Computação Bioinspirada

A.C.P.L.F. de Carvalho, A.C.B. Delbem, R.A.F. Romero, E.V. Simões, G.P. Telles

[1]Laboratório de Computação Bioinspirada–BioCom
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – Campus de São Carlos
Caixa Postal 668 – 13560-970
São Carlos - São Paulo - Brasil

{andre, acbd, rafrance, simoes, gpt}@icmc.usp.br

Abstract. *Traditional computing techniques are not efficient to solve a large number of problems, such as optimization and pattern recognition problems. For millions of years, nature has designed elegant and efficient solutions to such problems. Bioinspired Computing investigates how computers can be used to model nature and how solutions found by nature can originate new computing paradigms. In this text, the main concepts from this research area are presented.*

Resumo. *Técnicas tradicionais de computação não são eficientes para resolver um grande número de problemas como problemas de otimização e reconhecimento de padrões. Ao longo de milhões de anos de evolução, a natureza tem produzido soluções elegantes e eficientes para tais problemas. A Computação Bioinspirada investiga como computadores podem ser utilizados para modelar a natureza e como soluções encontradas pela natureza podem originar novos paradigmas de computação. Neste texto serão apresentados os principais conceitos desta área de pesquisa.*

1. Introdução

A natureza está repleta de exemplos de soluções elegantes e eficientes para um grande número de problemas. Estas soluções estão presentes nas mais diversas espécies de seres vivos. É fácil vê-las quando abelhas se organizam para construir uma colméia, facilitando a sobrevivência do enxame ou espalham o pólen de uma planta, permitindo a fecundação de uma outra planta, ou mesmo quando um urso fisga peixes em um rio, para suprir suas necessidades de alimento, entre vários outros exemplos. Estas soluções são, em geral, o resultado de um processo de evolução. Esta é uma das principais características presentes em sistemas biológicos. Outras características são:

- Controle por sistemas nervosos;
- Modulação por processos bioquímicos;
- Interação entre indivíduos de uma população.

Técnicas tradicionais de computação não são eficientes para resolver um grande número de problemas que são facilmente resolvidos pela natureza, como por exemplo, problemas de otimização e reconhecimento de padrões. A computação pode buscar

inspirações na Biologia para lidar com estes tipos de problemas. Este texto tem por objetivo apresentar os fundamentos e algumas das principais técnicas da área de pesquisa conhecida como Computação Bioinspirada.

Esta área de pesquisa da computação engloba diversas técnicas que buscam inspiração na Biologia, seja para definir seus princípios e mecanismos, seja para definir seus problemas alvo. Esta área de pesquisa investiga o relacionamento entre a computação e a biologia, estudando como computadores podem ser utilizados para modelar a natureza e como soluções encontradas pela natureza podem originar novos paradigmas de computação.

As técnicas da Computação Bioinspirada podem ser classificadas por diferentes critérios. Os critérios utilizados neste texto são a origem da inspiração, a dimensão do sistema biológico e o dispositivo físico utilizado para a implementação da técnica.

Com relação à origem da inspiração, as técnicas podem ser classificadas como:

- Computação Biológica ou Biocomputação: inclui técnicas utilizadas para o projeto de algoritmos computacionais baseadas em princípios biológicas. Fazem parte deste grupo: redes neurais artificiais, algoritmos evolutivos, sistemas imunológicos, otimização baseada em colônias de formigas, robótica adaptativa e evolutiva, computação baseada em moléculas, sistemas imunológicos, entre outras.
- Biologia Computacional ou Bioinformática: inclui técnicas computacionais empregadas para a resolução de problemas práticos da Biologia, geralmente da Biologia Molecular. Entre os problemas abordados, podem ser citados os problemas de alinhamento de seqüências, mapeamento físico de seqüências, agrupamento de ESTs, reconhecimento de genes, previsão de estrutura de proteínas, análise de expressão gênica e reconstrução de árvores filogenéticas.

Embora o termo Biologia Computacional seja freqüentemente associado à investigação de problemas da Biologia Molecular, este não se restringe a esta sub-área da Biologia. A Biologia Computacional também inclui:

- Neuroinformática: engloba técnicas computacionais empregadas para modelagem de neurônios naturais e sistemas nervosos. Investiga como ferramentas de computação podem ser utilizadas para a análise, modelagem, integração e compartilhamento das diferentes áreas da neurociências. Inclui, por exemplo, a análise de imagens de neurônios e a modelagem da dinâmica dos neurônios em um sistema nervoso;
- Computação Ambiental: reúne técnicas de computação utilizadas para resolver problemas relacionados ao meio ambiente. Entre os problemas tratados, pode-se destacar o controle de qualidade da água e de poluição atmosférica, a preservação do solo, a proteção contra ruídos, a purificação de água e a reciclagem.

Tem sido cada vez mais comum a utilização de técnicas de Computação Biológica em problemas de Biologia Computacional. Tais aplicações mostram como uma linha de pesquisa pode, simultaneamente, trazer benefícios para duas outras áreas, a Biologia e a Computação.

Uma outra forma de classificação considera o meio no qual a técnica é implementada. De acordo com ao meio físico utilizado em sua implementação, as técnicas de Computação Bioinspiradas podem ser ditas:

- Técnicas implementadas em computadores digitais: engloba as técnicas que são executadas em computadores digitais, em geral baseados na arquitetura de Von Neumann. Estas implementações simulam sistemas biológicos.
- Técnicas implementadas em computadores biológicos: inclui computadores construídos com matéria orgânica, por exemplo, baseados em moléculas de DNA ou em células nervosas.

Finalmente, com relação à dimensão do sistemas biológicos em que são baseadas, as técnicas de Computação Bioinspirada podem ser agrupadas em três grandes grupos:

- Molécula ou tecido: a inspiração ocorre em nível de moléculas (DNA, RNA ou proteína) ou tecidos. Engloba pesquisas em Bioinformática, que investigam moléculas de DNA, RNA e proteínas, e pesquisas em computação baseada em moléculas;
- Organismo: inspiradas na estrutura e comportamento de seres vivos observados de forma individual. Inclui técnicas como Redes Neurais Artificiais, Sistemas Imunológicos, Robótica Adaptativa;
- Sociedades ou Colônia: inspiradas no comportamento social de organismos, na forma como eles interagem. Inclui trabalhos que utilizam Computação Evolutiva, Inteligência de Enxames, Robôs Cooperativos.

Por limitação de espaço, neste texto serão discutidos um número limitado de técnicas de Computação Bioinspirada, a saber:

- Computação Baseada em Moléculas;
- Computação Evolutiva;
- Redes Neurais Artificiais;
- Inteligência de Enxames e Robótica Evolutiva;
- Robótica Adaptativa.

Esse texto está organizado da seguinte maneira: na Seção 2 são apresentados os principais conceitos de Computação Baseada em Moléculas. Na Seção 3, são apresentados os Algoritmos Evolutivos. Os principais fundamentos de Redes Neurais Artificiais são discutidos na Seção 4. A Seção 5 é dedicada à área de Robótica Adaptativa. Os conceitos básicos de Robótica Evolutiva e Inteligência de Enxames são comentados na Seção 6. Finalmente, na Seção 7 serão ilustradas algumas aplicações de técnicas de Computação Bioinspirada em problemas de Bioinformática e de Engenharia.

2. Bioinformática e Computação Baseada em Moléculas

Nesta seção são apresentados alguns conceitos de biologia molecular, bioinformática e computação baseada em moléculas. Os conceitos de biologia molecular que serão apresentados são básicos e foram incluídos nesta seção para facilitar a compreensão de idéias que aparecem ao longo do texto. Uma rápida definição de bioinformática e dos problemas que são estudados nesta área foi incluída nesta seção por ser uma área de aplicação importante das técnicas de computação bioinspirada. A biologia molecular e a bioinformática são estudadas em maior profundidade em vários livros e artigos, como por exemplo [Nelson and Cox, 2000, Setubal and Meidanis, 1997, Pevzner, 2000, Lesk, 2002].

2.1. Conceitos de Biologia Molecular e Bioinformática

A Biologia Molecular é a ciência que estuda a vida no interior das células do ponto de vista das moléculas e das interações entre elas. Nos últimos anos os métodos de investigação usados por esta ciência evoluíram muito, permitindo a geração e a análise de dados em larga escala. A Computação é parte integrante dos avanços na biologia molecular e contribui com algoritmos e equipamentos para armazenar, recuperar e analisar os dados gerados em laboratório pelos biólogos e, em alguns casos, é a computação que torna os métodos viáveis.

A aplicação da computação à problemas da biologia molecular é chamada de Bioinformática. Várias áreas da computação têm se envolvido com os problemas relacionados à biologia molecular, dentre elas teoria da computação, inteligência artificial, engenharia de software, sistemas distribuídos e bancos de dados. Em particular, costuma-se dar o nome de *biologia molecular computacional* ao projeto de algoritmos para problemas em biologia molecular e à análise de sua complexidade.

Uma célula é a menor unidade que exibe o comportamento que se chama vida. Nas células há três tipos de moléculas que têm funções muito importantes: as proteínas, o DNA e o RNA. As **proteínas** fazem parte de várias estruturas celulares e estão envolvidas em quase todos os processos em uma célula, como sinalização, transporte de moléculas, defesa, produção de energia e reprodução. Uma proteína é formada pela agregação de moléculas chamadas **aminoácidos**, formando uma cadeia de moléculas de nitrogênio, carbono e oxigênio alternantes. Há 20 aminoácidos diferentes e cada um deles é representado por uma letra distinta. Essa cadeia de aminoácidos dobra-se formando uma estrutura tridimensional que tem influência na função da proteína. Em vários casos uma proteína funcional é formada pelo agrupamento de mais de uma molécula de aminoácidos.

O **DNA** (sigla do inglês para ácido desoxiribonucléico) é o agrupamento de moléculas chamadas **nucleotídeos** ou **bases** ao longo de uma cadeia de moléculas de açúcar e fosfato alternantes. As bases que compõem o DNA são adenina, citosina, guanina e timina, representadas respectivamente pelas letras A, C, G e T. Pode-se pensar então no DNA como uma cadeia linear de letras, como no exemplo abaixo.

```
5’  ACCTGGATCAGCGCATAGATAACGAG  3’
```

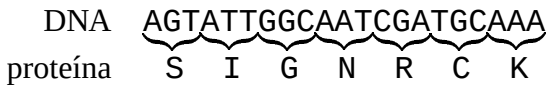
As extremidades livres do DNA são diferenciáveis; uma delas é chamada de 5’ e a outra de 3’. Em estado natural, o DNA forma uma estrutura chamada de **fita dupla**: duas cadeias de bases complementares e invertidas se mantém juntas através de pontes de hidrogênio. Na fita dupla, a extremidade 5’ de uma fita emparelha-se com a extremidade 3’ da outra, uma base A emparelha-se com uma base T e uma base C emparelha-se com uma base G, como no exemplo abaixo.

```
5’  ACTGAACATTCGTTCGAGG  3’
3’  TGACTTGTAAGCAAGCTCC  5’
```

As duas fitas do DNA são uma o complemento reverso da outra (com relação às regras de emparelhamento de bases) e são ditas complementares.

O RNA (do inglês para ácido ribonucléico) é uma molécula que tem estrutura semelhante à do DNA: é o agrupamento de bases ao longo de uma cadeia de moléculas de açúcar e fosfato alternantes. As bases do RNA são adenina (A), citosina (C), guanina (G) e uracila (U) e o RNA normalmente não forma fita dupla.

Uma célula produz proteínas a partir do seu DNA com a intervenção do RNA. Para isso a maquinaria celular primeiro separa as duas fitas do DNA e copia uma delas em uma molécula de RNA chamada RNA mensageiro (mRNA), através da regra da complementaridade (emparelhando A com U). Este processo de cópia é chamado de **transcrição**. Depois o mRNA é usado como molde para construção da proteína em estruturas chamadas de ribossomos, com a ajuda de moléculas de RNA transportador (tRNA), como no exemplo abaixo:



Este processo é chamado de **tradução**. A tabela que relaciona cada tripla de bases (chamada códon) a um aminoácido é chamada de código genético. O conjunto de todo o DNA de uma célula é chamado de **genoma** e está dividido em um ou mais cromossomos. Quando uma célula se duplica, todo o seu genoma é replicado. O genoma de bactérias tem alguns poucos milhões de pares de bases. A bactéria *Escherichia coli*, por exemplo, tem um genoma de aproximadamente 5×10^6 pares de bases. O genoma humano tem aproximadamente 3×10^9 pares de bases. Já uma proteína varia normalmente de uma dúzia a algumas centenas de aminoácidos.

Nem todo o DNA de uma célula codifica proteínas, apenas trechos chamados de **genes**, que são reconhecidos pela maquinaria celular. Além disso, nem todos os genes estão sendo usados como receita para proteínas todo o tempo. Diz-se que um gene que está dando origem a proteínas está sendo **expresso**. Células diferentes e em estágios de desenvolvimento ou condições diferentes expressam genes distintos e em intensidades diferentes. Estima-se que o genoma humano tenha em torno de 30.000 genes [Lander et al., 2001]. A bactéria *E. coli* tem aproximadamente 4.000 genes.

Usando técnicas em laboratório é possível extrair o DNA ou o RNA de células. Também é possível separar as duas fitas que formam o DNA, induzir a união de fitas simples de DNA que tenham seqüências complementares de bases, cortar o DNA em pontos específicos ou aleatórios, copiá-lo, estimar seu tamanho e marcá-lo com isótopos radioativos ou corantes fosforescentes que permitem detecção posterior. Pode-se ainda sintetizar pequenas cadeias de DNA com a seqüência de bases que se desejar, separar moléculas de DNA em função do seu tamanho aproximado e seqüenciar o DNA, isto é, obter a seqüência de bases que o compõe.

Um **projeto genoma** é um projeto que tem como objetivo determinar a seqüência de bases de todo o DNA de um organismo de interesse e determinar as características do genoma, identificando genes e outras estruturas importantes. A seqüência de bases do genoma será usada como fonte para pesquisas sobre o metabolismo e a vida do organismo. Outro tipo de projeto genoma é o Projeto EST. Neste tipo de projeto o alvo são as moléculas de mRNA. Elas são extraídas de células do organismo de interesse e sua

seqüência de bases é determinada. Esta abordagem identifica a seqüência de uma parte dos genes expressos pela célula no momento em que houve a extração das moléculas.

Muitos dos problemas em bioinformática envolvem o processamento de seqüências de caracteres que representam moléculas. Uma boa parte dos problemas é complicada pela presença de erros experimentais e pelo fato de que o conhecimento acerca dos processos biológicos é incompleto. Apenas para citar alguns desses problemas, alinhamento de seqüências, montagem de fragmentos de DNA, busca em bancos de dados de seqüências, mapeamento físico de DNA, agrupamento de ESTs, reconhecimento de genes, predição de estrutura tridimensional de RNAs e proteínas, análise de expressão gênica, reconstrução de árvores filogenéticas. Outros problemas importantes incluem o arranjamento e recuperação eficiente de dados.

As possibilidades de aplicação da ciência da computação em biologia molecular são vastas. À medida que esta ciência avança, novos problemas surgem e oferecem desafios que via de regra têm em comum a dificuldade para obter modelos e formalizações adequados.

2.2. Computação Baseada em Moléculas

Computação baseada em moléculas é uma forma de computação que usa enzimas e moléculas de DNA, RNA ou proteínas. Instâncias de problemas são codificadas como moléculas e reações químicas as transformam, de tal forma que ao final das reações novas moléculas codificam a resposta para o problema. Ao se tornarem operacionais, computadores baseados em biologia molecular serão capazes de armazenar volumes enormes de dados em espaço mínimo (1 bit por nm^3 em moléculas de DNA contra 1 bit por 10^{12} nm^3 em uma fita de vídeo) e realizar um número muito grande de operações (da ordem de 10^{20} operações por segundo), consumindo 10 ordens de grandeza menos energia por operação. Provavelmente os computadores baseados em DNA serão usados em conjunto com os computadores baseados em silício, resolvendo problemas cuja solução exija grande esforço computacional.

Neste capítulo mostra-se como o cientista da computação Leonard Adleman, em 1994 [Adleman, 1994], deu o passo inicial na computação baseada em moléculas. Ele demonstrou que a técnica funciona ao resolver um problema complexo usando DNA. Além disso, neste capítulo mencionam-se outras contribuições em direção a um computador baseado em moléculas.

