição.

Outra proposição de trabalho futuro é aplicar a abordagem neuro-genética para a solução de outros tipos de problemas de otimização combinatorial, como por exemplo, o problema do caixeiro viajante.

Por fim, pretende-se também utilizar a RHG em aplicações a serem desenvolvidas no Laboratório de Sistemas Inteligentes e Cognitivos (LASIC), junto ao Grupo de Pesquisa Básica e Avançada em Sistemas Inteligentes (GPBASI), do curso de Engenharia da Computação da Universidade Estadual de Feira de Santana, onde o autor é professor e pesquisador.

Referências Bibliográficas

- AHN, C. W. e RAMAKRISHNA, R. S. (2002). A genetic algorithm for shortest path routing problem and the sizing of populations. **IEEE Transactions on Evolutionary Computation**, vol. 6, pp. 566-579.
- AHN, C. W., RAMAKRISHNA, R. S., KANG, C. G. e CHOI, I. C. (2001) Shortest path routing algorithm using Hopfield neural network. **Electronics Letters**, vol. 37, pp. 1176-1178, DOI: 10.1049/el:20010800.
- AIYER, S. V. B., NIRANJA, M. e FALLSIDE, F. (1990). A theoretical investigation into the performance of the Hopfield model. **IEEE Transactions on Neural Networks**, vol. 1, n. 2, pp. 204-215.
- ALI, M. K. M. e KAMOUN, F. (1993). Neural networks for shortest path computation and routing in computer networks. **IEEE Transactions on Neural Networks**, vol. 4, pp. 941-953.
- ANTONIO, J. K., HUANG, G. M. e TSAI, W. K. (1992). A fast distributed shortest path algorithm for a class of hierarchically clustered data networks. **IEEE Transactions on Computers**, vol. 41, n. 6, pp. 710-724.
- ARAÚJO, F., RIBEIRO, B. e RODRIGUES, L. (2001). A neural network for shortest path computation. **IEEE Transactions on Neural Networks**, vol. 12, pp. 1067-1073.
- ARROYO, J. E. C. (2002). **Heurísticas e Metaheurísticas para Otimização Combinatória Multiobjetivo**, Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas - SP.

- BACK, T., HOFFMEISTER, F. e SCHWEFEL, H.-P. (1991). A Survey of Evolution Strategies. **International Conference on Genetic Algorithms**, San Diego - CA, pp. 2-9.
- BAZARAA, M. S., SHERALI, H. D. e SHETTY, C. M. (2006). **Nonlinear Programming: Theory and Algorithms**, 3ª ed., John Wiley & Sons.
- BELLMAN, R. (1957). **Dynamic Programming**, Princeton University Press, Princeton.
- BODIN, L., GOLDEN, B. L., ASSAD, A. e BALL, M. (1983). Routing and scheduling of vehicles and crews: the state of the art. **Computers and Operations Research**, vol. 10, n. 2, pp. 63-211.
- BRAGA, A. P., LUDERMIR, T. B. e CARVALHO, A. C. P. L. F. (2007). **Redes Neurais Artificiais - Teoria e Aplicações**, 2ª ed., LTC, Rio de Janeiro.
- BRASSARD, G. e BRADLEY, P. (1996). **Fundamentals of Algorithmics**, Prentice Hall.
- CHIU, C., MAA, C. Y. e SHANBLATT, M. A. (1991). Energy function analysis of dynamic programming neural networks. **IEEE Transactions on Neural Networks**, vol. 2, n. 4, pp. 418-426.
- CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L. e STEIN, C. (2002). **Algoritmos: Teoria e Prática**, Campus, Rio de Janeiro - RJ.
- D' AZZO, J. J. e HOUPIK, C. H. (1975). **Linear Control System Analysis and Design**, McGraw-Hill, New York.
- DE JONG, K. A. (1975). **An Analysis of the Behavior of a Class of Genetic Adaptive Systems**, Ph.D. thesis, University of Michigan, Ann Arbor.
- DELGADO, M. R. (2002). **Projeto Automático de Sistemas Nebulosos: Uma Abordagem Co-Evolutiva**, Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas - SP.

