CEZAR AUGUSTO SIERAKOWSKI

Inteligência Coletiva Aplicada a Problemas de Robótica Móvel

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

CURITIBA

2006

CEZAR AUGUSTO SIERAKOWSKI

Inteligência Coletiva Aplicada a Problemas de Robótica Móvel

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

Área de Concentração: *Automação e Controle de Processos*

Orientador: Prof. Dr. Leandro dos Santos Coelho

CURITIBA

2006

Sierakowski, Cezar Augusto
S572i Inteligência coletiva aplicada a problemas de robótica móvel / Cezar
2006 Augusto Sierakowski ; orientador, Leandro dos Santos Coelho. – 2006.
xix, 138 p. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2006
Inclui bibliografia

1. Robôs móveis. 2. Inteligência coletiva. 3. Algoritmo genético. 3. Controladores programáveis. I. Coelho, Leandro dos Santos. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Engenharia de Produção e Sistemas. III. Título.

CDD-20.ed. 629.892
511.8

"Se vi mais longe foi porque me apoiei no ombro de gigantes."
Isaac Newton (1642 - 1727)

Dedico esta dissertação à Vanessa,

Por seu inestimável apoio, enorme carinho
na minha busca por novos horizontes,
e paciência inesgotável em tantas ocasiões em que
tive que me dedicar inteiramente ao trabalho.
Eu tive a sorte de encontrá-la.

Agradecimentos

Eu gostaria de agradecer aos que contribuíram para que eu concluísse esta minha dissertação de mestrado:

Ao meu orientador ao longo desta dissertação Leandro S. Coelho, principal fonte de conhecimento, orientação e supervisão desta dissertação e por acreditar tanto neste trabalho.

Aos senhores Júlio César Nievola, Renato Krohling, Fernando Von Zuben e Heitor pelos comentários que foram de grande valia para a minha conclusão desta pesquisa.

À Pontifícia Universidade Católica do Paraná por me fornecer o ambiente físico e os recursos tecnológicos necessários para a realização desta pesquisa.

À Positivo Informática® e ao CNPq pelo suporte financeiro conjunto que me permitiu realizar este trabalho.

Por fim, agradeço à minha família pelo auxílio nos momentos difíceis.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

$[x_1 x_2 ... x_n]$	Vetor composto pelos valores x_1, x_2 até x_n.
$\sum_{i=1}^{n} x_i$	Somatório dos valores de x_1 até x_n.
[*min*, *max*]	Limites mínimo e máximo possíveis para uma determinada variável.
rand	Número aleatório com distribuição uniforme
$randn(\mu, \sigma)$	Número aleatório com distribuição gaussiana com média μ e desvio padrão σ.
$\Re^p$	Espaço dos números reais com dimensão *p*.
exp(x)	Função exponencial de *x*.
μ(x)	Função de pertinência de *x*.
^	Operador de mínimo.

Lista de Abreviaturas

AG	Algoritmo Genético
IC	Inteligência Computacional
PSO	*Particle Swarm Optimization*
SAM	*Standard Additive Model*
TSK	Takagi-Sugeno-Kang
TSP	*Traveling Salesman Problem*
ACO	*Ant Colony Optimization*
HVAC	*Heating, Ventilating and Air Conditioning*
CPSO	*Charged Particle Swarm Optimization*
RNA	Redes Neurais Artificiais
NASA	*National Aeronautics and Space Administration*
AS	*Ant System*
SIA	Sistema Imunológico Artificial
NB	*Negative Big* – Negativo Grande
NM	*Negative Medium* – Negativo Médio
PM	*Positive Medium* – Positivo Médio
PB	*Positive Big* – Positivo Grande

Resumo

A presente dissertação abrange os conceitos de algoritmos evolucionários e inteligência coletiva com o intuito de determinar o planejamento de trajetória, de modo *off-line*, a ser seguida por um robô móvel. O algoritmo evolucionário selecionado é o algoritmo genético e as técnicas de inteligência coletiva são: nuvem (enxame) de partículas, colônia de formigas, colônia de bactérias e sistema imunológico artificial. Estas técnicas foram selecionadas com o propósito de realizar um estudo comparativo entre diversas técnicas de inteligência computacional para a solução de problemas de robótica móvel. Estas técnicas são também utilizadas para projetar um controlador *fuzzy* com o intuito de controlar um robô móvel, cujo objetivo é explorar ao máximo um ambiente previamente desconhecido. Nesta dissertação, são apresentados também estudos de caso para testar as técnicas descritas.

Palavras-Chave: inteligência coletiva, controle *fuzzy*, planejamento de trajetórias, robótica móvel, algoritmo genético.

Abstract

This work involves concepts of evolutionary algorithms and swarm intelligence with a purpose of off-line path planning for a mobile robot. The selected evolutionary algorithm is a genetic algorithm, and the swarm intelligence techniques are particle swarm, ant colony, bacteria colony and artificial immunological system. These techniques were selected for a comparative study among some soft computing techniques for mobile robotics problems solving. These techniques are also used for optimizing a fuzzy controller for a mobile robot, being the objective of the robot to explore a previously unknown environment. Case studies are presented to show the suitability of the presented techniques.

Keywords: swarm intelligence, fuzzy control, path planning, mobile robotics, genetic algorithms.

Capítulo 1

Introdução

A área de inteligência computacional (IC) foi proposta, em 1994, por L. Zadeh e J. Bezdek (Zadeh, 1994; Bezdek, 1994), e é utilizada para denominar sistemas que exploram a tolerância à imprecisão, a incerteza, a verdade parcial e aproximações para obter soluções robustas, a baixo custo e condizentes com a realidade (Bonissone, *et al.*, 1999).

As abordagens da IC são inspiradas em mecanismos biológicos de aprendizado e adaptação ao meio. A IC é subdividida basicamente em algoritmos evolucionários (ou evolutivos), inteligência coletiva, sistemas *fuzzy*, que são mais bem explicados no Capítulo 4 desta dissertação, e as redes neurais artificiais (RNA) (Jin, 2004).

A popularidade da IC vem em parte da possibilidade de sinergia entre seus componentes. Uma característica relevante da IC é a capacidade intrínseca de conceber sistemas híbridos baseados na integração das tecnologias que a compõem. Esta integração permite um mecanismo de raciocínio complementar e métodos de busca que permitam combinar domínio do conhecimento e dados empíricos para desenvolver ferramentas computacionais flexíveis, capazes de resolver problemas complexos (Bonissone, *et al.*, 1999).