O problema que Adleman escolheu foi o do caminho Hamiltoniano em grafos. Intuitivamente, dado um mapa de uma cidade em que as mãos de direção das ruas estão indicadas e dados endereços de entregas para um caminhão, o problema é encontrar uma rota que começa em um ponto inicial do mapa (por exemplo, um depósito), passa por todos os endereços uma única vez respeitando a mão de direção das ruas e termina em um outro ponto do mapa (por exemplo, uma garagem).

Formalmente, um grafo orientado G com vértices distinguíveis v_s e v_t tem um caminho Hamiltoniano se e somente se existe uma seqüência de arestas $e_1, e_2, \ldots, e_k$ que começa em v_s, termina em v_t e passa por todos os outros vértices uma única vez. Por exemplo, considerando-se $v_s = 1$ e $v_t = 7$, o grafo na Figura 1 contém o caminho Hamiltoniano

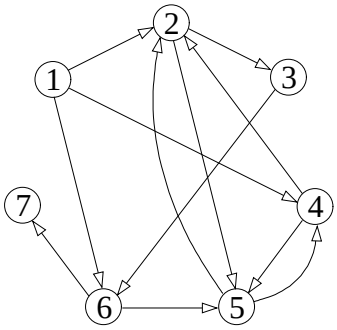


Figura 1: Grafo orientado com 7 vértices e 10 arestas. A seta indica a orientação da aresta, que intuitivamente é a direção em que pode-se ir de um vértice a outro.

$$(1,4),(4,5),(5,2),(2,3),(3,6),(6,7),$$

em que o par ordenado (i,j) representa a aresta que tem origem i e destino j.

O problema do caminho Hamiltoniano é um problema NP-completo, isto é, um problema para o qual não se conhece algoritmo eficiente. Isto implica que mesmo instâncias moderadas do problema (por exemplo, grafos com mais de 1000 vértices) demandam um tempo impraticável de computação. O Algoritmo 2.1 pode ser usado para resolver o problema.

Algoritmo 2.1 Algoritmo para caminho Hamiltoniano (não-determinístico) [Adleman, 1994].

1: Gere caminhos no grafo aleatoriamente
2: Mantenha apenas os caminhos que começam em v_s e terminam em v_t.
3: Se o grafo tem n vértices, mantenha apenas os caminhos com n vértices.
4: Mantenha apenas os caminhos que contém os vértices do grafo apenas uma vez.
5: Se resta algum caminho, responda sim. Senão responda não.

Para que esse algoritmo funcione em um computador digital, é necessária uma garantia de que, se houver caminho Hamiltoniano no grafo de entrada então esse caminho será gerado no Passo 1. Não é difícil perceber que ele não é muito eficiente: pode ser necessário gerar um número enorme de caminhos no Passo 1 e depois processar todos esses caminhos.

A idéia de Adleman foi codificar os vértices e arestas do grafo como moléculas de DNA e, através de ligações químicas entre as moléculas, construir todos os caminhos ao mesmo tempo no Passo 1, alcançando um grau de paralelismo muito maior do que é possível com computadores digitais. Todos os passos do algoritmo são feitos "em tubos de ensaio", usando técnicas da biologia molecular. A seguir, explica-se em mais detalhes a implementação de Adleman para um grafo com 7 vértices parecido com o da Figura 1.

Como foi visto na Seção 2.1, a molécula de DNA é formada por duas cadeias (ou fitas) complementares de DNA, isto é, uma base A emparelha com uma base T e uma base C emparelha com uma base G, como no exemplo a seguir:

ACTGAACATTCGTTCGAGG
TGACTTGTAAGCAAGCTCC

Viu-se também que em condições adequadas, as fitas de uma molécula de DNA podem ser separadas em solução e que, da mesma forma, moléculas de DNA em fita simples em solução podem se unir por complementaridade.

Adleman codificou cada vértice do grafo como uma seqüência aleatória de DNA com 20 bases. Para cada aresta (u, v) do grafo ele criou uma molécula de DNA com o complemento das 10 últimas bases do vértice u e com o complemento das 10 primeiras bases do vértice v. Exceto para as arestas da forma $(1, u)$ e $(u, 7)$, em que todas as bases dos vértices 1 e 7 foram adicionadas. Por exemplo, para codificar o grafo da Figura 1 pode-se construir as seguintes moléculas:

Vértices		Arestas	
1	ctgaatgtcatccaattgga	(1,2)	gacttacagtaggttaacctcgttttccgg
2	gcaaaaggccaacatgacct	(1,4)	gacttacagtaggttaacctcgacgctcgc
3	ggaggtatcgctacaggtga	(1,6)	gacttacagtaggttaaccttcatcctccc
4	gctgcgagcgtctgagacgc	(2,3)	ttgtactggacctccatagc
5	atcaccacgctcacgatgat	(2,5)	ttgtactggatagtggtgcg
6	agtaggagggcaatcacgaa	(3,6)	gatgtccacttcatcctccc
7	tgtcaagcaatcacttgacg	(4,2)	agactctgcgcgttttccgg
		(4,5)	agactctgcgtagtggtgcg
		(5,2)	agtgctactacgttttccgg
		(5,4)	agtgctactacgacgctcgc
		(6,5)	gttagtgctttagtggtgcg
		(6,7)	gttagtgcttacagttcgttagtgaactgc

Para gerar todos os possíveis caminhos no grafo, Adleman conduziu um experimento em que primeiro uma certa quantidade de moléculas representando vértices e arestas foi misturada em solução. Depois, as condições da solução foram alteradas para que as fitas se unissem. As uniões entre as moléculas no tubo de ensaios são aleatórias e o resultado é que todos os caminhos orientados no grafo são formados com alta probabilidade, por conta da forma das moléculas. Na Figura 2 estão ilustradas algumas moléculas que provavelmente são formadas pela reação entre os vértices e arestas construídos para o grafo.

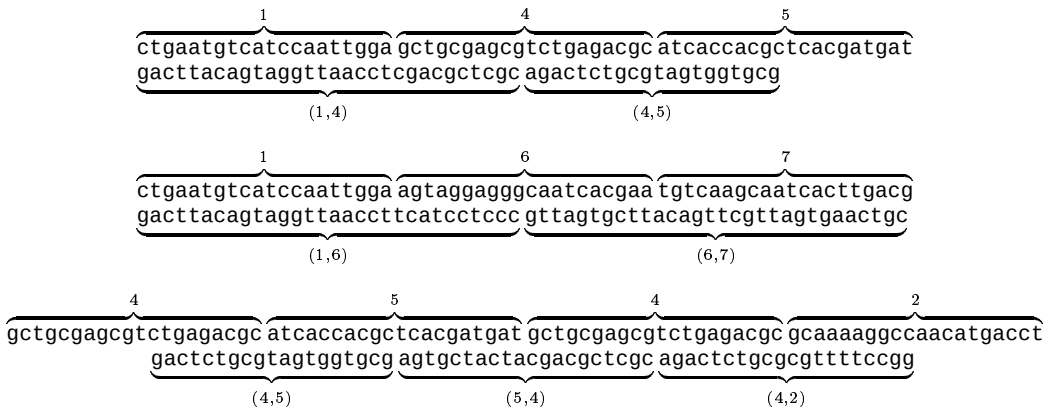


Figura 2: Exemplos de caminhos formados no grafo da Figura 1 pela reação das moléculas construídas para os vértices e arestas.

Os Passos de 2 a 5 consistiram da aplicação de algumas técnicas moleculares para selecionar as moléculas obtidas no Passo 1, de tal forma que restassem apenas as moléculas que correspondem a caminhos Hamiltonianos.

O procedimento de Adleman levou uma semana, mas a reação que gerou todos os caminhos codificados como moléculas de DNA levou menos de um segundo. Para a reação ele usou em torno de 3×10^{13} cópias de cada molécula representando vértices e arestas. Este volume de moléculas foi escolhido para garantir alta probabilidade de que, se houvesse um caminho Hamiltoniano no grafo então ele seria codificado pelas moléculas. O tamanho das moléculas que codificam vértices e arestas foi escolhido para que a probabilidade de que arestas e vértices incorretos se unissem fosse pequena e para que as moléculas formadas fossem estáveis à temperatura ambiente.

Mesmo tento usado um grafo pequeno, Adleman demonstrou a possibilidade de usar moléculas para computar problemas combinatórios que são difíceis de resolver usando computadores baseados em silício. A técnica é sensível a alguns tipos de falhas, como a formação de soluções que não são factíveis ou a perda de soluções factíveis durante o processamento *in vitro*.

Desde o trabalho de Adleman, alguns outros problemas foram resolvidos usando moléculas com intervenção humana no processamento [Lipton, 1995, Faulhammer et al., 1997]. A técnica desenvolveu-se de forma a construir um computador baseado em DNA autônomo, isto é, um conjunto de enzimas que quando misturado às moléculas de DNA que codificam o problema que se quer resolver, desencadeiam uma série de reações que resultam em cadeias de DNA que codificam a solução do problema sem intervenção humana [Mao et al., 2000, Benenson et al., 2001]. Outros trabalhos exploram os limites teóricos da computação com DNA [Boneh et al., 1996, Winfree et al., 1996] e mostram como contornar possíveis erros experimentais neste tipo de computação [Boneh et al., 1999].

Este conjunto de avanços aponta na direção da construção de um computador programável autônomo para resolver problemas gerais que não podem ser resolvidos em tempo aceitável pela tecnologia atual (como por exemplo, a quebra de algoritmos criptográficos [D. Boneh, 1995]).

Mais recentemente, outra possibilidade com sabor de ficção científica foi aberta: o uso de computadores baseados em DNA *in vivo*, atuando dentro de células de organismos para corrigir ou prevenir problemas de saúde, como câncer de próstata ou pulmão [Benenson et al., 2004], mostrando que esta técnica não está restrita aos limites da ciência da computação.

3. Computação Evolutiva

A Computação Evolutiva (CE) é um paradigma de computação inspirado na evolução natural formalizada por Darwin [Ridley, 1996]. Aplicados principalmente na resolução de problemas de otimização, os algoritmos baseados neste paradigma possuem características como auto-organização e comportamento adaptativo [Goldberg and Holland, 1988]. Uma das principais vantagens da CE é que seus algoritmos são constituídos por passos genéricos e adaptáveis, capazes de resolver um grande número de problemas práticos.

O paradigma evolutivo de solução de problemas “abre mão” da garantia de obtenção da solução ótima para conquistar a “tratabilidade” de problemas computacionalmente complexos [Zuben, 2000] por meio de uma ferramenta de propósito geral, sem perder a capacidade de evitar ótimos locais. Deve-se observar que técnicas que utilizam heurísticas, em geral, produzem soluções em tempos de computação aceitáveis, mas facilmente convergem para ótimos locais.

3.1. Algoritmos Evolutivos

Algoritmos inspirados na CE apresentam uma seqüência de procedimentos gerais que podem ser adaptados a cada problema. Os dois aspectos destacados a seguir são fundamentais para o desempenho de uma abordagem evolutiva:

- codificação dos indivíduos em representação genotípica, em que cada indivíduo representa um candidato à solução de um dado problema;
- uma função de adaptação que indica o valor de adequação de cada indivíduo, mostrando o quão próximo está este valor da solução procurada, dentro do contexto de cada problema.

A representação genotípica corresponde a uma descrição de cada indivíduo da população por meio de uma estrutura de dados (vetor, lista ordenada ou árvore) denominada cromossomo ou genótipo, descrito a partir de um alfabeto finito. Cada atributo da estrutura é equivalente a um gene, onde os possíveis valores que um determinado gene pode assumir são denominados alelos e a posição do cromossomo em que estes aparecem é chamada de locus. Estas soluções são avaliadas por algum critério, produzindo valores (ou *fitness*) que indicam o quanto cada uma das soluções está adaptada ao modelo. Esses valores de adaptação são utilizados na seleção de indivíduos para reprodução (geração de novas soluções com base em soluções anteriores), simulando o processo de seleção natural.

Um conjunto de possíveis soluções é chamado população. Em geral, a população inicial corresponde a soluções geradas aleatoriamente. Novos indivíduos são gerados aplicando operadores de reprodução a indivíduos selecionados da população. Esses operadores alteram os atributos simulando os processo de *crossover* e a mutação que ocorrem nos seres vivos. Os indivíduos mais aptos tendem a ser mantidos (sobreviver) formando uma nova população. Cada ciclo de produção de uma nova população é chamado de geração. Desta forma, os indivíduos são submetidos a um processo de evolução que corresponde a um procedimento de busca em um espaço de possíveis soluções para um problema.

As técnicas de CE têm em comum a manutenção de uma população de soluções potenciais, a utilização de processos baseados na adaptação de um indivíduo e o emprego de operadores para a produção de novos indivíduos. Diversas abordagens para sistemas baseados em evolução têm sido propostas, as principais diferenças entre elas estão nos método de seleção e operadores empregados. No entanto, existem elementos adicionais que diferenciam as abordagens evolutivas, tais como a estrutura de dados utilizadas para codificar um indivíduo e os métodos utilizados na geração da população inicial. Por outro lado, elas compartilham o princípio comum de que uma população de indivíduos é transformada ao longo das iterações e a competição dos indivíduos pela sobrevivência produz a evolução da população.

As abordagens mais freqüentemente utilizadas têm sido:

- os algoritmos genéticos (AGs), cujos princípios podem ser encontradas em trabalhos de John Holland na década de 1960 [Jong et al., 2000]. Os algoritmos genéticos foram desenvolvidos com o propósito de formalizar matematicamente e explicar rigorosamente processos de adaptação em sistemas naturais e desenvolver sistemas artificiais (simulados em computador) que buscam simular os mecanismos originais encontrados em sistemas naturais [Zuben, 2000];
- as estratégias evolutivas (EEs), propostas na década de 1960 com o objetivo de solucionar problemas de otimização de parâmetros [Jong et al., 2000]. Por empregarem apenas operadores de mutação, importantes contribuições para a análise e síntese destes operadores foram elaboradas [Zuben, 2000];
- a programação evolutiva (PE), originalmente proposta por Fogel *et al.* [Fogel et al., 1996], tem como principal objetivo criar inteligência artificial por meio da evolução de máquinas de estado finito [Jong et al., 2000]. A programação evolutiva também emprega apenas operadores de mutação;
- a programação genética (PG), uma extensão dos AGs e que foi desenvolvida com o intuito de evoluir um conjunto de programas de computador, ao invés de um vetor de bits.

Apesar dessas abordagens terem sido desenvolvidas de forma independente, seus algoritmos possuem uma estrutura comum [Michalewicz, 1996]. O termo algoritmo evolutivo (AE) é utilizado como uma denominação comum para essas abordagens. A estrutura básica de um AE é apresentada no Algoritmo 3.1.

Algoritmo 3.1 - Programa evolutivo - transcrito do livro de Michalewicz [Michalewicz, 1996].

```
1: t ← 0;
2: inicialize P(t);
3: avalie P(t);
4: while (not(condição de parada)) do
5:     t ← t + 1;
6:     selecione P(t) a partir de P(t − 1);
7:     altere P(t);
8:     avalie P(t);
9: end while
```

Um algoritmo evolutivo mantém uma **população** de indivíduos $P(t)$ na iteração **(geração)** t. Cada indivíduo representa um candidato à solução do problema em estudo e é avaliado por meio de uma função de avaliação. Em seguida, são selecionados os indivíduos mais adaptados, formando a população da geração $t + 1$. Parte dos indivíduos selecionados são submetidos a um processo de alteração por meio de operadores de reprodução: mutação e/ou *crossover*, para formar novas soluções. O operador de **mutação** cria novos indivíduos por meio de pequenas modificações de atributos de um indivíduo. O operador de ***crossover*** gera novos indivíduos pela combinação de dois ou mais indivíduos. A cada geração verifica-se a condição de parada do algoritmo. Esta condição geralmente indica a existência na população de um indivíduo que represente uma solução aceitável para o problema ou que o número máximo de gerações foi atingido.

Em AGs um indivíduo da população é geralmente representado por um único cromossomo, o qual contém a codificação (genótipo) de uma possível solução do problema

(fenótipo).

Nas próximas seções são discutidos os principais aspectos de um AG.

3.1.1. Codificação do Problema

O primeiro aspecto a ser considerado na implementação de um AG é a representação do problema investigado, isto é, a representação escolhida deve mapear o espaço de soluções no espaço da codificação de modo a preservar as principais características do problema original. Uma codificação inadequada pode levar a problemas de convergência prematura do AG, que em geral indica a obtenção de um ótimo local.