- DIJKSTRA, E. W. (1959). A note on two problems in connexion with graphs. **Numerische Mathematik**, vol. 1, pp. 269-271.
- EPHREMIDES, A. e VERDU, S. (1989). Control and optimization methods in communication network problems. **IEEE Transactions on Automatic Control**, vol. 34, n. 9, pp. 930-942.
- FOGEL, L. J., OWENS, A. J. e WALSH, M. J. (1966). **Artificial Intelligence through Simulated Evolution**, John Wiley.
- FU, Z., KURNIA, A., LIM, A. e RODRIGUES, B. (2003). Shortest path problem with cache dependent path lengths. **Congress on Evolutionary Computation**, vol. 4, pp. 2756-2761.
- GEE, A. H. e PRAGER, R. W. (1991). **Alternative Energy Functions for Optimizing Neural Networks**, Cambridge University Department of Engineering, Technical Report Cued/F-Infeng/TR95.
- GEN, M., CHENG, R. e WANG, D. (1997). Genetic algorithms for solving shortest path problem. **IEEE International Conference on Evolutionary Computation**, pp. 401-406.
- GOLDBARG, M. C. e LUNA, H. P. L. (2005). **Otimização Combinatória e Programação Linear**, 2ª ed., Elsevier, Rio de Janeiro.
- GRAHAM, A. (1981). **Kronecker Products and Matrix Calculus: with Applications**, Ellis Horwood Ltd, Chichester, UK.
- HAYKIN, S. (1999). **Neural Networks - A Comprehensive Foundation**, 2ª ed., Prentice-Hall, New York.
- HAYKIN, S. (2008). **Neural networks and learning machines**, 3ª ed., Prentice Hall.

- HEGDE, S. U., SWEET, J. L. e LEVY, W. B. (1998). Determination of parameters in a Hopfield/Tank computational network. **Proceedings of the International Conference on Neural Network**, vol. 2, pp. 291-298.
- HILLIER, F. S. e LIEBERMAN, G. J. (2001). **Introduction to Operations Research**, McGraw-Hill, Boston, MA.
- HODGKIN, A. L. e HUXLEY, L. F. (1952). A quantitative description of membrane current and its applications to conduction and excitation in nerve. **Journal of Physiology**, vol. 117, pp. 500-544.
- HOLLAND, J. H. (1975). **Adaptation in Natural and Artificial Systems**, MIT Press.
- HOPFIELD, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. **Proceedings of the National Academy of Sciences**, vol. 79, pp. 2554-2558.
- HOPFIELD, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. **Proceedings of the National Academy USA**, vol. 81, pp. 3088-3092.
- HOPFIELD, J. J. e TANK, D. W. (1985). Neural computation of decisions in optimization problems. **Biological Cybernetics**, vol. 52, pp. 141-152.
- HUSH, D. R. e HORNE, B. G. (1993). Progress in supervised neural networks. **IEEE Signal Processing Magazine**, vol. 81, pp. 3088-3092.
- JANG, J.-S. R., SUN, C.-T. e MIZUTANI, E. (1997). **Neuro-Fuzzy and Soft Computing**, Prentice Hall, Englewood Cliffs, NJ.
- JUN, S. e SHIN, K. G. (1991). Shortest path planning in distributed workspace using dominance relation. **IEEE Transactions on Robotics and Automation**, vol. 7, n. 3, pp. 342-350.

- KAMGAR-PARSI, B. e KAMGAR-PARSI, B. (1990). On problem solving with Hopfield neural networks. **Biological Cybernetics**, vol. 62, pp. 415-423.
- LACERDA, E. G. M. (2003). **Model Selection of RBF Networks via Genetic Algorithms**, Tese de Doutorado, Universidade Federal de Pernambuco, Recife - PE.
- LACERDA, E. G. M. e CARVALHO, A. C. P. L. F. (1999). Introdução aos algoritmos genéticos. In **Jornada de Atualização em Informática - Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação**, Rio de Janeiro, vol. 2, pp. 51-126.
- LI, Q., LIU, G., ZHANG, W., ZHAO, C., YIN, Y. e WANG, Z. (2006a). A specific genetic algorithm for optimum path planning in intelligent transportation system. **International Conference on ITS Telecommunications Proceedings**, pp. 140-143.
- LI, Y., HE, R. e GUO, Y. (2006b). Faster genetic algorithm for network paths. **International Symposium on Operations Research and Its Applications**, Xinjiang, China, pp. 380-389.
- LIN, C.-H., YU, J.-L., LIU, J.-C. e LEE, C.-J. (2008). Genetic algorithm for shortest driving time in intelligent transportation systems. **International Conference on Multimedia and Ubiquitous Engineering**, pp. 402-406.
- LUGER, G. F. (2004). **Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos**, 4ª ed., Bookman, Porto Alegre - RS.
- MCCULLOCH, W. S. e PITTS, W. (1943). A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, vol. 5, pp. 115-133.
- MICHALEWICZ, Z. (1996). **Genetic Algorithms + Data Structures = Evolution Programs**, 3ª ed., Springer-Verlag.
- MITCHELL, M. (1996). **An Introduction to Genetic Algorithms**, MIT Press, Cambridge, Massachusetts.