A computação evolucionária difere dos paradigmas convencionais de busca e de otimização em três aspectos principais pela utilização de: (i) uma população de potenciais soluções, (ii) uma função de adequabilidade ao meio (*fitness*) e (iii) regras de transição probabilísticas.

A maioria dos paradigmas de otimização convencionais (ou clássicos) baseados em cálculo move a solução potencial de um ponto a outro dentro de um determinado espaço de busca, utilizando algumas regras determinísticas. Uma das desvantagens de métodos que obedecem este princípio é a possibilidade da solução permanecer presa em mínimos (ou

máximos) locais. A computação evolucionária, por outro lado, inicia com uma população de pontos. Com isso, vários pontos de mínimo (ou de máximo) podem ser explorados simultaneamente, reduzindo a probabilidade de a mesma ficar presa em mínimos locais ruins.

Ao contrário de muitos paradigmas de busca, os algoritmos evolucionários não necessitam de informações auxiliares ao problema, como funções dependentes de cálculos de derivadas para a determinação de subidas (problemas de maximização) ou descidas de encosta (problemas de minimização). Na computação evolucionária, a aptidão (*fitness*) de cada membro da população guia a busca por melhores soluções. A função de aptidão é uma métrica direta do desempenho do indivíduo da população quanto à função objetivo (custo) a ser maximizada (ou minimizada).

O fato de os paradigmas da computação evolucionária utilizarem regras de transição probabilísticas não significa que uma busca estritamente aleatória seja realizada. Neste contexto, operadores estocásticos são aplicados de forma a direcionar a busca para regiões do espaço de busca que possivelmente possuam melhores valores de aptidão.

De forma análoga às técnicas de computação evolucionária, as técnicas de inteligência coletiva também não necessitam de informações relacionadas ao cálculo de derivadas e utilizam operadores estocásticos para o direcionamento das potenciais soluções.

Recentemente, as técnicas de computação evolucionária e inteligência coletiva estão sendo utilizadas na área de robótica móvel, principalmente em sensoriamento do ambiente (fusão de sensores) (Nadimi e Bhanu, 2004), planejamento de trajetórias (Wang, 2003), otimização de controladores (Messom, 2002) e cooperação entre robôs (Huntsberger, *et al.*, 2004).

O sensoriamento do ambiente é responsável por prover informações que permitam ao robô móvel se localizar e interagir com o ambiente em que está situado. Os sistemas de sensoriamento dos robôs podem ser compostos de um simples sensor, uma combinação de múltiplos sensores ou de sensores de diferentes tipos, onde os dados são processados em conjunto (Pereira, *et al.*, 2001).

O problema de otimização de trajetórias de robôs móveis tem recebido atenção, nos últimos anos, devido ao apelo cada vez maior à autonomia dos robôs e também a ser um problema complexo (Cai e Peng, 2002). O planejamento de uma trajetória livre de colisões é um requisito importante para que um robô móvel realize tarefas sem a supervisão de seres humanos. O planejamento de trajetórias consiste de encontrar uma trajetória contínua da

posição inicial à posição de destino que evite os obstáculos presentes no ambiente e satisfaça algum certo critério de desempenho (Diéguez, *et al.*, 2003).

O foco de pesquisas sobre a cooperação de robôs é um tópico emergente no meio acadêmico. A idéia da utilização de vários robôs ao invés de um só para a solução de um problema originou-se da necessidade de soluções de tarefas que são excessivamente difíceis de serem realizadas por um único robô (Pereira, *et al.*, 2001).

Devido ao crescente interesse em autonomia na área de robótica móvel, uma importante tarefa passa a ser a de projeto de controladores para robôs. Entretanto, devido à complexidade existente no projeto de controladores para robôs, muitas vezes algoritmos evolucionários (Alcalá, *et al.*, 2003) e técnicas de inteligência coletiva (Esmin, *et al.,* 2002) são aplicadas na otimização destes controladores.

1.1 Objetivo do Trabalho

Um dos principais desafios na abordagem escolhida para esta dissertação diz respeito à dificuldade de realização do planejamento de trajetórias em robótica móvel, devido ao grande número de possíveis trajetórias a serem percorridas que possibilitam ao robô atingir o seu destino. Além deste item, existem dificuldades no desenvolvimento de controladores em aplicações de robôs móveis, de forma a garantir que o robô siga a trajetória previamente planejada.

Uma área emergente neste foco também é a cooperação entre robôs. Recentemente, o interesse por este tópico tem crescido, principalmente devido à possibilidade do cumprimento de tarefas complexas com a utilização de diversos robôs simples, ao invés de necessitar de um robô com comportamento complexo. Isto compreende basicamente o fato de o custo ser menor para o desenvolvimento de diversos robôs simples do que um robô com mecanismos aprimorados de projeto e movimentação.

Tomando como base os desafios apresentados, e também se motivando na crescente necessidade de desenvolvimento de robôs móveis que apresentem mecanismos de autonomia, aprendizado e adaptação. Foi optado por realizar nesta dissertação uma análise comparativa entre diversas técnicas de inteligência computacional em problemas de robótica móvel, incluindo planejamento de trajetos e planejamento de controladores.

Levando em consideração a quantidade de técnicas de inteligência computacional existentes, se optou por realizar a análise comparativa, tomando como base as características de busca em largura e não de busca em profundidade, ou seja, o estudo dos algoritmos não foi aprofundado.

A opção pela utilização destas diversas técnicas de inteligência computacional veio do fato de as mesmas possuírem extrema aplicação e eficiência na natureza, como pode ser visto pelo simples fato de os animais conseguirem alimentos e sobreviverem na natureza, mesmo eles individualmente não possuindo características inteligentes.

1.2 Estado da Arte

De forma a facilitar a descrição do estado da arte desta dissertação, o mesmo é apresentado dividido em subseções: (i) inteligência computacional, (ii) controle de processos, e (iii) robótica móvel.

1.2.1. Inteligência Computacional

A área de inteligência computacional é dividida em diversas abordagens, entre as quais redes neurais, sistemas *fuzzy*, algoritmos evolucionários e inteligência coletiva. Esta dissertação envolve sistemas *fuzzy*, computação evolucionária e inteligência coletiva.

A computação evolucionária tem sua origem na década de 1950, porém este campo permaneceu sem grandes aplicações por três décadas, devido principalmente à falta dos recursos computacionais necessários naquela época. A computação evolucionária é composta por diversos algoritmos, sendo que são mais conhecidos os algoritmos genéticos (AGs) (Mitchell, 1996), programação genética (Banzhaf, 1998), programação evolucionária (Fogel e Fogel, 1990), estratégias evolutivas (Duarte e Tome, 2000) e evolução diferencial (Back, *et al.*, 1997).