A representação de um problema, codificada em um cromossomo, deve ser a mais simples possível. O cromossomo pode ser representado por números inteiros, reais ou binários.

Em problemas de otimização restrita, a codificação adotada pode produzir indivíduos inválidos pelos operadores de *crossover*/mutação. Assim, cuidados especiais devem ser tomados na definição da codificação e/ou dos operadores que utilizam essa codificação.

3.1.2. Definição da População Inicial

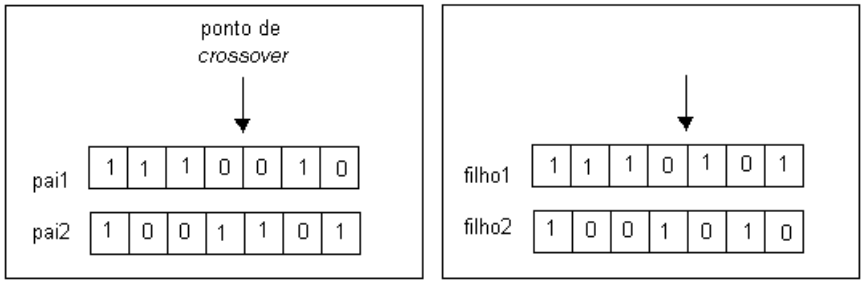
O método utilizado na criação da população inicial é a inicialização aleatória dos indivíduos dentro do espaço de solução [Zuben, 2000]. É importante que a população inicial corresponda a um distribuição de soluções aproximadamente uniforme sobre o espaço de busca. Se houver algum conhecimento prévio a respeito do problema, esse pode ser utilizado na inicialização da população. Em problemas com restrições, deve-se evitar a geração de indivíduos inviáveis na etapa de inicialização.

3.1.3. Operadores de Reprodução

O principal objetivo dos operadores de reprodução é transformar a população por meio de sucessivas gerações, até alcançar um resultado satisfatório. Os operadores de reprodução são essenciais na introdução da variabilidade da população e manutenção de características de adaptação adquiridas pelas gerações anteriores. Os principais operadores utilizados em AEs são os operadores de *crossover* e mutação. O operador de *crossover* ou recombinação cria novos indivíduos por meio da combinação de dois ou mais indivíduos, levando-se em conta a probabilidade de cruzamento (taxa de *crossover*). O operador de *crossover* pode ser empregado de várias maneiras:

- Um-ponto: um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas;
- Multi-pontos: é uma generalização da idéia anterior, onde muitos pontos de cruzamento podem ser utilizados;
- Uniforme: não utiliza pontos de cruzamento, mas determina por meio de uma máscara qual a probabilidade de cada gene ser trocado entre os pais.

O operador mais comumente empregado é o *crossover* de um-ponto. Para a aplicação deste operador, são selecionados dois indivíduos (pais), como ilustrado na Figura 3(a), e aleatoriamente é escolhido um ponto de corte nos pais. Os segmentos de cromossomos criados a partir do ponto de corte são trocados, formando os indivíduos filhos.



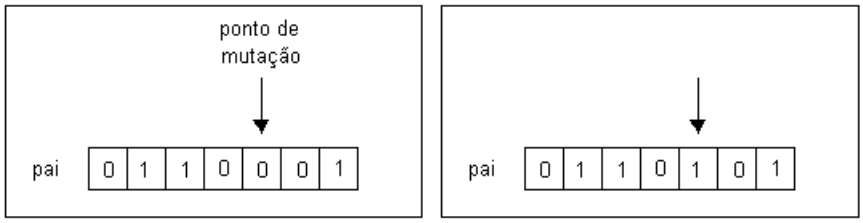
(a) Representação de dois indivíduos.

(b) Criação de dois novos indivíduos.

Figura 3: Exemplo de aplicação do operador de *crossover*.

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene é denominada taxa de mutação. Geralmente, são atribuídos valores pequenos para a taxa de mutação. A idéia intuitiva por trás do operador de mutação é criar uma diversidade extra na população, de forma a evitar que as soluções obtidas se concentrem em um máximo (mínimo) local. Esses operadores permitem a exploração de novas regiões no espaço de soluções sem destruir o progresso já obtido com a busca.

Considerando a codificação binária, o operador de mutação padrão simplesmente inverte o valor de um gene em um cromossomo. Assim, se um gene selecionado para mutação tem valor 1, o seu valor passará a ser 0 e vice-versa, como ilustrado nas Figuras 4(a) e 4(b).



(a) Operador de mutação a ser submetido em uma posição do cromossomo.

(b) Indivíduo após a aplicação do operador de mutação.

Figura 4: Exemplo da aplicação do operador de mutação.

3.1.4. Seleção

Dada uma população em que cada indivíduo possui um valor de aptidão, vários métodos podem ser utilizados para selecionar os indivíduos sobre os quais serão aplicados os operadores de reprodução. Dentre esses métodos, os mais comuns são a seleção por roleta e a seleção por torneio.

Na seleção por roleta, cada indivíduo da população é representado em uma roleta proporcionalmente ao seu índice de aptidão [Goldberg, 1989, Michalewicz, 1996, Mitchell, 1996]. Desta forma, os indivíduos com alta aptidão ocupam uma porção maior da roleta, enquanto indivíduos de aptidão mais baixa preenchem uma porção relativamente menor. O método da roleta pode ser descrito pelo Algoritmo 3.2:

Algoritmo 3.2 - Método da Roleta - adaptado do livro de Goldberg [Goldberg, 1989].

1: $T \leftarrow$ soma dos valores de aptidão de todos os indivíduos da população;
2: $i \leftarrow 1$;
3: $S \leftarrow 0$;
4: $r \leftarrow$ valor aleatório no intervalo [0 e T];
5: **repeat**
6: $\quad S \leftarrow S+$ aptidão do indivíduo i;
7: $\quad i \leftarrow i + 1$;
8: **until** ($S >= r$ **or** $i =$tamanho da população);
9: **return** i;

Na seleção por torneio, s indivíduos da população são escolhidos aleatoriamente. O indivíduo deste grupo que apresentar maior (menor) valor de aptidão é geralmente selecionado. De acordo com Mitchell [Mitchell, 1996], a seleção por torneio para $s = 2$ pode ser implementada da seguinte maneira:

Algoritmo 3.3 - Seleção por torneio - adaptado do livro de Mitchell [Mitchell, 1996].

1: $k \leftarrow 0.75$;
2: Escolha 2 indivíduos da população aleatoriamente;
3: $r \leftarrow$ valor aleatório entre 0 e 1;
4: **if** ($r < k$) **then**
5: $\quad$ O indivíduo com maior aptidão, dentre os dois indivíduos selecionados anteriormente, é escolhido;
6: **else**
7: $\quad$ O indivíduo com pior aptidão é selecionado;
8: **end if**

A estratégia mais empregada para melhorar a convergência dos AGs é o Elitismo [Mitchell, 1996]. O elitismo passa um certo número de "melhores" indivíduos para a próxima iteração. Tais indivíduos poderiam ser perdidos se não fossem selecionados para reprodução ou se fossem destruídos por cruzamento ou mutação.

Geralmente, a elite tem um tamanho reduzido, 1 a 2 indivíduos para cada 50 indivíduos, e a sua amostragem pode ser direta, escolhendo-se simplesmente os melhores, ou por sorteio, escolhendo-se os melhores entre os melhores da população.

4. Redes Neurais Artificiais

Redes Neurais Artificiais, RNs, são modelos computacionais paralelos e distribuídos baseados na estrutura e funcionamento do cérebro. As RNs são inspiradas na forma como a estrutura paralela e densamente conectada do cérebro processa informação. Esta característica faz das RNs uma alternativa promissora à computação tradicional para aqueles problemas tratados com relativa facilidade por seres vivos, mas não por técnicas convencionais de computação. Também como os seres vivos, as RNs possuem a capacidade de aprender por meio da experiência. As principais características de uma RN são portanto sua arquitetura e capacidade de aprendizado.

A arquitetura de uma RN é tipicamente composta por um número unidades de processamento simples (nodos), que simulam simplificadamente os neurônios naturais, e suas conexões, que simulam o papel das sinapses.

RNs apresentam a capacidade de aprender a partir da apresentação de um conjunto de exemplos ou padrões de treinamento. Idealmente, exemplos de treinamento são exemplos representativos de situações que podem ser encontradas pela rede durante a sua utilização. Assim, ao contrário de técnicas de computação convencionais, soluções baseadas em RNs não precisam ser explicitamente programadas. Suponha, por exemplo, que o problema a ser resolvido seja o diagnóstico médico de um paciente baseado em um conjunto de sintomas. RNs podem ser treinadas com um conjunto de exemplos formados por quadros de sintomas passados e seus diagnósticos ou classes. Após seu treinamento, a RN deve ser capaz de para sugerir o diagnóstico para novos pacientes.

4.1. Arquitetura

A arquitetura de uma RN é composta por um conjunto de nodos e de conexões interligando nodos entre si ou nodos a elementos de entrada da rede. Os nodos de uma rede são dispostos em uma ou mais camadas. Dependendo do número de camadas empregadas, uma rede pode ser classificada como de uma única camada ou multicamadas. Independentemente do número de camadas, de acordo com o sentido de suas conexões, uma rede pode ser classificada como *feedforward* ou recorrente. A Figura 5 ilustra exemplos de redes pertences a estas classes.

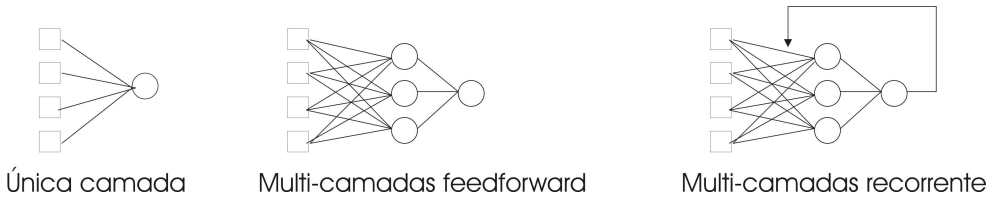


Figura 5: Exemplos de redes com diferentes arquiteturas

Ainda referente a sua arquitetura, a abrangência das conexão entre neurônios dentro um RN pode assumir as seguintes formas:

- Completamente conectadas: cada nodo de uma camada está conectado a todos os nodos da camada anterior ou a todos os elementos de entrada da rede;
- Parcialmente conectadas: cada nodo de uma camada está conectado a alguns dos nodos da camada anterior ou a alguns dos elementos de entrada da rede;

- Localmente conectadas: é um tipo de conexão parcial, em que os nodos da camada anterior ou elementos de entrada conectados pertencem a uma região contígua.

Estes diferentes arranjos de conexões podem ser melhor compreendidos por meio dos exemplos ilustrados na Figura 6.

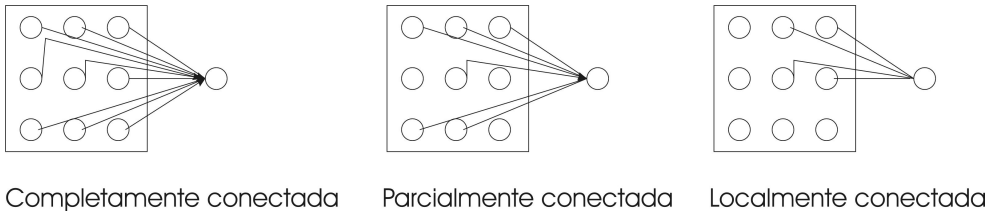


Figura 6: Diferentes arranjos de conexões em redes multi-camadas.

A computação realizada por cada nodo da rede é muito simples, geralmente funções matemáticas não-lineares. Cada nodo apresenta conexões de entrada e de saída. A cada conexão é associado um peso, valor numérico que pondera a influência da entrada recebida pelo nodo através desta conexão no cálculo de seu valor de saída. Os pesos armazenam o conhecimento representado na rede.

Assim, o valor de saída de um nodo é o resultado da aplicação de uma função de ativação à soma ponderada de suas entradas. A Equação 1 ilustra o cálculo da saída de um nodo n da rede como função da entrada total por ele recebida.

$$w_{ij}(0) = \frac{1}{1+n}, \forall i, j \tag{1}$$

Diferentes funções de ativação têm sido propostas na literatura. As mais utilizadas são as funções degrau, linear, sigmoid, gaussiana e tangente hiperbólica.

A definição da arquitetura de uma RN não é uma tarefa simples, sendo geralmente realizada por uma busca exaustiva. Existe um grande número de trabalhos que investigam alternativas para automatizar a definição da arquitetura de RNs. Uma das alternativas mais bem sucedidas emprega AGs para o projeto de redes.

4.2. Aprendizado

O objetivo do processo de treinamento é extrair o conhecimento necessário para a resolução do problema em questão. O conhecimento armazenado nos pesos é usado, posteriormente, para gerar a resposta da rede para novos padrões.

O aprendizado em sistemas biológicos envolve ajustes nas sinapses que existem entre os neurônios. De forma similar, o aprendizado em RNs envolve o ajuste iterativo dos pesos de suas conexões por meio da apresentação de um conjunto de padrões de treinamento. O ajuste dos pesos é realizado pela utilização de um algoritmo de aprendizado, seguindo um certo paradigma de aprendizado.

Um algoritmo de aprendizado compreende um conjunto de regras que define quando e como os pesos têm seus valores modificados. Existem diversos algoritmos de aprendizado, que podem ser divididos em quatro grandes categorias:

- Aprendizado por correção de erros: o ajuste dos pesos é efetuado de forma a minimizar o erro cometido pela rede para os padrões de treinamento;
- Aprendizado Hebbiano: o ajuste do peso de uma conexão está associado à correlação dos valores de saída produzidos pelos nodos conectados;
- Aprendizado competitivo: nodos da rede competem entre si para definir qual produzirá a resposta da rede;
- Aprendizado de Boltzman: processo estocástico baseado em termodinâmica e teoria da informação;

Enquanto o algoritmo de aprendizado define as regras empregadas para o ajuste de pesos, a iteração da rede com o ambiente externo durante seu treinamento e, conseqüentemente, a natureza dos exemplos de treinamento apresentados à rede, é definida pelo paradigma de aprendizado. Três diferentes paradigmas de aprendizado podem ser seguidos:

- Aprendizado supervisionado: junto com o padrão de entrada, é fornecida à rede a resposta desejada para aquele padrão. A RN ajusta seus pesos com o intuito de produzir a resposta desejada para o padrão apresentado.
- Aprendizado por reforço: aprendizado por tentativa e erro em que a rede é recompensada em caso de sucesso e punida em caso de fracasso;
- Aprendizado não supervisionado: não é fornecida à rede a resposta desejada para cada padrão de entrada, nem valor de punição ou recompensa. A RN extrai características estatisticamente relevantes dos padrões de entrada.

Finalizado o processo de treinamento, espera-se que a RN apresente a capacidade de classificar com sucesso padrões novos, não apresentados na fase de treinamento. A esta capacidade dá-se o nome de generalização.

Todo o conhecimento adquirido pela rede durante seu treinamento é codificado nos pesos associados às suas conexões. Como este conhecimento não é facilmente interpretado, técnicas de extração de conhecimento têm sido empregadas para explicar, de forma mais amigável, o conhecimento embutido na rede. Geralmente, esta explicação assume o formato de um conjunto de regras lógicas.

O número de modelos de RNs propostos na literatura já passa de uma centena. Existem tanto modelos gerais, utilizados nas mais diversas aplicações, quanto modelos específico para uma classe restrita de aplicações. A seguir serão apresentados os principais modelos.

4.3. Principais modelos

As primeiras RNs implementadas, as redes Perceptron [Rosenblatt, 1962] e Adaline [Widrow and Hoff, 1960], possuem uma única camada. Apesar de simples, este tipo de rede tem sido utilizada em diversas aplicações. Figura 7 ilustra uma rede Perceptron.

A rede Perceptron utiliza um algoritmo de aprendizado supervisionado por correção de erros. Rosenblatt provou um teorema para a rede Perceptron, denominado teorema da convergência. Este teorema afirma que se uma rede Perceptron pode resolver um dado problema, seu algoritmo de treinamento encontrará os valores de peso necessários para que a rede resolva este problema. Uma variante da rede perceptron, a rede

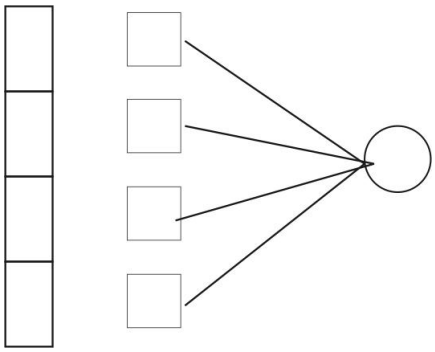


Figura 7: Rede Neural Perceptron.