- MUNEMOTO, M., TAKAI, Y. e SATO, Y. (1998). A migration scheme for the genetic adaptive routing algorithm. **IEEE International Conference on Systems, Man, and Cybernetics**, pp. 2774-2779.
- PAPADIMITRIOU, C. H. e STEIGLITZ, K. (1998). **Combinatorial Optimization: Algorithms and Complexity**, Dover Publications, Mineola, NY.
- PARK, D.-C. e CHOI, S.-E. (1998). A neural network based multi-destination routing algorithm for communication network. **IEEE International Joint Conference on Neural Networks**, vol. 2, pp. 1673–1678.
- PETERSON, C. e SODERBERG, B. (1989). A new method for mapping optimization problems onto neural networks. **International Journal of Neural Networks**, vol. 1, pp. 3-22.
- PIRES, M. G., SILVA, I. N. e BERTONI, F. C. (2008a). NeuroGenetic Approach for Solving Dynamic Programming Problems. **IEEE International Conference on Systems, Man and Cybernetics**, Cingapura, pp. 2144-2149.
- PIRES, M. G., SILVA, I. N. e BERTONI, F. C. (2008b). Resolução do Problema do Caminho Mínimo usando Redes de Hopfield e Algoritmos Genéticos. **Congresso Brasileiro de Automática**, Juiz de Fora, MG, CD-ROM // Paper No. 40976 // 06 Páginas.
- PIRES, M. G., SILVA, I. N. e BERTONI, F. C. (2008c). Solving Shortest Path Problem Using Hopfield Networks and Genetic Algorithms. **International Conference on Hybrid Intelligent Systems**, Barcelona, Espanha, pp. 643-648.
- POOLE, D., MACKWORTH, A. e GOEBEL, R. (1998). **Computational Intelligence: A Logical Approach**, Oxford University Press, New York, NY.
- RAUCH, H. E. e WINARSKE, T. (1988). Neural networks for routing communication traffic. **IEEE Control Systems Magazine**, pp. 26-31.
- REZENDE, S. O. (2003). **Sistemas Inteligentes - Fundamentos e Aplicações**, 1ª ed., Manoele.

- ROSENBROOK, H. H. e STOREY, C. (1970). **Mathematics of Dynamical Systems**, Thomas Nelson & Sons.
- SILVA, I. N. (1995). **Estimação Paramétrica Robusta Através de Redes Neurais Artificiais**, Dissertação de Mestrado, Universidade de Campinas, DCA/FEE, Campinas - SP.
- SILVA, I. N., GOEDTEL, A. e FLAUZINO, R. A. (2007). The modified Hopfield architecture applied in dynamic programming problems and bipartite graph optimization. **International Journal of Hybrid Intelligent Systems**, vol. 4, pp. 17-26.
- THOMOPOULOS, S. C. A., ZHANG, L. e WANN, C. D. (1991). Neural network implementation of the shortest path algorithm for traffic routing in communication networks. **IEEE International Joint Conference on Neural Networks**, vol. 3, pp. 2693-2702.
- TOPKIS, D. M. (1988). A k shortest path algorithm for adaptive routing in communication networks. **IEEE Transactions on Communications**, vol. 36, n. 7, pp. 855-859.
- TOSCANI, L. V. e VELOSO, P. A. S. (2005). **Complexidade de Algoritmos**, 2ª ed., Sagra Luzzatto, Porto Alegre - RS.
- VIDYASAGAR, M. (1993). Location and stability of the high-gain equilibria of nonlinear neural networks. **IEEE Transactions on Neural Networks**, vol. 4, n. 4, pp. 660 - 672.
- WANG, J. (1993). Analysis and design of a recurrent neural network for linear programming. **IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications**, vol. 40, pp. 613-618.
- WANG, J. (1998). Primal and dual neural networks for shortest-path routing. **IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans**, vol. 28, pp. 864-869.

- WEN, U.-P., LAN, K.-M. e SHIH, H.-S. (2008). A review of Hopfield neural networks for solving mathematical programming problems. **European Journal of Operational Research**, doi:10.1016/j.ejor.2008.11.002 (in press).
- WILSON, V. e PAWLEY, G. S. (1988). On the stability of the TSP problem algorithm of Hopfield and Tank. **Biological Cybernetics**, vol. 58, pp. 63-70.
- WIRTH, N. (1976). **Algorithm + Data Structures = Programs**, Prentice Hall, Englewood Cliffs, NJ.
- WOLSEY, L. A. (1998). **Integer Programming**, John Wiley & Sons.
- WU, W. e RUAN, Q. (2004). A gene-constrained genetic algorithm for solving shortest path problem. **International Conference on Signal Processing**, vol. 3, pp. 2510-2513.
- ZIVIANI, N. (2005). **Projeto de Algoritmos: com Implementações em Pascal e C**, 2ª ed., Thomson Learning, São Paulo - SP.