Os algoritmos evolucionários têm sido usados de forma eficiente para resolver uma grande variedade de problemas ao longo das últimas décadas (Dasgupta e Michalewicz, 2001), principalmente quando são considerados problemas de otimização não-lineares e com restrições (Tan, *et al.*, 2002), tais como a determinação de estruturas de aviões (Karr, *et al.*, 2004), validação funcional de *pipelines* de microprocessadores (Corno, *et al.*, 2004), síntese de circuitos lógicos (Aguirre e Coello, 2003; Miller, *et al.*, 2000), otimização de controladores

(Nanayakkara, *et al.*, 2001), diagnóstico médico (Podgorelec e Kokol, 2001; Papageorgiou, *et al*., 2004) e controle de robôs (Randall e Pipe, 2000; Muscato, 2000).

A operação dos AGs é baseada no princípio da evolução natural descrita por C. Darwin. De forma distinta aos algoritmos de busca convencionais, onde uma solução ótima é buscada a partir de uma única solução potencial, existe um conjunto de soluções potenciais em um AG, que competem entre si ao longo da execução do algoritmo. De acordo com Aickelin e White (Aicklin e White, 2004), os algoritmos genéticos não foram inicialmente desenvolvidos com o objetivo de otimizar funções, a otimização de funções somente foi possível após algumas modificações na estrutura do AG.

Resumindo, o funcionamento de um AG simples é dado pelas seguintes etapas: (i) a população inicial é gerada aleatoriamente com distribuição uniforme; (ii) os melhores indivíduos desta população têm maior probabilidade de serem escolhidos para sofrer recombinação, de forma a gerar os novos indivíduos; e (iii) além do processo de recombinação, os indivíduos sofrem o processo de mutação, de forma a aumentar a diversidade da população. Este processo ocorre até que um determinado critério de parada seja atingido (Aicklin e White, 2004).

Os AGs são utilizados nas mais diversas aplicações, entre elas a otimização de problemas de escalonamento de horários (Glibovets e Medvid, 2003), na obtenção de informações a partir de documentos (Boughanem, *et al.*, 1999), simulações de comportamento econômico (Novkovic, 1998), no desenvolvimento de controladores neurais (Dasgupta, 1998), na redução de estoques (Onwubolu e Mutingi, 2003), modelagem do *layout* de empresas (Mavridou e Pardalos, 1997), treinamento de redes neurais (Durand, *et al.*, 2000), planejamento de trajetórias (Pires, *et al.*, 2004; Yang e Hu, 2005; Sierakowski, *et al.*, 2004), entre outros.

Ao longo do tempo, várias modificações tem sido propostas em relação aos AGs, entre elas a proposta de um AG acelerado (Podlena e Hendtlass, 1998), a possibilidade de desenvolvimento de *softwares* híbridos (Hunter, 1998; Yun, *et al.*, 2003), AG estruturado (Dasgupta, 1998), AG baseado em estrutura de árvores (Syarif e Gen, 2003), AGs adaptativos (Yun e Gen, 2003), AGs paralelos e escalonáveis (Rivera, 2001) e AG adaptativo cooperativo co-evolucionário (Cai e Peng, 2002).

Outro campo da IC é a inteligência coletiva (*swarm intelligence*). A inteligência coletiva é um conceito computacional inspirado em exemplos extraídos da biologia,

principalmente de insetos e das suas características de trabalho em grupos. Os algoritmos de inteligência coletiva são baseados no princípio de que as habilidades do grupo em questão transcendem as habilidades dos indivíduos que constituem aqueles grupos (Bonissone, *et al.*, 1999; Agassounon, *et al.*, 2004), ou seja, na capacidade que indivíduos com pouca inteligência possuem de apresentar um comportamento inteligente quando estão em grupo (Passino, 2002).

As técnicas de inteligência coletiva têm provado possuir propriedades como robustez, flexibilidade e habilidade na solução de problemas complexos explorando características como paralelismo e auto-organização (Mondana, *et al.*, 2004). Devido a estas propriedades o número de aplicações de inteligência coletiva tem crescido, principalmente em aplicações como robótica (Mondana, *et al.*, 2004; Agassounon, *et al.*, 2004; Cao*, et al.*, 2004; Trianni, *et al.*, 2003).

Os algoritmos destacados dentro de inteligência coletiva são: nuvem de partículas ou *Particle Swarm Optimization* (PSO) (Kennedy e Eberhart, 1995), colônia de formigas (Dorigo*, et al.*, 1996), colônia de bactérias e sistemas imunológicos artificiais (Dasgupta, *et al*., 2003).

O algoritmo PSO é originariamente uma abordagem de simulação do comportamento social de organismos como bandos de pássaros e cardumes de peixes (Kennedy e Eberhart, 1995). Estas simulações eram baseadas na manipulação das distâncias entre os indivíduos do grupo em questão, ou seja, o comportamento do grupo era considerado como um esforço dos indivíduos do grupo em manterem uma distância ótima entre os seus companheiros. Estas simulações foram tomadas como base para o desenvolvimento do algoritmo PSO.

O funcionamento básico do PSO leva em consideração que cada partícula presente na população possui uma memória onde fica armazenada a melhor posição já visitada pela mesma dentro do espaço de busca. O movimento de cada partícula é acelerado em direção à melhor posição já visitada e também em direção a melhor posição já visitada pelas partículas de sua vizinhança. No limite, a vizinhança é o próprio enxame (Parsopoulos e Vrahatis, 2002).

Existem algumas variantes do algoritmo PSO, uma delas onde a busca é realizada localmente e outra onde a busca é realizada globalmente, em uma terceira variante a aceleração das partículas é inversamente proporcional à distância (Parsopoulos e Vrahatis, 2002), PSO utilizando aprendizado cooperativo (Bergh e Engelbrecht, 2000), o PSO

carregado (*Charged Particle Swarm Optimization – CPSO*) (Blackwell, 2003), PSO com base na estratégia min-max (Li, 2004), PSO hierárquico (Janson e Middendorf, 2004), *quantum* PSO (Yang, *et al.*, 2004), PSO com operador de mutação (Qin, *et al.*, 2004), algoritmos resistentes à ruído (Pugh, *et al.*, 2005) e também algoritmos de co-evolução (Doctor, *et al.*, 2004; Krohling, *et al.*, 2004).