Adaline, aprimorou o algoritmo de aprendizado utilizado pela rede Perceptron. Este algoritmo aprimorado, denominado Algoritmo de Widrow-Roff ou Regra Delta, é apresentado no Algoritmo 4.1.

Algoritmo 4.1 Algoritmo Regra Delta

```
1:  Inicializar pesos da rede com valores aleatórios;
2:  repeat
3:      erro_total = 0;
4:      for all cada padrão de treinamento do
5:          calcular saida_produzida do nodo;
6:          erro = saida_desejada - saida_produzida;
7:          ajustar pesos do nodo;
8:          erro_total = erro_total + erro;
9:      end for
10: until erro_total > valor_desejado
```

A rede Perceptron original utiliza apenas um nodo cujos pesos são ajustados durante seu treinamento. Uma limitação das RNs com uma única camada é sua incapacidade para lidar com problemas que não sejam linearmente separáveis[1] [Minsky and Papert, 1969].

Problemas não-linearmente separáveis podem ser tratados por RNs com uma ou mais camadas intermediárias. Apenas em 1986 foi apresentado um algoritmo de treinamento para redes multicamadas, mais esspecificamente para redes Perceptron multicamadas ou MLP (do inglês *Multi-Layer Perceptron*). Este algoritmo é denominado algoritmo *backpropagation* [Rumelhart et al., 1986]. Redes MLPs treinadas com o algoritmo *backpropagation* têm sido um dos modelos de RNs mais utilizados em aplicações [Haykin, 1999, Braga et al., 2000], inclusive em problemas de Biologia Molecular [Towell et al., 1990, Rampone, 1998] e Engenharia [Delbem et al., 2003, Leite et al., 2002, Rodgher et al., 1997]. A Figura 8 ilustra uma típica rede MLP.

Assim como a Regra Delta, o algoritmo *backpropagation* é um algoritmo de aprendizado supervisionado por correção de erros. O treinamento usando esse algoritmo ocorre em duas fases, cada uma percorrendo a rede em um sentido: fase *forward*, em que

[1] Um conjunto de padrões é linearmente separável se padrões de classes diferentes podem ser separados por um hiperplano [Mitchell, 1997].

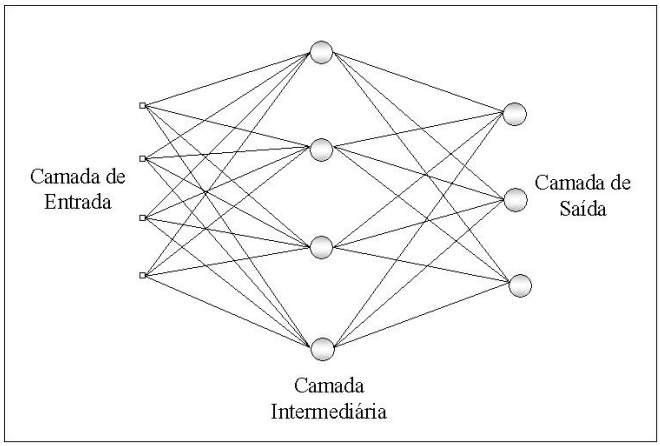


Figura 8: Rede Neural MLP

é produzida a saída da rede para um dado padrão de entrada, e fase *backward*, em que os pesos das conexões da rede são atualizados de acordo com o erro, calculado a partir da diferença entre a saída desejada e a saída produzida pela rede. O algoritmo *backpropagation* é baseado na Regra Delta, sendo também conhecido como Regra Delta generalizada (por se aplicar a rede com uma ou mais camadas de nodos).

Existem diversas variações do algoritmo *backpropagation*, que têm como objetivo acelerar o processo de treinamento e reduzir as taxas de erros obtidas. O Algoritmo 4.2 apresenta os principais passos desse algoritmo.

Algoritmo 4.2 Algoritmo Backpropagation.

```
1:  Inicializar pesos da rede com valores aleatórios;
2:  repeat
3:     erro_total = 0;
4:     for all cada padrão de treinamento do
5:        for all cada camada i da rede, i = 1, 2, ..., n do
6:           for all cada nodo n_ij da i-ésima camada do
7:              calcular saida_produzida do nodo;
8:           end for
9:        end for
10:       erro = saida_desejada - saida_produzida;
11:       for all cada camada i da rede, i = n, n − 1, ..., 1 do
12:          for all cada nodo n_ij da i-ésima camada do
13:             ajustar pesos do nodo;
14:          end for
15:       end for
16:       erro_total = erro_total + erro;
17:    end for
18: until erro_total > valor_desejado
```

RNs do tipo **Mapa Auto-Organizável** (SOM - do inglês *Self-Organizing Map*) são freqüentemente utilizadas para aprendizado não supervisionado [Kohonen, 1995]. As redes SOM combinam aprendizado competitivo e o conceito de vizinhança entre nodos.

Estas redes organizam os nodos em grades (ou mapas), onde cada nodo influencia o aprendizado dos nodos em sua vizinhança. A arquitetura de uma rede SOM é ilustrada na Figura 9.

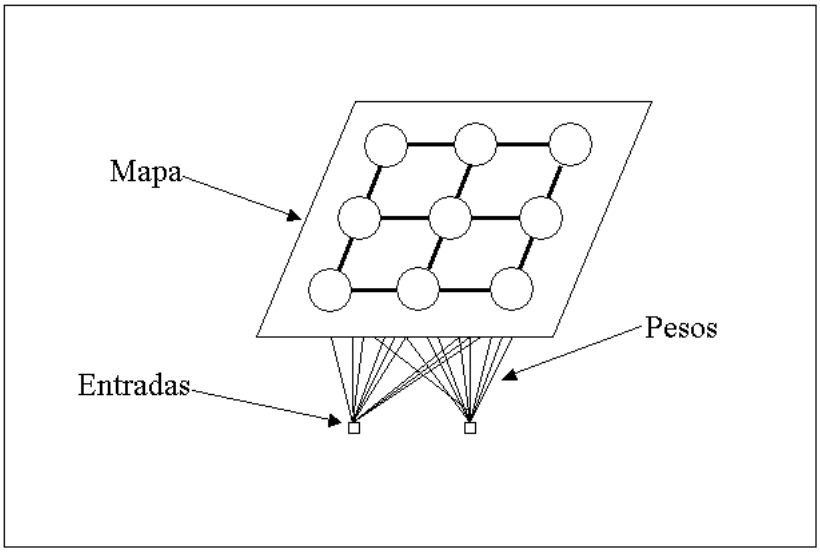


Figura 9: Exemplo de um topologia SOM 3x3 com duas entradas

Uma das principais características das redes SOM é a ordenação topológica dos agrupamentos gerados. De maneira intuitiva, o algoritmo treinamento das redes SOM funciona como segue. Inicialmente, escolhe-se uma grade (mapa) de nodos, por exemplo, um mapa 3×3. Cada elemento do vetor de entrada representa um atributo. Todos os nodos do mapa são ligados aos atributos de entrada por meio de conexões ponderadas.

Os pesos são inicialmente escolhidos aleatoriamente e depois ajustados de forma iterativa. Cada iteração envolve a escolha aleatória de um padrão do conjunto de treinamento, seguida de uma competição entre os nodos do mapa. Nessa competição, aquele nodo cujo vetor de pesos seja, de acordo com uma distância D, mais próximo ao vetor de entrada tem seus pesos (e dos nodos na vizinhança) ajustados na direção dos valores deste padrão de entrada. Assim, redes SOM impõem uma estrutura sobre os dados, em que nodos vizinhos tendem a definir aglomerados similares. A seguir, é apresentado o Algoritmo 4.3, uma versão simplificada do algoritmo utilizado para o treinamento de redes SOM.

As redes SOM são apropriadas para análise exploratória dos dados quando não há informação *a priori* sobre sua distribuição. Um problema com essa técnica é o grande número de parâmetros ajustáveis, que incluem a topologia, a taxa de aprendizado, a função de vizinhança, o raio de vizinhança, entre outros. O sucesso do mapa é dependente desses parâmetros [Kohonen, 1995].

4.4. Comentários Finais

As RNs apresentam uma série de vantagens, como tolerância a dados com ruídos, habilidade de representar qualquer função (linear ou não) e capacidade de lidar com padrões de entrada representados por vetores de alta dimensão, em que os valores dos atributos podem ser contínuos ou discretos. Os principais problemas são a dificuldade de definição de

Algoritmo 4.3 Algoritmo SOM.

```
1: Inicializar pesos da rede com valores aleatórios;
2: Definir raio e taxa de aprendizado iniciais;
3: while ocorrerem mudanças significativas no mapa do
4:    for all padrão de entrada  do
5:       for all nodo do
6:          Calcular a distância entre padrão de entrada e pesos do nodo;
7:       end for
8:       Selecionar nodo n_k com menor distância;
9:       Atualizar pesos do nodo n_k e de seus nodos vizinhos;
10:      Reduzir taxa de aprendizado e raio;
11:   end for
12: end while
```

seus parâmetros, como por exemplo, no caso das redes MLP, o número de nodos em suas camadas intermediárias, o tipo de função de ativação e o valor da taxa de aprendizado, além da dificuldade de compreensão dos conceitos aprendidos pela rede, codificados nos valores finais dos pesos da rede.

Por fim, é importante ressaltar que existem vários modelos de RNs, cada um deles mais adequado para um dado tipo de problema. Alguns dos modelos mais populares incluem as redes MLP com o *backpropagation* que foram revisados nesta seção, as redes do tipo mapa auto-organizável, redes com funções de base radial, redes de Hopfield, entre outras [Haykin, 1999].

As **Máquinas de Vetores Suporte** (SVMs - do inglês *Support Vector Machines*) constituem uma técnica de aprendizado que vem recebendo grande atenção nos últimos anos [Hearst et al., 1998]. Entre as principais características que popularizaram seu uso em Bioinformática estão sua boa capacidade de generalização e robustez diante de dados de grande dimensão, como os presentes em grande parte das aplicações envolvendo o reconhecimento de genes e a análise de dados de expressão gênica.

5. Inteligência de Enxames e Robótica Evolutiva

5.1. Inteligência de Enxames

O comportamento de colônias de animais tem servido de inspiração para o desenvolvimento de diversas técnicas baseadas em meta-heurísticas voltadas para problemas de otimização combinatória. Estas técnicas fazem parte de uma área de pesquisa denominada Inteligência de Enxames, IE, que investiga modelos de computação baseados em inteligência social ou coletiva.

Assim como Computação Evolutiva, IE é uma abordagem baseada em populações de indivíduos que interagem entre si. A Computação Evolutiva consiste em procedimentos de busca e otimização inspirados na biologia. Estes tentam abstrair e imitar algumas das características da evolução natural com o propósito de encontrar soluções ótimas para problemas que requerem adaptação, busca e otimização [Tomassini, 1995]. Sistemas baseados em enxames apresentam como características principais:

- Autonomia;
- Auto-organização;
- Comportamento emergente;
- Funcionamento distribuído.

O comportamento de diferentes espécies tem sido modeladas por esta abordagem. As principais espécies modeladas na literatura são:

- Colônias de formigas;
- Colônia de abelhas;
- Colônias de lagartos.

Dentre estas colônias, as colônias de formigas têm sido modeladas com mais freqüência, além de aplicadas a um universo maior de problemas práticos [Dorigo et al., 1999].

5.1.1. Colônias de formigas

A otimização baseada em colônias de formigas, ACO (do inglês *Ant Colony Optimization*), utiliza meta-heurísticas baseadas no comportamento de colônias de formigas para resolver problemas de otimização combinatória.

Embora as capacidades de uma única formiga sejam muito limitadas, a ação coletiva de formigas em colônias pode realizar tarefas que exigem um elevado grau de sofisticação. Um exemplo clássico é como uma colônia de formigas consegue determinar o menor caminho entre o formigueiro e uma fonte de alimentos.

As formigas comunicam-se entre si por meio de substâncias químicas, o feromônio. O feromônio exerce um papel importante na troca de informações entre as formigas para a realização de suas atividades básicas. Existem diferentes tipos de feromônios, que são utilizados para diferentes fins. O tipo de feromônio também varia de acordo com a espécie e colônia [Vittori, 2004].

Durante seu movimento, as formigas depositam trilhas de feromônio no trecho de solo percorrido. Como as formigas são atraídas pelo feromônio, outras formigas tendem a seguir o mesmo caminho. Inicialmente, formigas se movem de forma aleatória. Ao encontrar um caminho com feromônio, a formiga precisa decidir se segue ou não o caminho encontrado. Se ela seguir o caminho, ela depositará uma quantidade adicional de feromônio, aumentando a chance de que outras formigas percorram este caminho. Caminhos com uma maior quantidade de feromônio atraem uma maior quantidade de formigas.

Se existe mais de um caminho entre o formigueiro e a fonte de alimentos, caminhos aleatórios podem ser inicialmente formados. Supondo-se que n formigas percorrem seqüencialmente (uma após a outra) cada um destes caminhos e supondo-se que cada formiga vai do formigueiro à fonte de alimentos, coleta um alimento e retorna ao formigueiro. Caminhos mais curtos terão mais rapidamente formigas viajando nos dois sentidos e, conseqüentemente, um número maior de formigas. Quanto mais formigas no caminho, maior a quantidade de feromônio depositado. Logo, caminhos mais curtos atrairão mais formigas que caminhos mais longos. Tal fato é ilustrado na Figura 10.

A figura mostra as trilhas de feromônio depositadas por uma formiga no solo (linha tracejada) em dois possíveis caminhos para um intervalo de tempo t. Assumindo

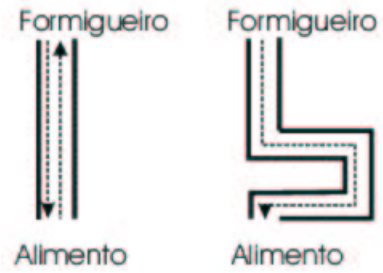


Figura 10: Trilha de feromônio em dois caminhos

que o caminho maior possui o dobro de distância do caminho menor, a quantidade de feromônio depositado no menor caminho, no intervalo de tempo considerado, é o dobro da quantidade depositada no maior caminho.

Técnicas baseadas em IE têm sido utilizadas em várias aplicações, dentre elas:

- Problema de Roteamento de Veículos;
- Problema do Caixeiro Viajante;
- Planejamento de Operação de Sistemas Hidrotérmicos;
- Robótica;
- Gerenciamento de portifólios.

O procedimento genérico que define o funcionamento de sistemas ACO pode ser descrito como um meta-algoritmo ou meta-heurística. Este procedimento envolve três passos, repetidos seqüencialmente até que algum critério de parada seja atingido [Dorigo and Gambardella, 1997]:

- Gerar formigas e ativá-las: as formigas são geradas, alocadas em posições iniciais e executam alguma tarefa, deixando um rastro de feromônio. Ao final deste passo as formigas "morrem".
- Evaporar o feromônio: ao término da tarefa das formigas, o sistema sofre uma atualização global de feromônio.
- Realizar ações "daemon": outras tarefas, como observar o comportamento das formigas ou coleta de informações globais podem ser executadas.

A partir desta descrição geral, vários sistemas foram construídos, como o ACS, descrito a seguir, o Ant-Q[Gambardella and Dorigo, 1995], o Ant System [Dorigo et al., 1999] e outros sistemas propostos mais recentemente [Dorigo, 2001]. O algoritmo *Ant Colony System* (ACS) é o mais conhecido entre os algoritmos de ACO (ver Algoritmo 5.1). Ele foi proposto por Dorigo e Gambardella [Dorigo and Gambardella, 1997] para resolver o problema do Caixeiro Viajante, no qual diversas formigas viajam entre cidades e o trajeto mais curto é reforçado. O processamento do ACS pode colocado do seguinte modo. Depois que as formigas são posicionadas em cidades iniciais, cada formiga realiza uma excursão. Durante a excursão, a regra de atualização local é aplicada e modifica o nível de feromônio das arestas. Quando as formigas terminam suas excursões, a regra de atualização global é aplicada, modificando outra vez o nível de feromônio. Este ciclo é repetido até nenhuma melhoria seja obtida ou um número fixo de iterações seja alcançado. Esta regra de transição favorece o uso de arestas com uma grande quantidade de feromônio e que tenham custo baixo (no problema do caixeiro viajante, favorece a escolha da cidade mais próxima).

Algoritmo 5.1 Algoritmo ACS.

```
1: Inicializar a tabela de feromônio, as formigas e a lista de cidades;
2: repeat
3:    for all formiga do
4:       colocar a formiga em uma cidade inicial;
5:       repeat
6:          Escolher a próxima cidade usando a regra de transição de estados;
7:          Atualizar a lista J_k de cidades a serem visitadas pela formiga k;
8:          Atualizar quantidade de feromônio nas trilhas;
9:       until Todas as cidades terem sido visitadas;
10:   end for
11: until Satisfazer critério de parada;
```

5.2. Robótica Evolutiva

Esta seção apresenta uma aplicação de Computação Evolutiva no desenvolvimento de uma estratégia de controle para sistemas multi-robôs. Assim, será descrito um sistema de controle distribuído e descentralizado para coordenar a navegação de robôs, buscando comportamentos exploratórios, desvio de obstáculos e coleta de objetos (forrageamento). Através do sistema evolutivo resultante, comportamentos complexos e adaptativos podem emergir da interação entre indivíduos simples.

Nos últimos anos, a robótica inteligente vem sendo uma das áreas de pesquisa que mais tem se desenvolvido [Dorigo and Caro, 1999]. Inúmeros esforços estão sendo despendidos para a descoberta de novas técnicas de controle de robôs autônomos. Neste contexto, a natureza tem inspirado muitos cientistas, com seus seres vivos e processos naturais, a criar alternativas às soluções tradicionais, como é o caso dos algoritmos evolutivos e inteligência de enxames (swarm intelligence) [Parker, 1993].

Existem duas abordagens principais para a utilização de robôs: sistemas com um único robô, que possui todos os componentes necessários para realizar uma determinada tarefa e sistemas multi-robôs, composto por um time de robôs mais simples, podendo haver subgrupos de robôs especializados em partes da tarefa. Sistemas multi-robôs têm a vantagem de reagirem robusta e efetivamente ao seu ambiente [Simões, 2000]. Caso um dos robôs falhe, os outros serão capazes de assumir o seu trabalho. Desta forma, as chances de realizar uma tarefa serão maiores. Já no sistema baseado em um único robô, se ocorrer uma falha, há uma chance muito maior da execução da tarefa ser comprometida [Ahmadabadi and Ghaderi, 2004]. Além disso, o robô pode ter que ser maior do que no sistema multi-robô, para que possa comportar os equipamentos, além de consumir mais energia, podendo deixá-lo com um custo maior que um conjunto de robôs simples. Uma característica de sistemas multi-robôs é a necessidade de controladores mais robustos, capazes de coordenar todos os robôs na execução e gerenciamento das tarefas, que envolvem comunicação e cooperação entre mais de um membro do time. Assim, este capítulo abordará a Computação Evolutiva no desenvolvimento de métodos de controle de sistemas multi-robôs, gerenciando aprendizado, execução de tarefas e planejamento, em ambientes complexos e dinamicamente mutáveis.

A Robótica Evolutiva surgiu da utilização de técnicas de Computação Evolutiva

para sintetizar automaticamente controladores embarcados para equipes de robôs, com o propósito de treiná-los para desenvolver tarefas específicas. Algoritmos Genéticos têm sido empregados com sucesso no desenvolvimento automatizado de controladores para robôs em diversas formas de abordagem. Em simulações, por exemplo, a população de robôs não existe fisicamente (Figura 5.2(a)), mas é representada matematicamente em software [Cliff and Miller, 1996]. Alguns pesquisadores empregam simuladores para desenvolver um controlador adequado e, depois, transferi-lo para um robô físico [Smith, 1998]. Outros conduzem sua pesquisa de maneira que o algoritmo genético é executado em simulação (Figura 5.2(b)), mas cada indivíduo da população é transferido para um robô real a fim de ser avaliado individualmente [Floreano and Mondada, 1995]. O problema desta abordagem é o fato de que os indivíduos da população não podem interagir uns com os outros e muitas não-linearidades de um sistema físico, onde todos os indivíduos existem e interagem em tempo real, não estão presentes [Jakobi et al., 1995].

Experimentos publicados por Floreano e Mondada [Dautenhahn and Nehaniv, 1998] fazem uso de dois ou mais robôs reais, mas ainda assim, conectados por um cordão umbilical a um computador externo, onde o algoritmo genético estava presente (Figura 5.2(c)). Mais recentemente, foram publicados experimentos em que o sistema evolutivo está totalmente embarcado em robôs reais (Figura 5.2(d)), tornando-os autônomos não apenas na tarefa que realizam, mas também na execução de seu algoritmo genético [Watson et al., 1999] [Simões and Dimond, 1999]. Um sistema evolutivo embarcado é aquele em que o processo evolutivo ocorre sobre uma população de robôs reais, completamente independentes de computação externa ou da intervenção do usuário a fim de avaliar, reproduzir e reposicionar os robôs para novos testes na geração seguinte [Simões and Dimond, 1999].

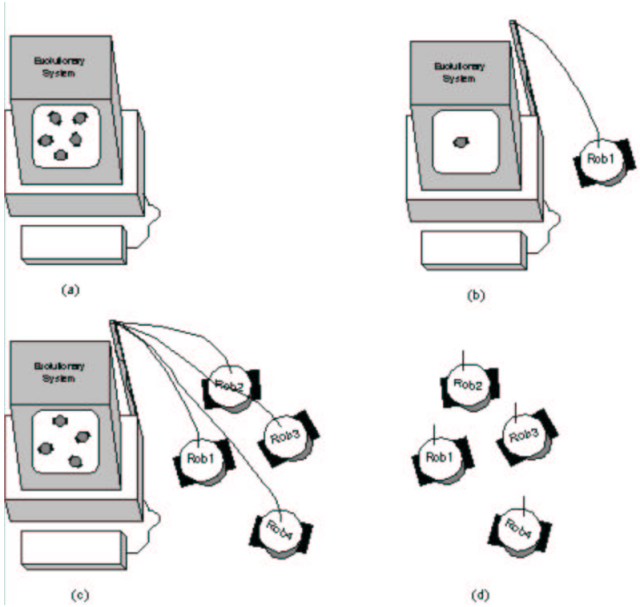


Figura 11: Diferentes maneiras de se empregar um sistema evolutivo em robótica: (a) ambiente simulado; (b) ambiente simulado com avaliação em robô real; (c) população real conectada ao computador externo, onde o algoritmo genético reside; e (d) população real com sistema evolutivo embarcado, independente de computador externo.

Os trabalhos realizados com robótica evolutiva embarcada são muito recentes e as tarefas abordadas ainda são muito simples [Watson et al., 1999]. Porém, a metodologia disponível já é eficaz em produzir controladores complexos. Nos últimos anos, a robótica inteligente vem sendo uma das áreas de pesquisa que mais tem se desenvolvido [Dorigo and Caro, 1999]. Inúmeros esforços estão sendo despendidos para a descoberta de novas técnicas de controle de robôs autônomos. Neste contexto, a natureza tem inspirado, com seus seres vivos e processos naturais, muitos cientistas a criar alternativas às soluções tradicionais, como é o caso dos algoritmos evolutivos e inteligência de enxames (swarm intelligence).

Para o desenvolvimento de robôs inteligentes, os projetistas podem adotar a estratégia de implementar módulos de programas que contenham as necessárias habilidades ou características de forma gradual e incremental. Nessa abordagem, o programador tem de prever as situações que o robô deve encontrar no seu futuro ambiente de trabalho e programá-lo para agir corretamente perante elas. Este método é sujeito a falhas, pois podem surgir situações imprevistas em que o robô não saberá como se comportar, algo potencialmente perigoso no caso de robôs que convivem com seres humanos em ambientes complexos como uma residência familiar. Essas falhas de programação, se não forem interceptadas a tempo por um programa supervisor-tutor, podem ter sérias conseqüências [Simões, 2003].

Uma alternativa para o projeto de robôs inteligentes é o uso de algoritmos genéticos e técnicas de robótica evolutiva para "evoluir" esses comportamentos e características nos robôs através de seleção natural. Robótica Evolutiva é um conceito relativamente novo cunhado por D. Cliff, I. Harvey e P. Husbands [Cliff et al., 1992] e D. Floreano e F. Mondada [Floreano and Mondada, 1994] em seus experimentos com simulações de populações de robôs.

A principal diferença entre essas duas metodologias é que a primeira é uma abordagem top-down, como a Inteligência Artificial, que inicialmente procura identificar comportamentos complexos para depois tentar construir um sistema que apresente todos os detalhes desses comportamentos. Já a Robótica Evolutiva, começa com unidades elementares simples, como na Vida Artificial, para gradualmente adicionar níveis de complexidade através da evolução de novas características essenciais, de uma maneira botton-up. Enquanto a inteligência artificial tradicionalmente se concentra em funções humanas complexas, vida artificial consiste de comportamentos naturais básicos, enfatizando a sobrevivência em ambientes complexos [Dautenhahn and Nehaniv, 1998].

6. Robótica Adaptativa

Robótica Adaptativa refere-se a um conjunto de técnicas existentes para realizar o controle de robôs através de um processo de aquisição de conhecimento adquirido através da interação com o meio ambiente, gerando um modelo aproximado do mundo. Essas técnicas de controle diferem assim do controle clássico, onde um modelo dever ser previamente fornecido. Muitos destas técnicas foram desenvolvidas com base em estudos realizados sobre comportamentos de seres vivos atuando em seu ambiente natural, tais como, insetos, aranhas, peixes, etc. Certamente, o desenvolvimento de técnicas adaptativas depende muito de quanto a biologia se desenvolve, pois é daí que podem surgir idéias

para a construção de sistemas inteligentes que se comportem de algum modo de forma semelhante aos seres vivos.

Na maioria dos casos os robôs móveis são projetados de forma a interagir com um meio específico e é a sua capacidade de lidar com mudanças no meio onde atua que o caracteriza como mais ou menos apto para realizar determinadas tarefas. Para uma interação efetiva com o ambiente, o robô deve ser capaz de obter informações do mesmo.Para isto são utilizados sensores, dispositivos responsáveis pela captação de informações do ambiente, sendo que os mais comuns são: câmeras, sonares que medem a distância de possíveis obstáculos através de pulsos acústicos e sensores de varredura: faz medidas de distância a objetos a partir de radiação eletromagnética entre outros. São várias as tarefas que um robô móvel pode realizar. Entre elas encontram-se: exploração de ambientes, manipulação de objetos, mapeamento do ambiente, reconhecimento de objetos, limpeza de ambientes, coleta de objetos, etc. Neste capítulo são apresentadas algumas das técnicas adaptativas mais utilizadas no controle de robôs móveis.

6.1. Campos Potenciais

Esta técnica vem sendo muito utilizada para fazer com que robôs móveis ou manipuladores gerem trajetórias suaves em um ambiente dinâmico evitando colisões.

A idéia de imaginar forças atuando sobre um robô foi sugerida por Khatib [Khatib, 1985]. Neste método, obstáculos exercem forças repulsivas e a meta aplica uma força atrativa sobre o robô. A força resultante $\overrightarrow{F}$ é composta de uma força atrativa direcionada para a meta e forças repulsivas proveniente de obstáculos. Para cada nova posição do robô todas as forças deverão ser novamente calculadas. Com isto, é possível que o robô desvie dos obstáculos e chegue até a meta. Existem vários trabalhos que aprimoraram esta técnica como em [Krogh, 1984], [Thorpe, 1984], [Newman and Hogan, 1987]. [Krogh, 1984] aprimorou este conceito considerando a velocidade do robô na vizinhança dos obstáculos. Thorpe [Thorpe, 1984] aplicou o método de campos potenciais para planejamento *off-line*.

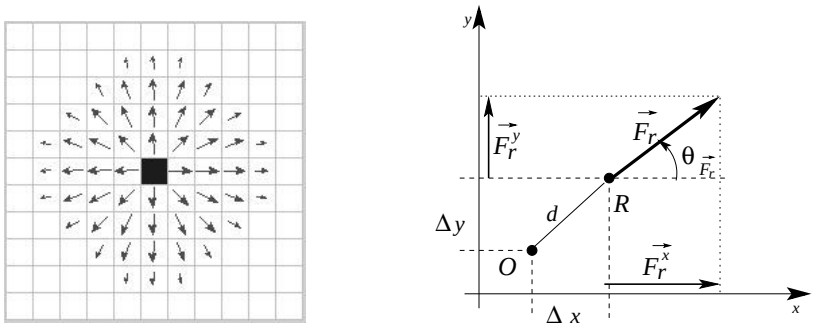
Em todos esses trabalhos foi assumido um modelo conhecido do mundo, com formas geométricas pré-definidas representando obstáculos e o caminho do robô foi gerado *off-line*. No entanto, Brooks [Brooks, 1986] e Arkin [Arkin, 1989] são os pioneiros em utilizar a técnica de campos potenciais sem utilizar informações sobre o ambiente e o cálculo do campo potencial utiliza os dados recebidos pelos sensores ultrasônicos (sonar) em cada instante de tempo. Brooks utilizou campos potenciais como um controlador reflexivo, que reage executando ações diretamente ligadas as percepções, resultando assim em respostas imediatas a estímulos externos. Arkin propôs um método similar, denominado *Motor Schema*. Este método consiste em ativar vários comportamentos simultâneos que produzem um vetor força (atração ou repulsão) como resposta. A direção a ser seguida pelo robô será o vetor resultante da soma de todos os vetores gerados. Esta técnica é fundamentada na Teoria de Esquemas que é considerada ideal para expressar funções do cérebro.

Outras abordagens utilizando campos potenciais para o planejamento de trajetórias utilizando robôs móveis pode ser encontradas em [Faria and Romero, 2000], [Pacheco and Costa, 2002] e [Gomes and Campos, 2001]. Nas subseções seguintes será mostrado como realizar os cálculos das forças de repulsão,

atração e da força resultante.

6.2. Força de Repulsão

A força de repulsão ($\overrightarrow{F_r}$) é um vetor cujo módulo é inversamente proporcional ao quadrado da distância (d) entre o robô (R) e objeto observado (O) (Figura 12(b)). Na Figura 12(a) nota-se que o módulo da força de repulsão decai a medida que o robô se afasta de um obstáculo. O vetor $\overrightarrow{F_r}$ pode ser representado por suas componentes: módulo e direção,



(a) (b)

Figura 12: (a)Força de repulsão na vizinhança de um obstáculo; (b) Força de repulsão do objeto O sobre um robô localizado em R.

dados por:

$$\left|\overrightarrow{F_r}\right| = \frac{Q}{d^2} \;\; e \;\; \theta_{\overrightarrow{F_r}} = \arctan(-\Delta y, -\Delta x) \tag{2}$$

onde Q representa uma constante de repulsão; θ é o ângulo cuja tangente é $(-\Delta y/-\Delta x)$ e os sinais negativos de Δx e Δy fornecem a direção oposta ao objeto detectado (O).Para calcular a força de repulsão para vários obstáculos, deve-se fazer o somatório dos vetores das forças de repulsão geradas, isto é, $\overrightarrow{F_R} = \sum \overrightarrow{F_r}$

6.3. Força de Atração

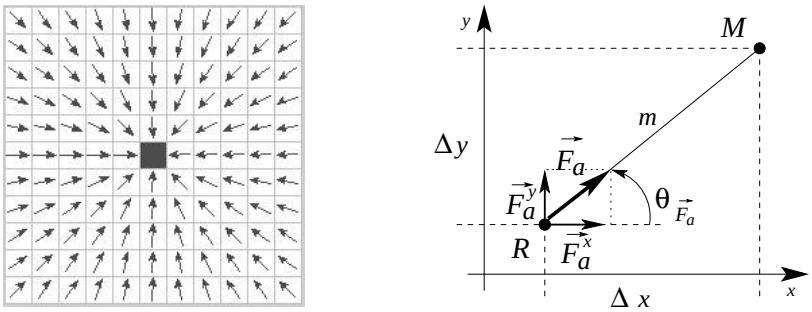
Nas Figuras 13(a) e 13(b) mostra-se o campo potencial de atração, no qual deve-se considerar $\left|\overrightarrow{F_a}\right|$ constante para que o agente seja atraído pela meta mesmo estando distante da mesma (Equação 3).

$$\left|\overrightarrow{F_a}\right| = C \;\; e \;\; \theta_{\overrightarrow{F_a}} = \arctan(\Delta y, \Delta x) \tag{3}$$

no qual C representa uma constante de atração. Note que Δx e Δy diferem da equação (2) por não estarem acompanhados do sinal negativo. Como resultado, a direção do vetor força de atração será na direção da meta.