A gama de aplicações do algoritmo PSO é variada. Alguns exemplos que podem ser citados são análise de níveis de poluentes (Lu, *et al.*, 2002), descobrimento de padrões de seqüências de proteínas (Chang, *et al.*, 2004), otimização de funções (Parsopoulos e Vrahatis, 2002), treinamento de redes neurais (Loi, *et al.*, 2004; Brandler e Hendtlass, 2002; Sierakowski, *et al.*, 2005) e robótica (Doctor, *et al.*, 2004).

Uma outra abordagem emergente da inteligência coletiva é o algoritmo de colônia de bactérias. O algoritmo de colônia de bactérias baseia-se no comportamento das células de bactérias na presença de atrativos ou repelentes químicos (Muller, *et al.*, 2002).

O funcionamento básico do algoritmo de colônia de bactérias faz com que as bactérias continuem se movendo em uma determinada direção enquanto a concentração dos atraentes também estiver crescendo, quando a concentração parar de crescer a bactéria passa a procurar uma nova direção para a qual deve se locomover. Após um determinado número de passos, as bactérias que conseguiram captar mais nutrientes se reproduzem e as bactérias com menor quantidade de nutrientes morrem, ocorrem processos de eliminação e dispersão das bactérias, onde as mesmas são reposicionadas em uma posição aleatória do espaço de busca (Passino, 2002).

Algumas aplicações de colônia de bactérias podem ser a otimização de funções (Muller, *et al.*, 2002), identificação de modelos não-lineares (Nawa e Furuhashi, 1997), problemas de escalonamento (Miwa, *et al.*, 2002), desenvolvimento de controladores *fuzzy* (Nawa e Furuhashi, 1999) e robótica (Dhariwal, *et al.*, 2004; Coelho e Sierakowski, 2005).

Outra técnica de inteligência coletiva muito conhecida é a técnica de colônia de formigas, cuja inspiração veio da habilidade de uma colônia de formigas exibir um comportamento complexo e coletivo a partir de comportamentos simples de diversas formigas (Heck e Ghosh, 2002).

Inicialmente, o algoritmo de colônia de formigas foi desenvolvido para otimização combinatória (Dorigo, *et al.*, 1996). Entretanto, para a otimização de problemas que não

podem ser representados através de grafos foi desenvolvida uma variante do algoritmo (Monmarché, *et al.*, 1999; Prabhaharan, *et al.*, 2004).

O funcionamento básico do algoritmo para otimização contínua consiste em simular o comportamento de cada formiga individualmente, onde cada formiga deve criar as suas regiões de busca, ou seja, um círculo (no caso de problemas bidimensionais) dentro de um determinado raio ao redor do ninho. A seguir, deve determinar qual a região de busca a ser pesquisada, explorar esta região e esquecer esta região, se for o caso. A posição do ninho vai sendo atualizada com o decorrer da execução do algoritmo de forma que o ninho represente sempre o melhor ponto encontrado até o momento (Monmarché, *et al.*, 1999).

Existem diversas aplicações dos algoritmos de colônia de formigas na literatura, entre eles estão a otimização de problemas de *layout* de máquinas (Corry e Kozan, 2004), seleção de portfólios (Doerner, *et al.*, 2004), problemas do tipo caixeiro viajante (Louarn, *et al.*, 2004; Dorigo, *et al.*, 1999), roteamento de veículos (Dorigo, *et al.*, 1999; Tian, *et al.*, 2003; Ellabib, *et al.*, 2002), coloração de grafos (Dorigo, *et al.*, 1999), planejamento de trajetórias (Sierakowski e Coelho, 2005) e reconhecimento facial (Yan e Yuan, 2004).

O algoritmo de sistema imunológico artificial foi desenvolvido baseado nas teorias e conceitos da imunologia. Segundo estes conceitos, os sistemas imunológicos, sejam eles biológicos ou artificiais, devem possuir a capacidade de adaptação, reconhecimento de padrões e memória (Silva e Zuben, 2004). Existem diversos algoritmos baseados nos sistemas imunológicos biológicos, alguns deles são os algoritmos de seleção negativa, seleção clonal e hipermutação somática (Silva, 2001). De acordo com alguns autores, o algoritmo de Sistemas Imunológicos Artificiais pode ser classificado como um ramo independente dos algoritmos evolucionários e de inteligência coletiva (Silva, 2002).

Existem diversas aplicações de sistemas imunológicos artificiais, entre elas sistemas imunológicos de informações (Chão e Forrest, 2003), detecção de anomalias (González e Dasgupta, 2003), detecção de falhas (Dasgupta, *et al.*, 2004), sistemas tolerantes a falhas (Canham e Tyrrel, 2003; Bradley e Tyrrell, 2000), proteção contra fraudes, detecção de vírus em computadores (Carvalho e Freitas, 2000), classificação de doenças (Ando e Ibo, 2003) e reconhecimento de padrões (White e Garrett, 2003).

1.2.2. Controle de Processos

A configuração convencional de controle por realimentação consiste na utilização das variáveis a serem controladas para manipular as entradas do processo, sendo que o ajuste das variáveis de entrada é efetuado de acordo com as leis de controle implementadas no controlador. Resumidamente, o objetivo de controle é manter algumas variáveis do processo próximas a um valor de referência estabelecido (Ogata, 2003).

As técnicas convencionais de controle utilizam informações dos modelos matemáticos dos processos para controlá-los. Porém, a análise e o projeto de técnicas convencionais de controle podem ser complexos quando são levados em consideração a presença de não-linearidades e o comportamento variante no tempo em processos industriais.

Recentemente, as novas gerações de controladores têm utilizado técnicas de IC de forma a lidar com a crescente complexidade dos sistemas a serem controlados, incluindo não-linearidades, perturbações e acoplamento entre variáveis (Seng, *et al.*, 1998).

Uma parte dos controladores destas novas gerações utilizam conceitos da lógica *fuzzy*. O controle *fuzzy* baseado em regras é uma metodologia que visa transformar o conhecimento de um experiente operador humano em um controlador automático (Khoury, *et al.*, 2004). Vários exemplos de controle *fuzzy* têm sido apresentados para aplicações em robótica (Driesse, *et al.*, 1999; Hendzel, 2004; Moudgal e Kwong, 1995; Chang e Li, 2002).