6.4. Força Resultante

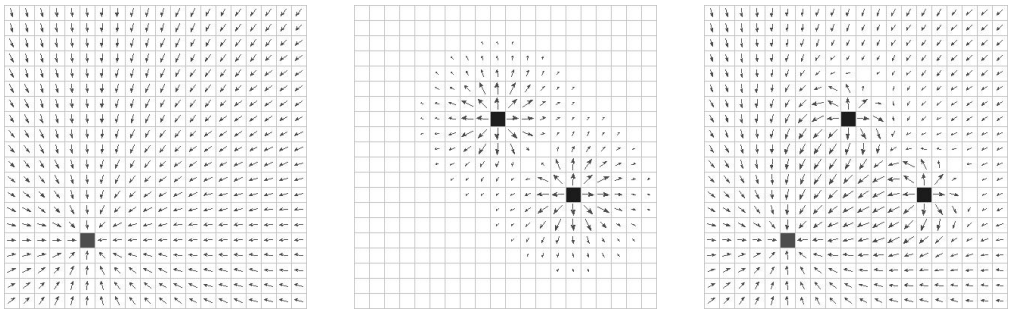
Ao realizar a soma dos vetores força de atração e força de repulsão obtém-se a força resultante dada por: $\overrightarrow{F} = \overrightarrow{F_R} + \overrightarrow{F_a}$ Para ilustrar a superposição dos campos de atração



(a) (b)

Figura 13: (a)Campo de atração constante; (b)Atração entre o robô R e a meta M.

e repulsão, é mostrado na Figura 14(a), o campo de atração gerado pela meta, na Figura 14(b), o campo de repulsão gerado por dois obstáculos e na Figura 14(c), a soma dos campos de atração e de repulsão.

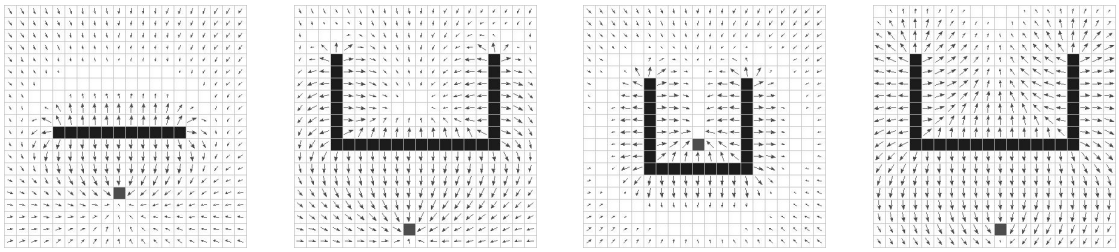


(a) (b) (c)

Figura 14: (a) potencial atrativo da meta, (b) potencial repulsivo de dois obstáculos, (c) soma destes dois campos.

Algumas vantagens desta técnica são: custo computacional baixo se comparado a métodos no qual se utilizam mapas, nenhuma modificação precisa ser feita para tratar obstáculos dinâmicos pois o cálculo é refeito para cada nova posição do robô, é um sistema modular no qual as forças podem ser calculadas em paralelo o que facilita implementações em hardware.

Como desvantagem tem-se que a força de atração e de repulsão podem se anular em determinados pontos gerando mínimos locais. Na Figura 15(a),(b) e (c) pode-se observar a presença de mínimos locais representados por células em branco. O problema de mínimos locais também pode gerar armadilhas no ambiente. Na Figura 15(b) mostra-se uma destas armadilhas na qual um robô pode ser direcionado para um local do qual não há saída. Já na Figura 15(c) o método não consegue gerar um caminho até a meta pois esta tem sua força de atração anulada pelas paredes que estão próximas. Um outro problema do método é encontrar valores para as constantes de atração (C) e de repulsão (Q) de forma a maximizar a eficiência do robô para um determinado ambiente. Isto pode ser observado se for aumentado de 4 vezes o valor de (Q) para o ambiente apresentado na Figura 15(b). Neste caso, obtém-se o campo mostrado na Figura 15(d). Nota-se que o problema da armadilha foi solucionado, mas a meta teve seu campo anulado pelo campo gerado pelos obstáculos próximos a ela.



(a) (b) (c) (d)

Figura 15: Mínimos locais gerados pela anulação das forças de atração e repulsão.

Esses problemas podem ser contornados com a abordagem proposta por Borenstein & Koren [Borenstein and Koren, 1991], conhecida como *Virtual Field Histogram (VFH)*, que utiliza um mapa para representar o ambiente com seus obstáculos [Elfes, 1989]. A partir da probabilidade de uma célula estar ocupada é gerado o campo potencial para uma dada posição do robô.A seguir será apresentado um método utilizado para tornar o robô capaz de mapear ambientes.

6.5. Grade de Ocupação

O mapeamento autônomo de ambientes é muito importante pois a tarefa de ensinar um robô se auto-localizar requer um mapa detalhado do ambiente, o que pode ser difícil de se conseguir manualmente ou por meio de plantas, uma vez que estas geralmente não contém informações sobre móveis e outros objetos pertencentes ao ambiente.

A técnica Grade de Ocupação [Elfes, 1989] usa uma representação probabilística e reticulada da informação espacial. A grade de ocupação é um campo aleatório multidimensional que mantém estimativas estocásticas dos estados das células em um espaço reticulado. Para representar este espaço utiliza-se uma matriz multidimensional (tipicamente 2D ou 3D), que mapeia o espaço em células, onde cada célula armazena uma estimativa probabilística de seu estado. Os estados possíveis são ocupado ou livre.

As estimativas do estado das células são obtidas por meio da interpretação das leituras de distâncias usando modelos probabilísticos dos sensores. Procedimentos de estimativa Bayesianos permitem a atualização incremental da grade de ocupação usando leituras realizadas por vários sensores sobre múltiplos pontos de vista.

A variável de estado $s(C)$ associada a célula C da grade de ocupação é definida como uma variável aleatória discreta com dois estados, ocupada e vazia, denotadas OCC e EMP. Dessa forma, a grade de ocupação corresponde a um campo aleatório de estados discretos e binários, uma vez que os estados das células são exclusivos e exaustivos, $P[s(C) = OCC] + P[s(C) = EMP] = 1$. Para interpretar os dados sensoriais (medidas de distâncias), é usado um modelo estocástico do sensor definido por uma função de densidade de probabilidade na forma $p(r|z)$ que relaciona a leitura r ao parâmetro de espaço z referente a uma posição do mapa.

Partindo da estimativa atual do estado da célula C_i, $P[s(C_i) = OCC|\{r\}_t]$, baseada nas observações $\{r\}_t = \{r_1, ..., r_t\}$ e dada uma nova observação r_{t+1}, a estimativa atualizada é dada por:

$$P[s(C_i) = OCC|\{r\}_{t+1}] = \frac{p[r_{t+1}|s(C_i) = OCC]P[s(C_i) = OCC|\{r\}_t]}{\sum_{s(C_i)} p[r_{t+1}|s(C_i)]P[s(C_i)|\{r\}_t]} \quad (4)$$

A estimativa anterior do estado da célula, $P[s(C_i) = OCC|\{r\}_t]$, serve como uma estimativa a priori. Essa nova estimativa é então armazenada no mapa. Dessa forma, obtém-se um mapa que representa as probabilidades de ocupação de cada célula. Uma aplicação desta técnica sem maiores transformações pode ser encontrada em [Bianchi, 2003].

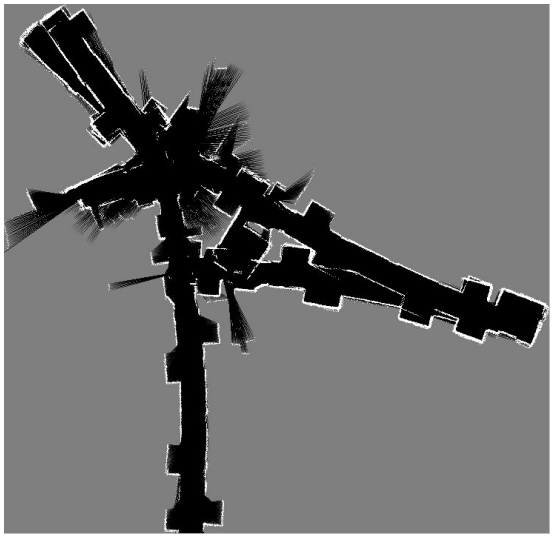


Figura 16: Mapeamento gerado ao explorar um ambiente [Bianchi, 2003].

6.6. Localização de Markov

A localização de robôs móveis é uma tarefa que consiste em estimar a posição de um robô móvel por meio de um mapa do ambiente em que o robô está inserido e informações obtidas de seus sensores.

Localização é uma técnica que consiste em estimar a posição de um robô em seu ambiente por meio de informações de seus sensores. Para isso, o robô deve ter, a priori, um mapa do ambiente, o qual pode ser construído automaticamente (pelo robô) ou manualmente (por seres humanos, usando ferramentas de CAD[2], por exemplo).

O método de localização de Markov [Fox et al., 1998] mantém múltiplas hipóteses sobre a posição do robô. Cada hipótese possui um peso associado que é um fator probabilístico numérico. A idéia principal é calcular uma distribuição de probabilidades sobre todos os possíveis locais do ambiente. Uma de suas principais implementações é conhecida como algoritmo de localização Monte Carlo [Fox et al., 1999, Thrun, 2000].

Seja $l = \langle x, y, \theta \rangle$ uma posição no espaço de estados, em que x e y são coordenadas cartesianas do robô e θ é a orientação do mesmo. A distribuição $Bel(l)$ sobre todos os locais l expressa a crença subjetiva que o robô de estar na posição l. Se a posição inicial é conhecida, $Bel(l)$ é centrada na posição correta, caso contrário, ela é uniformemente

[2]Ferramenta de projeto auxiliado por computador

distribuída para refletir essa incerteza. Assim que o robô inicia sua operação, $Bel(l)$ é refinada de modo incremental.

A Localização de Markov aplica dois modelos probabilísticos diferentes para atualizar $Bel(l)$: um modelo de ação para incorporar os movimentos do robô e um modelo perceptual para atualizar a crença baseando-se na entrada sensorial.A movimentação do robô é modelada pela probabilidade condicional $p(l \mid l', a)$ que especifica a probabilidade de que uma ação a, quando executada em l', leve o robô para l. $Bel(l)$ é então atualizada por:

$$Bel(l) \longleftarrow \sum_{l'} p(l \mid l', a) \cdot Bel(l'). \tag{5}$$

O termo $p(l \mid l', a)$ representa um modelo do movimento do robô. As leituras dos sensores são integradas de acordo com a regra de Bayes. Seja s uma leitura de sensor e $p(s \mid l)$ a probabilidade de se perceber s dado que o robô está na posição l, então $Bel(l)$ é atualizada de acordo com a regra:

$$Bel(l) \longleftarrow \alpha p(s \mid l) Bel(l), \tag{6}$$

onde α é um fator de normalização que garante que o somatório de $Bel(l)$ para todo l seja igual a 1.

Ambos os passos de atualização são aplicáveis apenas se o problema for *Markoviano*, ou seja, se as leituras de sensores passadas forem condicionalmente independentes de leituras futuras dada a localização do robô. Desse modo, a hipótese de Markov assume que o mundo é estático. Essa técnica funcionou bem até mesmo em ambientes que continham pessoas e que desta forma violavam a hipótese de Markov. No entanto, os experimentos reportados em [Fox et al., 1998] mostram que ela não se adequa bem a ambientes densamente povoados. Uma das formas de resolver esse problema consiste na aplicação de filtros que selecionam quais leituras dos sensores serão utilizadas efetivamente na localização. Existem algumas formas de se implementar a técnica de Localização de Markov. Elas diferem em três pontos: na representação da crença do robô, na atualização da crença e nos modelos probabilísticos de ação e reação.

Será descrito aqui o Algoritmo Monte Carlo (Algoritmo 6.1). Este algoritmo representa a crença $Bel(s)$ por um conjunto de amostras s associadas a um fator numérico de importância p. Esse fator indica a probabilidade da amostra ser relevante para a determinação da posição do robô. A crença inicial é obtida gerando-se aleatoriamente N amostras da distribuição prévia $P(s_o)$ e atribuindo-se o fator de importância uniforme N^{-1} para cada amostra.

Leituras de sensores o e ações a são processadas em pares. Uma nova crença é construída repetindo a seguinte seqüencia de "suposições": primeiro, um estado aleatório s é gerado a partir da crença atual, sob consideração dos fatores de importância. Para aquele estado s, um novo estado s' é suposto de acordo com o modelo de ação $P(s' \mid a, s, m)$ em que m contém os dados do mapa. O fator de importância para esse novo estado s' recebe um valor proporcional à consistência perceptual desse estado, como medido por $P(o \mid s, m)$. A nova amostra e o seu fator de importância são memorizados e o laço básico é repetido. A geração de amostras é finalizada quando a soma total dos fatores

de importância exceder um limite p_{max}, ou quando o próximo par $< o, a >$ chega. Finalmente, os fatores de importância são normalizados e o método de Monte Carlo retorna o novo conjunto de amostras definido como a crença corrente.

Algoritmo 6.1 Algoritmo Monte Carlo(S,a,o) [Thrun, 2000].

```
1: S' = 0
2: p_sum = 0
3: enquanto p_sum < p_max faça
4:    crie uma amostra aleatória ¡ s, p ¿ de S de acordo com p_1, ..., p_N
5:    gere um s' aleatório de acordo com P(s' | a, s,m)
6:    p' = P(o | s',m)
7:    adicione ( s', p' ) a S'
8:    p_sum = p_sum + p'
9: fim-enquanto
10: normalize os fatores de importância p em S'
11: retorne S'
```

Uma desvantagem desse método é que ele possui uma tendência de descartar hipóteses corretas e pode não se recuperar de falhas de localização global (não resolvendo o problema do robô sequestrado). Isto pode ser resolvido adicionando-se amostras aleatórias, matematicamente justificadas, assumindo que o robô pode ser sequestrado com uma pequena probabilidade. Ele possui uma série de vantagens: é extremamente eficiente, pois a computação tem seu foco em regiões de alta probabilidade, é robusto a ruídos e a mudanças que afetem a disponibilidade de recursos computacionais.

Uma implementação desta técnica pode ser encontrada em [Bianchi, 2003], na qual foi assumido que os erros de odometria são uniformemente distribuídos. Nesta implementação a localização e o mapeamento são realizados de forma combinada. Primeiro, o robô se localiza e em seguida mapeia o ambiente e este processo é repetido até que o ambiente esteja totalmente mapeado.

7. Aplicações

Nesta seção serão apresentadas algumas aplicações de Computação Bioinspirada. Para organizar estas aplicações, elas foram divididas em dois grandes grupos: aplicações em Bioinformática e Aplicações em Engenharia.

7.1. Bioinformática

Existe um grande número de aplicações de técnicas de Computação Convencional e Computação Biológica em problemas de Bioinformática ou Biologia Computacional. Nesta seção serão apresentadas algumas destas aplicações.