A utilização de controladores *fuzzy* tem recebido um crescente interesse por parte dos pesquisadores. Algumas das aplicações de controle *fuzzy* são em: controle da concentração de nutrientes (Lenas, *et al.*, 1997), controle de partículas em levitação (Ciocirlan, *et al.*, 1998; Orhan, *et al.*, 2001), controle de sistemas HVAC (*Heating, Ventilating and Air Conditioning*) (Alcalá, *et al.*, 2003; Sierakowski, *et al.*, 2004), controle de robôs (Erden, *et al.*, 2004; Howard e Zilouchian, 1998; Khoury, *et al.*, 2004; Kulitz e Ferreira, 2004; Thongchai, *et al.*, 2000; Yang e Hu, 2005), controle de cadeiras de rodas (Pires e Nunes, 2002), controle de veículos (Driessen, *et al.*, 1999; Kodagoda, *et al.*, 2002), controle de motores (Lee, *et al.*, 2003) e controle do pouso de aeronaves (Ionita e Safron, 2002).

Uma aplicação difundida na literatura é a otimização de controladores *fuzzy* através de algoritmos genéticos (Wu e Liu, 2000; Alcalá, *et al.*, 2003; Erden, *et al.*, 2004), PSO (Esmin, *et al.*, 2002; Parsopoulos, *et al.*, 2004) e colônia de bactérias (Nawa e Furuhashi, 1998; Nawa e Furuhashi, 1999). As metodologias de IC são também utilizadas para o desenvolvimento de controladores utilizados para robôs móveis (Messon, 2002).

Alguns controladores desta nova geração também utilizam controladores neurais. Floreano (Floreano e Mondada, 1996) apresenta um controlador neural otimizado através de procedimentos evolutivos. Meyer (Meyer, 1998) apresenta uma coletânea de resultados obtidos a partir da aplicação de controladores neurais a robôs móveis reais.

1.2.3. Robótica

Uma definição básica de um robô fixo pode ser um manipulador programável multifuncional capaz de mover materiais, partes, ferramentas ou dispositivos específicos através de movimentos variáveis programados para realizar uma variedade de tarefas (Silva, 2003).

Robôs móveis autônomos podem ser definidos como aqueles capazes de se locomover com pouca ou sem intervenção humana. Existem diversas técnicas que podem ser utilizadas para dotar um robô com características inteligentes, fundamentadas em inteligência computacional (Barberá e Skarmeta, 2001).

Recentemente, o interesse no desenvolvimento de robôs móveis autônomos para trabalhar em missões de exploração e missões perigosas em ambientes desconhecidos tem crescido, principalmente pelo fato de que a realização deste trabalho por seres humanos pode ser repetitiva, tediosa, requerer grande concentração, ou pode vir até mesmo a ser perigosa (Davison e Kita, 2002; Minchev, *et al.*, 2004).

Os robôs têm sido cada vez mais utilizados no cotidiano. Alguns exemplos de aplicações de robótica são na direção de veículos em estradas (Martinet e Thibaud, 2000), controle de veículos utilizados na agricultura (Ferentinos, *et al.*, 2002), inspeção de túneis (Yao, *et al.*, 2003), rotinas de inspeções industriais (Davison e Kita, 2002), controle na navegação de uma cadeira de rodas em um ambiente fechado (Sgouros, 2002), controle de processos em meios aquáticos (Maruyama, *et al.*, 2004) e veículos autônomos aéreos (Carruthers, *et al.*, 2005). Uma outra área com perspectivas crescentes da utilização de robôs é a área de saúde (Buckmann, *et al.*, 1998; Hoppenot e Colle, 2002).

A autonomia de robôs móveis pode ser dividida em duas grandes áreas, aquelas onde é necessário o planejamento prévio da trajetória, e aquelas onde este planejamento não é necessário. Existem diversas propostas para o planejamento de trajetórias na literatura. Gallina e Gasparetto propõem a geração da trajetória através da soma de harmônicas (Gallina e Gasparetto, 2000). Haït *et al.* propõem um algoritmo para planejamento de trajetórias em

terrenos acidentados baseado na estática e na cinemática dos robôs (Häir, *et al.*, 2002). Diéguez *et al.* propõem a integração dos planejamentos de trajetória local e global através de um mapeamento do ambiente (Diéguez, *et al.*, 2003). Cai e Peng propõem um algoritmo para o planejamento de trajetória para um grupo de robôs baseado em uma nova abordagem de AGs (Cai e Peng, 2002). Ge e Cui demonstram o planejamento de trajetória através do método de campos de potenciais (Ge e Cui, 2002). Wang utiliza um processo de planejamento de trajetórias baseado em visão computacional (Wang, 2003). Watanabe *et al.* propõem um método do planejamento de trajetórias baseado em estratégias evolucionárias (Watanabe, *et al.*, 1999). Thomaz *et al.* utilizam algoritmos genéticos para o planejamento de trajetórias (Thomaz, *et al.*, 1999). Wu *et al.* analisam o problema de planejamento de trajetórias com obstáculos dinâmicos (Wu, *et al.*, 2001). Xiao *et al.* apresentam um planejador adaptativo de trajetórias (Xiao, *et al.*, 1997). Fujimori *et al.* apresentam um sistema de navegação de robôs baseada em desvio de obstáculos (Fujimori, *et al.*, 1997), entre outras propostas.

O interesse na cooperação de robôs tem aumentado recentemente pelo fato de um único robô mais complexo não possuir a autonomia, mobilidade nem a capacidade de manipulação que diversos robôs mais simples têm quando em conjunto (Hunsberger, *et al.*, 2004). Existem diversos exemplos de utilização de cooperação entre robôs ao invés da utilização de um único robô, entre elas as pesquisas espaciais da NASA (*National Aeronautics and Space Administration*) (Hunsberger, *et al.*, 2004; Schenker*, et al.*, 2003), diagnóstico de falhas (Yan, *et al.*, 2002), comunicação entre robôs não-homogêneos cooperativos (Jung e Zelinski, 2000), exploração de terrenos (Rekleitis, *et al.*, 2001; Tunstel, *et al.*, 2002), cooperação para o transporte de objetos (Kube e Bonabeau, 2000) e cooperação de robôs em processos industriais (Preuss, 2004). Além destas, existem aplicações onde o nível de interação entre os robôs não se restringe apenas à cooperação propriamente dita, mas ocorre também uma adição de novas funcionalidades mecânicas a um simples robô, conforme mencionado em (Mondana, *et al.*, 2004).