7.1.1. Reconhecimento de Genes

A identificação de genes em seqüências de DNA constitui uma tarefa muito custosa se realizada por meios laboratoriais. A obtenção de um procedimento (algoritmo) que automatize esta tarefa também é inviável, pois o conhecimento acerca da natureza dos genes

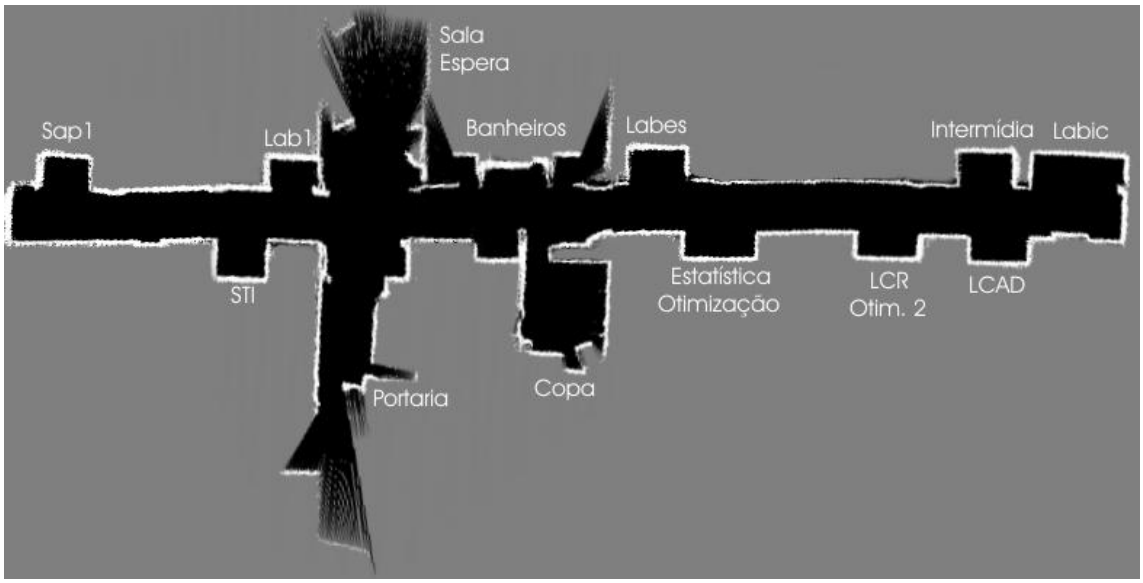


Figura 17: Grid de Ocupação gerado ao explorar um ambiente [Bianchi, 2003].

ainda é incompleto. Há muitas variações na forma como os genes se apresentam, o que torna seu reconhecimento um problema complexo.

RNs têm sido largamente utilizadas nesta aplicação [Craven and Shavlik, 1994, Pavlidis et al., 2001, Shavlik, 1991, Uberbacher et al., 1993, Xu et al., 1996]. No reconhecimento de genes por meios computacionais, duas abordagens de busca são usualmente empregadas: por sinal e por conteúdo [Craven and Shavlik, 1994]. Essas abordagens diferem nas características das seqüências em que se concentram. A **busca por sinal** envolve a localização de sítios presentes na molécula de DNA que participam do processo de expressão gênica. Na **busca por conteúdo** procura-se padrões nas seqüências genômicas que indiquem a presença de um gene. Para obter melhores resultados, essas duas abordagens são usualmente aplicadas em conjunto.

7.1.2. Análise de Expressão Gênica

A análise da expressão dos genes é de grande interesse para as Ciências Biológicas. Esse tipo de análise pode fornecer informações importantes sobre as funções de uma célula, uma vez que as mudanças na fisiologia de um organismo são geralmente acompanhadas por mudanças nos padrões de expressão de seus genes.

Existem trabalhos que utilizam tanto técnicas não supervisionadas como técnicas supervisionadas para a análise de expressão gênica. Enquanto as técnicas não supervisionados determinam como um conjunto de genes se organiza em grupos funcionais, técnicas supervisionadas determinam que características da expressão de um conjunto de genes define a classe de um dado tecido.

7.2. Filogenia

Um dos principais problemas da Biologia é tentar explicar o processo evolutivo das espécies existentes e a forma que essas espécies relacionam-se em termos de ancestrais

comuns. A essa tentativa de descrever a diversidade biológica existente dá-se o nome de filogenia ou reconstrução de árvores filogenéticas. O conceito de Filogenia surgiu com Darwin, a partir do conceito de ancestralidade entre espécies, sendo dele o primeiro diagrama publicado representando relações filogenéticas. Assim, filogenias nada mais são que a indicação das relações de ancestralidade suposta para um conjunto de espécies.

Um problema fundamental em filogenia consiste no fato das espécies ancestrais que existiram no passado, não poderem ser observadas diretamente. Assim, é necessário buscar mecanismos para, analisando os organismos atuais, recuperar informações a respeito das relações de parentesco com os organismos ancestrais hipotéticos. Neste sentido, as técnicas filogenéticas buscam determinar os ancestrais hipotéticos que melhor representam um processo evolutivo que explique as espécies existentes hoje. Os Algoritmos Evolutivos têm mostrado resultados significativos em filogenia.

7.3. Multialinhamento de Seqüências

Dentro da bioinformática uma das atividades mais realizadas é o multialinhamento de seqüências biológicas. Seus resultados são utilizados em várias atividades que desdobram-se em áreas de pesquisa interdisciplinares com geração de diversos subprodutos. Sendo uma das primeiras etapas de tais tarefas, o multialinhamento é então importante para garantir a qualidade dos resultados obtidos em vários estudos do material genético. Para esse problemas, têm sido desenvolvido Algoritmos Evolutivos para multialinhamento global de seqüências biológicas com a obtenção de resultados relevantes.

7.4. Estruturas de Proteínas

As proteínas são macromoléculas compostas por vinte tipos diferentes de aminoácidos. Essas moléculas dobram-se formando estruturas tridimensionais, conhecidas como Estruturas Terciárias, as quais determinam as funções das proteínas. A partir da modelagem da estrutura de uma proteína pode-se, por exemplo, auxiliar o processo laboratorial para o desenvolvimento de fármacos. O processo de modelagem de proteínas consiste em determinar a estrutura tridimensional de uma proteína. Em geral, utilizam-se processos experimentais como cristalografia por raio-X e ressonância magnética para obter tais estruturas.

Os processos experimentais são em geral demorados, caros e não podem ser empregados para quaisquer proteínas. Buscando reduzir essas dificuldades experimentais, tem sido investigado o desenvolvimento de técnicas computacionais para determinar a estrutura tridimensional das proteínas. Abordagens baseadas em computação evolutiva têm apresentado resultados motivadores na busca por soluções computacionais para este problema.

7.5. Projeto de Redes

O projeto de redes envolve problemas clássicos modelados por grafos como os de árvore gerada mínima, ou *Minimum Spanning Tree (MST)*; de caminho mínimo, ou *Shortest Path (SP)*; e o do caixeiro viajante, ou *Traveling Salesman (TS)*. No mundo real, muitos problemas modelados por grafos não são exatamente iguais, mas similares aos problemas clássicos. Em geral, esses problemas são denominados de extensões. Para a MST,

algumas extensões estão formalmente definidas na literatura como, por exemplo, o *constrained MST*, o *multicriteria MST*, o *generalized MST* e outros. As extensões são geralmente *NP-Hard*. Conseqüentemente, os problemas de projeto de redes envolvendo tais extensões são computacionalmente complexos.

Entretanto, muitos desses problemas estão presentes no mundo real e, portanto, soluções adequadas precisam ser encontradas. Nesse sentido, técnicas alternativas começaram a ser investigadas. Dentre essas, destacam-se as abordagens baseadas em computação evolutiva.

7.6. Problemas Multiobjetivos

Problemas do mundo real freqüentemente envolvem vários objetivos, em geral não-lineares, e restrições. Os métodos clássicos para solução desses problemas, como técnicas de programação linear e não-linear e algoritmos de busca heurística, apresentam dificuldades para obter soluções adequadas se o número de variáveis, objetivos, restrições ou não linearidades são grandes. Algoritmos Evolutivos MultiObjetivos (AEMO) têm sido desenvolvidos para lidar com tais problemas. Esses algoritmos têm mostrado serem capazes de resolver problemas considerados relativamente complexos.

7.7. Engenharia

Existe uma sensibilidade crescente da comunidade científica nacional para o incentivo a pesquisas aplicadas, que gerem novas tecnologias. Tem sido largamente observado que vários dos problemas brasileiros podem ser resolvidos ou minimizados com o desenvolvimento da tecnologia adequada para tratá-los. Um exemplo destes problemas é o tratamento de água e esgoto, inexistente ou ineficiente para grande parte da população brasileira. Outro exemplo é a otimização do sistema de geração e distribuição de energia elétrica, necessária para lidar com a crescente demanda existente no país.

Técnicas de CB têm sido largamente utilizadas para a resolução de problemas de engenharia, tanto por empresas como por instituições de pesquisa. Os resultados obtidos têm comprovado a utilidade desta técnica para a solução destes problemas. A seguir serão apresentados algumas das aplicações onde técnicas de CB têm sido utilizadas com sucesso.

Detecção de Intrusão Hoje em dia, informação é um dos bens mais valiosos. O grande crescimento, tanto em número como em tamanho, das redes de computadores traz junto preocupações sobre a segurança dos sistemas conectados e das informações armazenados.

A identificação do comportamento intrusivo tem sido realizado de forma bem sucedida por RNs. A utilização de RNs adiciona adaptabilidade ao sistema, permitindo o acompanhamento das modificações das técnicas de ataque. Em [Junior et al., 1998], uma RN do tipo MLP aprendeu a distinguir seqüências de operações, ou assinaturas de ataque.

Classificação de Solos Tropicais Alguns solos tropicais, principalmente os lateríticos, são mais adequados para utilização na construção viária, principalmente para rodovias com baixo volume de tráfego. Grande parte do solo brasileiro é recoberta por solos lateríticos ou de comportamento laterítico.

A classificação de solos geralmente requer a realização de um grande número de ensaios, são muito demorados e precisam de técnicos e laboratórios especializados. RNs têm sido utilizadas para a classificação de amostras de solo [Rodgher et al., 1997], reduzindo a necessidade de técnicos e laboratórios especializados.

Tratamento de Esgoto O mau funcionamento de uma estação de tratamento de água e esgoto pode acarretar problemas tanto do ponto de vista social quanto do ponto de vista biológico. Esgoto tratada de forma inadequada pode gerar conseqüências desagradáveis para seres humanos e para o meio ambiente.

A operação de uma estação de tratamento de esgoto envolve processos dinâmicos, que não podem ser descritos explicitamente por métodos matemáticos, ou podem ser tratados apenas através de exaustivos esforços. RNs têm sido utilizadas com sucesso para prever parâmetros químicos e biológicos de uma amostra de esgoto, o que caracteriza previsão de séries temporais [Hanisch et al., 1998].

Restabelecimento de Energia em Sistemas de Distribuição Quando uma falha ocorre em um sistema de distribuição de energia elétrica, a zona da falta é isolada e os operadores do sistema precisam encontrar um plano apropriado para o restabelecimento de energia na área desenergizada. O restabelecimento ótimo de energia após o local de uma falha ser identificado e isolado é uma área de pesquisa que tem despertado mais crescente interesse.

Em [Delbem et al., 2003], AGs são utilizados para o restabelecimento de energia em sistemas de distribuição. Para avaliar o desempenho dos Algoritmos Genéticos, foi utilizado um sistema de distribuição simplificado, baseado no sistema de distribuição da cidade de São Carlos.

Planejamento de Operação de Sistemas Hidrotérmicos O custo de geração de energia é função dos custos de geração de energia térmoelétrica e de energia hidroelétrica. Como o segundo custo é bem menor que o primeiro, especialmente no Brasil, dadas as suas condições geográficas, procura-se aumentar a quantidade de energia gerada por hidroelétricas para minimizar o custo total. Para tal, é necessário definir o volume ótimo do(s) de uma ou mais usinas durante um certo período de tempo. Este é um problema de otimização que foi tratado utilizando Algoritmos Genéticos.

Leite et al [Leite et al., 2002] investigam a adequação da técnica de Algoritmos Genéticos para solução do problema de planejamento da operação de sistemas hidroelétrico de potência. Neste trabalho, Cada indivíduo representa uma combinação de volumes do reservatório de uma ou mais usinas durante um número N de meses. No caso de apenas uma usina, o indivíduo apresenta N genes, um para cada mês.

7.7.1. Outras Aplicações

Otimização baseada em colônias de formigas têm sido utilizadas para resolver problemas de roteamento de mensagens em sistemas de telecomunicações.

Computação Baseada em DNA foi empregada com sucesso para lidar com problemas de segurança, mais especificamente criptografia.

Dispositivos baseados em sensores, como línguas e narizes eletrônicos baseados em RNs têm sido investigados.

7.7.2. Uma Colônia Artificial de Robôs-Formiga

Ao se ter robôs inspirados em formigas, pode-se conseguir um comportamento de exploração com algoritmos de controle mais simples e mais econômicos em termos de hardware. Com robôs mais simples e, por conseqüência, de menor custo, pode-se utilizar um número maior de robôs e se obter uma solução mais tolerante a falhas, já que a perda de um robô não fará tanta diferença no cumprimento da tarefa [Cao et al., 1997].

Outra característica importante a ser observada é a utilização de Algoritmos Genéticos para evoluir os comportamentos das formigas nos robôs. Assim, é produzido um sistema híbrido aplicado a robôs reais, onde algoritmos inspirados no comportamento de formigas são evoluídos por um Algoritmo Genético (ver Figura 18). O resultado é um sistema de controle de navegação de robôs móveis autônomos distribuído e descentralizado modelado a partir de observações realizadas em campo do comportamento de formigas naturais.

Este sistema é uma colônia artificial de formigas onde vários comportamentos biológicos foram reproduzidos e um algoritmo evolutivo é utilizado para selecionar as funções comportamentais mais adequadas e configurar seus parâmetros. Assim, obtém-se um sistema de controle onde os robôs-formiga permanecem próximos uns dos outros e ao mesmo tempo exploram o ambiente para encontrar objetos específicos, trazendo-os para o grupo (“colônia”) sem comunicação global ou conhecimento global do ambiente.

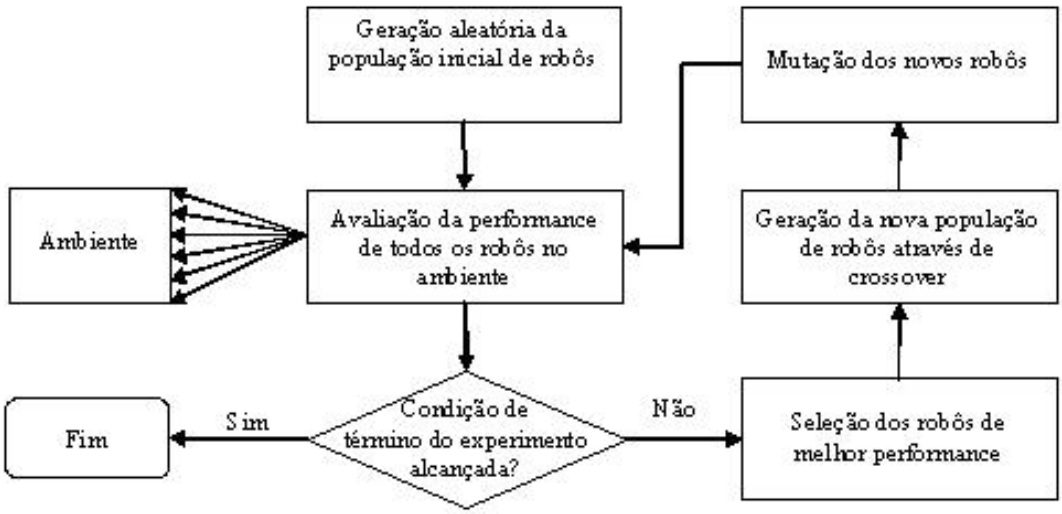


Figura 18: Fluxograma do Sistema Evolutivo Implementado.

Os comportamentos dos robôs-formiga dependem da situação onde se encontram os indivíduos, de seus estados e das funções comportamentais selecionadas em seus respectivos cromossomos. Estados são regras que proporcionam uma orientação ao mo-

vimento (características de autômatos celulares [Oliveira, 2000]) que todos os robôs-formiga possuem, diferentemente das funções que são selecionadas pelos cromossomos. E cada um desses estados apresenta um valor respectivo (uma variável) que funciona como um limiar, ou seja, quando um valor de um estado se torna maior que o outro, a ”formiga”muda de estado e passa a seguir as regras do novo estado em que se encontra. Existem funções que modificam essa troca de estados, aumentando ou diminuindo seus valores, proporcionando uma mudança de estado diferente para cada formiga, dependendo de seu cromossomo. Os estados que cada indivíduo pode assumir são:

1. TRISTE - neste estado, a formiga procura seguir as trilhas de feromônios das outras, indo de encontro a outras formigas.
2. FELIZ - neste estado, a ”formiga”procura afastar-se dos feromônios colocados pelas outras, tendendo a explorar o meio.

Cada robô-formiga apresenta diferentes tipos de comportamentos, definidos pelo seu cromossomo. Os cromossomos, por sua vez, possuem 15 ”genes”que caracterizam a presença ou não de funções ou aumento da intensidade destas (o cromossomo é um vetor Booleano).