Um dos robôs móveis mais utilizados atualmente em estudos nas áreas de robótica é o Khepera. Este tem sido muito utilizado devido ao menor custo, à facilidade de transporte, ao tamanho reduzido e à facilidade que se tem de adquirir dados originários de tal robô. Por ser pequeno, pode-se facilmente ter um computador próximo a ele para coletar dados (Ludgrenm 2003). O robô móvel Khepera possui 55 mm de diâmetro, 30 mm de altura e aproximadamente 70g de peso, sendo composto por duas rodas paralelas responsáveis por

movimentá-lo e oito sensores infravermelhos responsáveis por identificar o ambiente ao seu redor (Alves e Osório, 2003).

Os robôs móveis Khepera têm sido utilizados em diversos estudos de caso, entre eles desvio de obstáculos, seguimento de trajetórias (Ludgrenm 2003; Thomas, *et al.*, 1999), cooperação de robôs (Ijspeert, *et al.*, 2001; Fong, *et al.*, 2003; Ostergaard e Lund, 2003) e autonomia de robôs (Vellasco, *et al.*, 1999; Alves e Osório, 2003).

1.3 Proposta de Dissertação

Esta dissertação objetiva a utilização de diversas técnicas de inteligência coletiva para a obtenção das trajetórias "ótimas" para alguns estudos de caso. Além disso, são propostas algumas modificações nos algoritmos de inteligência coletiva como forma de melhorar a eficiência dos mesmos. Na dissertação será apresentado um estudo comparativo de desempenho entre os algoritmos: nuvem de partículas, colônia de bactérias, colônia de formigas e sistema imunológico artificial junto aos algoritmos genéticos para problemas como planejamento de trajetória e otimização de controladores *fuzzy* em robótica móvel.

Neste contexto, é proposta também a utilização de controladores *fuzzy* sobre robôs móveis para maximizar a exploração de ambientes previamente desconhecidos.

1.4 Organização da Dissertação

Esta dissertação é composta por sete capítulos. O capítulo 2 apresenta uma introdução sobre computação evolucionária, focando-se principalmente no funcionamento de um algoritmo genético.

O capítulo 3 apresenta os conceitos básicos de inteligência coletiva, e os fundamentos das técnicas de nuvem de partículas, colônia de bactérias, colônia de formigas e sistemas imunológicos artificiais. Cada uma das técnicas é descrita de forma a facilitar a compreensão do funcionamento dos algoritmos para a sua aplicação em problemas de otimização.

O capítulo 4 apresenta os conceitos de sistemas e controladores *fuzzy*. São também apresentados os modelos de controladores *fuzzy* mais conhecidos na literatura.

No capítulo 5, uma descrição dos problemas aos quais serão aplicados os algoritmos analisados anteriormente é apresentada. Neste capítulo é também apresentado o controlador *fuzzy* desenvolvido para o segundo estudo de caso.

No capítulo 6, são descritos os parâmetros utilizados para os algoritmos e também são discutidos os resultados obtidos com a utilização dos parâmetros e dos controladores apresentados no capítulo 5 aos problemas descritos naquele mesmo capítulo.

Por fim, no capítulo 7 são apresentadas as conclusões obtidas nesta dissertação, e algumas perspectivas de trabalhos futuros.

Capítulo 2

Computação Evolucionária

As técnicas de IC são, em boa parte, bio-inspiradas. Portanto, é de grande valia apresentar algumas regras que regem o funcionamento da natureza. Charles Darwin apresentou as hipóteses para explicar a origem das espécies (Darwin, 1959):

1. Os filhos tendem a ser em maior número que os pais;
2. O total da população de uma espécie permanece aproximadamente constante;
3. De (1) e (2) conclui-se que haverá luta pela sobrevivência;
4. Dentro de uma mesma espécie, os indivíduos apresentam pequenas diferenças.

Estas hipóteses se tornaram conhecidas como integrantes do Princípio da Seleção Natural. Segundo este princípio, os indivíduos cujas variações melhor se adaptam ao ambiente têm maior probabilidade de sobreviver e se reproduzir. De forma complementar a este princípio, foram adicionadas mais três hipóteses (Silva, 2001):

5. Algum processo de variação continuada deve ser responsável por introduzir novas informações junto ao material genético dos organismos;
6. Não há limite para a sucessão de variações que podem ocorrer;
7. A seleção natural é o mecanismo para a preservação das novas informações correspondentes a uma maior adaptação.

Os conceitos básicos da seleção natural foram a inspiração para o desenvolvimento da computação evolucionária. Pode-se dizer que a computação evolucionária baseia-se em conceitos da teoria da seleção natural e opera computacionalmente inspirada em mecanismos de evolução natural e genética.

A computação evolucionária é um conjunto de técnicas estocásticas de busca e otimização inspirados nos mecanismos de evolução e de genética. A idéia básica de um

algoritmo evolucionário é a evolução de uma população de indivíduos, de forma a melhorar o desempenho médio dos indivíduos pertencentes a esta população com base em uma medida de adaptabilidade relacionada ao ambiente em que eles estão localizados (Castro, 2001).

A computação evolucionária é composta de diversos paradigmas, entre os quais algoritmos genéticos, programação evolucionária, estratégias evolucionárias e programação genética. A seguir, é apresentada uma breve descrição dos algoritmos genéticos, por se tratar da técnica mais explorada da computação evolucionária na literatura.

2.1 Algoritmos Genéticos

2.1.1. Introdução

Na natureza, os indivíduos mais bem adaptados à competição por recursos sobrevivem. A adaptação às variações do ambiente é imprescindível para a sobrevivência dos indivíduos e das espécies. Com base nestes conhecimentos de evolução biológica, John Holland publicou o livro "*Adaptation in Natural and Artificial Systems*" (Holland, 1975), hoje considerado como a referência básica sobre os algoritmos genéticos (AGs). Desde então, estes algoritmos têm sido aplicados com sucesso em diversos problemas do mundo real de busca e otimização (Man, *et al.*, 1996).

Os AGs simulam a teoria da evolução biológica de forma a obter soluções para problemas de otimização (Filho, 1996), dentre outras possíveis aplicações. Os AGs operam com uma busca estocástica, onde o espaço de busca é percorrido baseando-se em transições probabilísticas (Dias, 2000).

Existem quatro diferenças básicas entre os AGs e os métodos tradicionais de busca e otimização. Os AGs (Dias, 2000):

- Empregam uma codificação do conjunto de parâmetros e não os próprios parâmetros;
- Operam em uma população e não em um único ponto;
- Utilizam informações do valor da função custo e não informações sobre derivadas da função de otimização;
- Utilizam regras de transição probabilísticas e não determinísticas.

O maior problema da utilização dos métodos tradicionais de busca é que não existe nenhuma garantia da obtenção de um ponto de mínimo (ou máximo) global, ou seja, o

algoritmo pode convergir para um ponto de ótimo local da função (Castro, 2001), problema também válido para os algoritmos genéticos.

Os AGs são compostos por uma população de indivíduos e um conjunto de operadores sobre a população. De acordo com a inspiração de teorias evolucionárias, segundo as quais foi desenvolvido o AG, os elementos mais bem adaptados ao seu meio têm maior probabilidade de sobreviver e de se reproduzir, transmitindo o seu material genético para as novas gerações.

Um AG é composto basicamente por quatro etapas (Filho, 1996):

1. Geração de uma população inicial;
2. Avaliação de cada um dos elementos da população;
3. Seleção dos melhores elementos da população;
4. Manipulação genética, através dos operadores de cruzamento e mutação, de forma a criar uma nova população.

Após a realização destas etapas, um *loop* é realizado retornando ao passo 2, enquanto um determinado critério de parada não for atingido.

2.1.2. Nomenclatura

Devido ao fato de os AGs terem se originado de mecanismos de evolução natural e genética, a terminologia utilizada é muito semelhante à terminologia da área biológica. Para facilitar a compreensão de um AG, define-se inicialmente a nomenclatura utilizada.

Um indivíduo é uma solução potencial do AG para o problema, uma população é um conjunto de indivíduos, ou seja, é o conjunto de possíveis soluções. Um cromossomo é a representação de um indivíduo em linguagem computacional, detalhada na seção 2.1.3.

Um gene é o menor elemento possível de um cromossomo, ou seja, é uma parte da representação de um indivíduo. No caso da representação binária, os alelos seriam os bits. Por fim, uma geração é um determinado instante de tempo de uma população. A Tabela 2.1 apresenta um glossário dos termos utilizados em AGs. Um alelo é a instância de um gene.

Tabela 2.1: Glossário dos termos utilizados em AGs.

Termo na Natureza	Termo em Computação
Indivíduo	Uma potencial solução do AG para o problema.
População	Conjunto de todas as potenciais soluções para o problema.
Cromossomo	Representação computacional de um indivíduo.
Gene	É a menor porção existente dentro de um cromossomo.
Alelo	Instância de um gene.
Geração	Determinado instante de tempo de uma população.

De forma a facilitar a compreensão destes termos, pode-se analisar a Fig. 2.1, levando-se em consideração que cada instante de tempo representa uma geração.

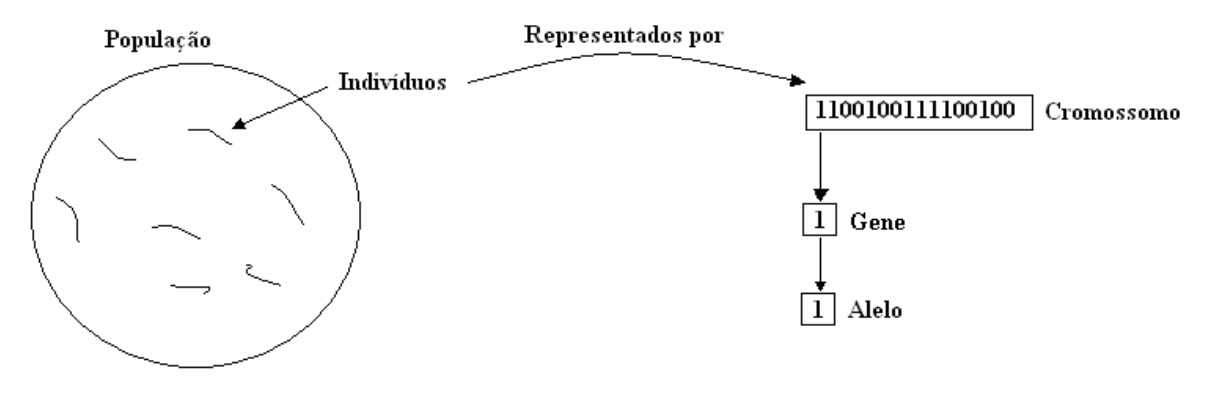

Figura 2.1: Representação utilizada pelos AGs.

2.1.3. Representação

De forma a desenvolver um AG de forma computacional, deve ser possível expressar as variáveis que se deseja otimizar em uma linguagem computacional. Basicamente, existem quatro formas de realizar a representação da população computacionalmente: a representação binária, a codificação Gray, a representação real e a representação inteira.

a. Representação Binária

A representação binária é a representação originalmente proposta por Holland (1975), esta representação utiliza apenas os números binários (0 e 1) para a representação das variáveis. Por isso, pode ser necessária uma operação de codificação e decodificação, ou seja, um procedimento de transformação do valor real da variável sendo analisada para a codificação binária e vice-versa, para problemas originariamente reais. Em casos onde o problema abordado é inerentemente binário, não existe esta necessidade de codificação e decodificação.

Para a utilização desta representação, é necessário definir o intervalo de interesse para cada uma das variáveis sendo analisadas e também a precisão desejada para cada uma dessas variáveis. Assim é possível determinar o número de bits necessários para a representação de cada variável.

A representação das variáveis é realizada da seguinte forma:

$$X = [00111 \;\; 10101 \;\; ..... \;\; Xn] \qquad (2.1)$$

A principal dificuldade existente em se trabalhar com a codificação binária é o fato de se necessitar de uma grande cadeia para se conseguir uma boa precisão. Outro problema é a existência de *Hamming cliffs*, ou seja, grandes diferenças nas cadeias de bits que codificam dois números inteiros próximos (por exemplo, os números $15_{10} - 01111_2$ e $16_{10} - 10000_2$) (Avila, 2001).

Utilizando-se esta codificação para um AG responsável por otimizar um parâmetro variante entre 0 e 1 de um determinado controlador, poderia ser utilizada a codificação apresentada na Tabela 2.2.

Tabela 2.2: Representação do Código Binário.