7.7.3. Um Sistema Evolutivo no Projeto de uma Colônia de Robôs-Formiga

No sistema evolutivo, apresentado na Figura 18, a população inicial é formada a partir da criação de indivíduos com cromossomos gerados aleatoriamente. Em seguida, os indivíduos são avaliados. Cada indivíduo recebe uma nota de aptidão (fitness). A seguir, é verificado se a condição de término do experimento foi alcançada. Esta condição pode ser, por exemplo, se todos os indivíduos estão se comportando de forma satisfatória. Caso a condição ainda não tenha sido satisfeita, o próximo passo é a seleção de indivíduos para o cruzamento genético. O método utilizado é o elitismo [Bramlette, 1991]. Após a seleção, é utilizado o crossover para gerar a nova população, onde cada ”gene”do cromossomo tem uma chance de 50% de vir do pai e 50% de vir da mãe. Os novos indivíduos podem vir a sofrer mutações em seus genes, o que pode vir a contribuir para aumentar a diversidade genética. Os robôs-formiga se movimentam pelo seu ambiente e de acordo com suas atitudes, eles são avaliados pela integração das funções de exploração, sociável e alimentação, descritas a seguir.

Função de Exploração Ao iniciar a simulação, cada indivíduo é avaliado isoladamente. É gerado um círculo de raio R em volta da “formiga” e esta se move por conta própria dentro dele, perdendo 1 ponto a cada iteração, até ultrapassar o círculo. Neste momento, ela recebe 10 pontos e, em seguida, é gerado um novo círculo de raio R em volta do indivíduo, reiniciando o processo (ver Figura 19).

Função Sociável A cada iteração as “formigas” fazem um círculo em volta de si mesmas e contam o numero de outras “formigas” que se encontram dentro do círculo. Deste modo, a função sociável aumenta sua pontuação em 4 pontos por “formiga” presente no círculo. A Figura 20 exemplifica esta função.

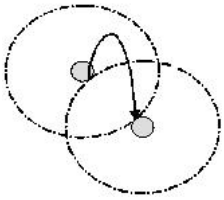


Figura 19: Diagrama representando o círculo de pontuação da Função Exploração.

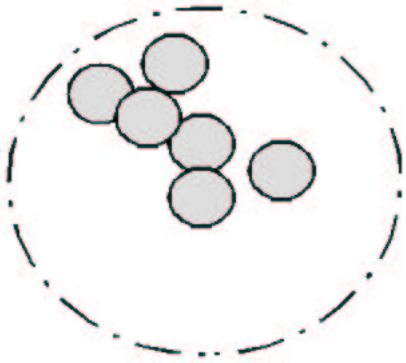


Figura 20: Diagrama representando o círculo de pontuação da Função Sociável.

Função Alimentação O robô recebe um bônus (prêmio) da função de avaliação por encontrar ”comida”(objetos de interesse da missão corrente) igual a 500 pontos. Porém, o robô que encontrar a comida começará a perder pontos (-10 por iteração) até o momento em que encontrar pelo menos outros três robôs e dividir com eles sua ”comida”, valorizando a volta para a colônia ou a exploração conjunta, onde ganha 5000 pontos.

Função de Avaliação Global A Função de Avaliação Global é a soma das três funções anteriores (Sociável, Exploração e Alimentação), valorizando os robôs-formiga que apresentarem desenvolvimento com todas as funcionalidades nestas três áreas, ou seja, valorizando os robôs multi-especialistas.

Como cada função interfere indiretamente ou diretamente nas outras, o sistema possui um número grande de possíveis soluções e precisa controlar e avaliar a relação entre elas. Assim, um Sistema Evolutivo (selecionando as melhores relações e descartando as outras) é utilizado para calcular a melhor combinação entre as funções para que o grupo de robôs-formiga realizem as tarefas estipuladas.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPESP e CNPq.

Referências

Adleman, L. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024.

Ahmadabadi, M. and Ghaderi, F. (2004). Distributed cooperative load redistribution for fault tolerance in a team of four object-lifting robots. *Advanced Robotics, Publisher: VSP BV, Netherlands*, pages 43–51.

Arkin, R. C. (1989). Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 4(8):92–112.

Benenson, Y., Gil, B., Ben-Dor, U., and Adar, R. (2004). An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429.

Benenson, Y., Paz-Elizur, T., Adar, R., et al. (2001). Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434.

Bianchi, R. E. (2003). Sistema de navegação de robôs móveis autônomos para o transporte de documentos. Master's thesis, ICMC – USP.

Boneh, D., Dunworth, C., Lipton, R., and Sgall, J. (1996). On the computational power of DNA. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 71.

Boneh, D., Dunworth, C., Lipton, R. J., and Sgall, J. (1999). Making DNA computers error resistant. In Landweber, L. and Baum, E., editors, *DNA Based Computers*, volume 44 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–170. AMS.

Borenstein, J. and Koren, Y. (1991). The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288.

Braga, A., de Carvalho, A., and Ludermir, T. (2000). *Redes Neurais Artificiais: Teoria e Aplicações*. Editora Livro Técnico e Científico.

Bramlette, M. (1991). Initialization, mutation and selection methods in genetic algorithms for function optimization. *Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann*, pages 100–107.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Cao, Y., Fukunaga, A., and Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:1–23.

Cliff, D., Harvey, I., and Husbands, P. (1992). Incremental evolution of neural network architectures for adaptive behaviour. *Report ID: Cognitive Science Research Paper Serial No. CSRP 256, The University of Sussex School of Cognitive and Computing Sciences, Falmer Brighton, UK*, pages 1–15.

Cliff, D. and Miller, G. F. (1996). Co-evolution of pursuit and evasion ii: Simulation methods and results. *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior - SAB96: From Animals to Animats 4, Maes, P., Mataric, M., Meyer, J. A. et. al. (Eds.), Publisher: MIT Press*, pages 506–515.

Craven, M. W. and Shavlik, J. W. (1994). Machine learning approaches to gene recognition. *IEEE Expert*, 9(2):2–10.

D. Boneh, C. Dunworth, R. L. (1995). Breaking DES using a molecular computer. Technical Report CS-TR-489-95, Princeton University.

Dautenhahn, K. and Nehaniv, C. (1998). Artificial life and natural stories. *Proceedings of the Third International Symposium on Artificial Life and Robotics - AROB III'98, v. 2, January 19-21, 1998, Beppu, Japan*, pages 435–439.

Delbem, A. C. B., Bretas, N. G., and de Carvalho, A. (2003). Optimal energy restoration in radial distribution systems using a genetic approach and graph chain representation. *Electric Power Systems Research Journal*, 67(3):197–205.

Dorigo, M. (2001). Ant algorithms and swarm intelligence. *Proc. of the Seventenn Int. Joint Conf. on Artificial Intelligence, Tutorial MP-1*.

Dorigo, M. and Caro, G. (1999). The ant colony optimization meta-heuristic. *New Ideas in Optimization*, pages 11–32.

Dorigo, M., Caro, G., and Gambardella, L. M. (1999). Ant algortihms for discrete optimization. *Artificial Life,*, 5(3):137–172.

Dorigo, M. and Gambardella, L. (1997). Ant colony system: a ccoperative learning approach to the travelin salesman problem. *IEEE Trans. on Evolutionary Computation*, 1(1).

Elfes, A. (1989). *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University.

Faria, G. and Romero, R. A. F. (2000). Incorporating fuzzy logic to reinforcement learning. In *Proceedings of the 9th IEEE International Conference on Fuzzy Systems*, volume 1, pages 847–851.

Faulhammer, D., Cukras, A., Lipton, R., and Landweber, L. (1997). Molecular computation: RNA solutions to chess problems. *Proceedings of the National Academy of Sciences*, 97(4):1385–1389.

Floreano, D. and Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior - SAB'94, From Animals to Animats 3, Cliff, D., Husbands, P., Meyer, J. A. et. al. (Eds.), Publisher: MIT Press/Bradford Books, Cambridge, MA, USA*, pages 421–430.

Floreano, D. and Mondada, F. (1995). Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous Systems*, 16:183–195.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1996). *Artificial Intelligence through Simulated Evolution*. Wiley.

Fox, D., Burgard, W., and Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427.

Fox, D., Burgard, W., Thrun, S., and Cremers, A. B. (1998). Position estimation for mobile robots in dynamic environments. *AAAI/IAAI*, pages 983–988.

Gambardella, L. and Dorigo, M. (1995). Ant-q: a reinforcement learning approach to the traveling salesman problem. *Proc. of the ML-95 - Twelfth Int. Conf. on Machine Learning*, pages 252–260.

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

Goldberg, D. E. and Holland, J. H. (1988). *Genetic Algorithms and Machine Learning: Introduction to the Special Issue on Genetic Algorithms*. Machine Learning.

Gomes, M. R. S. and Campos, M. F. M. (2001). Desvio de obstáculos para micro-robôs móveis utilizando campo potencial. In *Anais do V Simpósio Brasileiro de Automação Inteligente (SBAI)*, Canela/RS.

Hanisch, W., de Carvalho, A., and Pires, E. (1998). A neural network model to predict parameters of a wastewater treatment plant. In *III International Conference on Hydroinformatics*, pages 176–187, Copenhague, Dinamarca.

Haykin, S. (1999). *Neural Network, a comprehensive foundation*. Prentice Hall, New Jersey, USA.

Hearst, M. A., Schölkopf, B., Dumais, S., Osuna, E., and Platt, J. (1998). Trends and controversies - support vector machines. *IEEE Intelligent Systems*, 13(4):18–28.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life - ECAL95, LNAI 929, Moran et. al. (Eds.), Publisher: Springer Verlag*, pages 704–720.

Jong, K. D., Fogel, D. B., and Schwefel, H.-P. (2000). A history of evolutionary computation. *Evolutionary Computation1: Basic Algorithms and Operators*, pages 40–58.

Junior, J. B., Cansian, A., Moreira, E., and de Carvalho, A. (1998). An adaptive intrusion detection system using neural networks. In *Proceedings of the IFIP World Computer Congress - Security in Information Systems, IFIP-SEC'98*, page 13 páginas em CD, Viena, Austria. IFIP.

Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 500–505, St. Louis, Missouri.

Kohonen, T. (1995). *Self-Organising Maps*. Springer Verlag.

Krogh, B. H. (1984). A generalized potential field approach to obstacle avoidance. In *International Robotics Research Conference*, Bethlehem, Pennsylvania.

Lander, E., Linton, L., and Birren, B. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921.

Leite, P. T., Carneiro, A. A. F. M., and Carvalho, A. C. P. L. F. (2002). Energetic operation planning using genetic algorithms. *IEEE Trasaction on Power Systems*, 17(1):173–179.

Lesk, A. (2002). *Introduction to Bioinformatics*. Oxford University Press.

Lipton, R. (1995). Using DNA to solve NP-complete problems. Technical Report NJ 08540, Princeton University, Princeton.

Mao, C., LaBean, T., Reif, J., and Seeman, N. (2000). Logical computation using algorithmic self assembly of dna triple-crossover molecules. *Nature*, 407:493–496.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press Massachusetts.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Nelson, D. and Cox, M. (2000). *Lehninger Principles of Biochemistry*. Macmillan Press/Worth Publishers, 3 edition.

Newman, W. S. and Hogan, N. (1987). High speed robot control and obstacle avoidance. In *Proceedings of the 1987 IEEE International*, pages 14–24, Raleigh, North Carolina.

Oliveira, G. (2000). Autômatos celulares: Aspectos dinâmicos e computacionais. *III Jornada de Mini-cursos de Inteligência Artificial*, pages 297–339.

Pacheco, R. N. and Costa, A. H. R. (2002). Navegação de robôs móveis utilizando o método de campos potenciais. In Sakude, M. T. S. and de A. Castro Cesar, C., editors, *Workshop de Computação WORKCOMP'2002*, pages 125–130. SBC, ITA – São José dos Campos/SP.

Parker, L. E. (1993). An experiment in mobile robotic cooperation. *Proceedings of Robotics for Challenging Environments*, pages 090–096.

Pavlidis, P., Furey, T., Liberto, M., Haussler, D., and Grundy, W. (2001). Promoter region-based classification of genes. In *Proc. of the Pacific Symposium on Biocomputing*, pages 151–163.

Pevzner, P. (2000). *Computational Molecular Biology: An algorithmic approach*. MIT Press.

Rampone, S. (1998). Splice-junction recognition on gene sequences (DNA) by BRAIN learning algorithm. In *Neural Networks Proceedings*, volume 1, pages 774–779. IEEE World Congress on Computational Intelligence.

Ridley, M. (1996). *Evolution*. Blackwell Science.

Rodgher, S., Fabbri, G., and de Carvalho, A. (1997). A utilização de redes neurais artificiais para classificação de solos tropicais. In *XI Congresso de Ensino e Pesquisa em Transportes, XI ANPET*, pages 176–187, Rio de Janeiro, RJ. ANPET.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Books.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Parallel Distributed Processing*, volume 1: Foundations, chapter Learning internal representations by error propagation, pages 318–362. The MIT Press.

Setubal, J. and Meidanis, J. (1997). *Introduction to computational molecular biology*. PWS Publishing Co.

Shavlik, J. W. (1991). Finding genes by case-based reasoning in the presence of noisy case boundaries. *Proc. of the DARPA Cased-Based Reasoning Workshop*, pages 327–338.

Simões, E. (2000). Development of an embedded evolutionary controller to enable collision-free navigation of a population of autonomous mobile robots. *PhD Thesis, The University of Kent at Canterbury, UK*, pages 1–289.

Simões, E. (2003). Robótica evolutiva. *Sociedades Artificiais: A nova fronteira das máquinas. Editora ARTMED, Porto Alegre-RS*, pages 251–275.

Simões, E. and Dimond, K. (1999). Evolution of neural control structures: Some experiments on mobile robots. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 6:596–601.

Smith, T. (1998). Blurred vision: Simulation-reality transfer of a visually guided robot. *Proceedings of the First European Workshop on Evolutionary Robotics - EvoRobot98, Husbands, P. and Meyer, J. (Eds.), Publisher: Springer Verlag*, pages 123–136.

Thorpe, C. (1984). Path relaxation: Path planning for a mobile robot. Technical Report CMU-RI-TR-84-05, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Thrun, S. (2000). Probabilistic algorithms in robotics. Technical Report CMU–CS–126, Carnegie Mellon University.

Tomassini, M. (1995). A survey of genetic algorithms. *Annual Reviews of Computational Physics, World Scientific*, pages 87–118.

Towell, G., Shavlik, J., and Noordewier, M. (1990). Refinment of approximate domain theories by knowledge-based neural networks. In *Proc. of the National Conference on Artificial Intelligence*, pages 861–866. AAAI Press.

Uberbacher, E. C., Einstein, J. R., Guan, X., and Mural, R. J. (1993). Gene recognition and assembly in the GRAIL system: Progress and challenges. *Proc. of the International Conference on Bioinformatics, Supercomputing and Complex Genome Analysis*, pages 465–476. World Scientific, Singapura.

Vittori, K. (2004). *Aprendizagem-Q e Comportamento de Colônias de Formigas para roteamento em redes de telecomunicações*. PhD thesis, Departamento de Engenharia Elétrica, Escola de Engenharia de São Carlos, Universidade de São Paulo. Qualificação de Doutorado.

Watson, R., Ficici, S., and Pollack, J. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. *Proceedings of the Congress on Evolutionary Computation, Angeline, P., Michalewicz, Z., Schoenauer, M. et. al. (Eds.), Publisher: IEEE Press*, pages 335–342.

Widrow, G. and Hoff, M. E. (1960). Adaptative switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention*.

Winfree, E., Yang, X., and Seeman, N. (1996). Universal computation via self-assembly of dna: Some theory and experiments. In Lipton, R. and Landweber, L., editors, *Proceedings of the Second Annual Meeting on DNA Based Computers - DIMACS Workshop*, Princeton, NJ.

Xu, Y., Mural, R., Einstein, J., Shah, M., and Uberbacher, E. (1996). GRAIL: A multiagent neural network system for gene identification. In *Proceedings of the IEEE*, volume 84, pages 1544–1552.

Zuben, F. J. V. (2000). Computação evolutiva: Uma abordagem pragmática. *Anais da* 1^a *Jornada de Estudos em Computação de Piracicaba e Região (*1^a *JECOMP)*, 1:25–45.