Código Binário	**Valor**
0000_2	$0{,}0000_{10}$
0001_2	$0{,}0667_{10}$
0010_2	$0{,}1333_{10}$
0011_2	$0{,}2000_{10}$
0100_2	$0{,}2667_{10}$
0101_2	$0{,}3333_{10}$
0110_2	$0{,}4000_{10}$
0111_2	$0{,}4667_{10}$
1000_2	$0{,}5333_{10}$
1001_2	$0{,}6000_{10}$
1010_2	$0{,}6667_{10}$
1011_2	$0{,}7333_{10}$
1100_2	$0{,}8000_{10}$
1101_2	$0{,}8667_{10}$
1110_2	$0{,}9333_{10}$
1111_2	$1{,}0000_{10}$

Utilizando-se esta codificação para representar uma população responsável por otimizar um controlador *fuzzy* composto por 2 entradas e 1 saída, seriam necessários otimizar 18 parâmetros (ver Capítulos 4 e 5), considerando que cada entrada/saída fosse representada por 3 variáveis lingüísticas. Portanto a representação de um indivíduo hipotético seria a representada na Fig. 2.2.

$$Indivíduo = \begin{bmatrix} 0001\ 0010\ 1101\ 0011\ 1101\ 1110\ 1111\ 1100\ 0000 \\ 0001\ 0110\ 1110\ 0000\ 1001\ 1100\ 1111\ 0001\ 0001 \end{bmatrix}$$

Figura 2.2: Representação de um indivíduo representado pela codificação binária.

b. Representação com Código de Gray

Na codificação Gray, de forma semelhante à codificação binária, apenas os elementos 0 e 1 são utilizados para representar as variáveis sendo analisadas. A codificação Gray foi desenvolvida de forma a solucionar o problema de *Hamming cliffs*, isso ocorre devido ao fato de quaisquer dois números inteiros subseqüentes codificados segundo o código de Gray variem em apenas um bit a sua codificação.

Utilizando-se o código de Gray para representar o mesmo caso apresentado na representação binária, a codificação seria a seguinte.

Tabela 2.3: Representação da Codificação de Gray.

Código Binário	Valor
0000_2	$0{,}0000_{10}$
0001_2	$0{,}0667_{10}$
0011_2	$0{,}1333_{10}$
0010_2	$0{,}2000_{10}$
0110_2	$0{,}2667_{10}$
0111_2	$0{,}3333_{10}$
0101_2	$0{,}4000_{10}$
0100_2	$0{,}4667_{10}$
1100_2	$0{,}5333_{10}$
1101_2	$0{,}6000_{10}$
1111_2	$0{,}6667_{10}$
1110_2	$0{,}7333_{10}$
1010_2	$0{,}8000_{10}$
1011_2	$0{,}8667_{10}$
1001_2	$0{,}9333_{10}$
1000_2	$1{,}0000_{10}$

c. Representação Real

A representação real opera diretamente com os números reais. A representação real surgiu como uma opção para a aplicação em problemas complexos ou problemas com grandes

espaços de busca, onde se reduziria o custo computacional do processo de codificação e decodificação.

Esta forma de representação se tornou atraente pela redução do custo computacional, principalmente pela não necessidade de um processo de codificação e decodificação.

Na equação (2.2), é apresentada a codificação de uma variável representada segundo a representação real, para o mesmo exemplo utilizado nos outros casos.

$$X = \begin{bmatrix} 1{,}002 & 4{,}126 & 6{,}765 & 3{,}998 & 9{,}001 & 8{,}775 & 5{,}221 & 0{,}005 & 4{,}740 \\ 9{,}887 & 2{,}334 & 7{,}661 & 0{,}849 & 8{,}112 & 5{,}771 & 9{,}994 & 2{,}221 & 7{,}951 \end{bmatrix} \quad (2.2)$$

d. Representação Inteira

A representação inteira utiliza números inteiros para representar os indivíduos da população. Esta forma de representação é normalmente utilizada em problemas em que as soluções são representadas através de grafos, ou seja, em problemas combinatórios. Neste caso, os números representam os nós do grafo, e o cromossomo representa a ordem através da qual os nós são percorridos (Goldbarg e Luna, 2000).

Utilizando-se esta representação em um problema do tipo caixeiro viajante, onde existam 10 cidades a serem percorridas, a representação de um indivíduo hipotético é apresentada na equação (2.3)

$$X = [1\ 9\ 2\ 7\ 6\ 10\ 3\ 5\ 8\ 4] \quad (2.3)$$

Nos casos apresentados no Capítulo 5 deste trabalho, é utilizada a representação binária, pelo fato de a mesma ser a mais difundida no meio acadêmico.

2.1.4. População Inicial

Como apresentado na seção 2.1.1, o primeiro passo para a execução de um AG é a geração de uma população inicial. Este procedimento é geralmente aleatório com distribuição uniforme, sendo que cada um dos indivíduos gerados deve atender às exigências do problema proposto.

Contudo, podem existir ocasiões em que seja interessante utilizar uma seleção heurística para a geração da população inicial, ou seja, a inserção de algumas possíveis soluções interessantes previamente conhecidas. Porém, já foi comprovado que a população

inicial não é um fator crítico, desde que a mesma possua cromossomos suficientemente distribuídos no espaço de busca (Castro, 2001).

2.1.5. Função de Adaptação (*fitness*)

A função de *fitness* é um fator essencial para um AG, pois esta função é a responsável por relacionar as potenciais soluções existentes na população em função do grau de adequação à solução do problema em questão. Esta função de *fitness* é responsável por realizar uma medida quantitativa, de cada um dos indivíduos pertencentes à população, do grau de adequabilidade (ou aptidão) de cada um dos mesmos ao ambiente.

A função de *fitness* é a função a ser maximizada (ou minimizada) pelo AG.

2.1.6. Seleção

O processo de seleção se assemelha, de forma computacional, à seleção natural proposta por Darwin, sendo que os elementos mais adaptados ao ambiente (no caso dos algoritmos genéticos, à função de *fitness*) têm maior probabilidade de se reproduzir, ou seja, maior possibilidade de propagar os seus genes para a próxima geração. A seleção é responsável por determinar quais dos elementos presentes são submetidos à ação dos operadores genéticos, ou seja, quais dos elementos da população são selecionados para sofrer cruzamento e mutação.

Existem diversos tipos de processos de seleção, entre eles o método da roleta, o torneio, seleção de amostragem determinística, seleção de amostragem aleatória, seleção de amostragem estocástica. Porém, apenas os modelos mais utilizados são abrangidos neste documento (roleta e torneio). Mais detalhes podem ser encontrados em (Dias, 2000).

a. Método da Roleta

O método da roleta (*roulette wheel*) é o processo original de seleção proposto por Holland. Neste método, é calculado o *fitness* relativo de cada um dos elementos pertencentes à população. Isto significa que o *fitness* de cada um dos indivíduos é convertido em um valor percentual, sendo que este valor percentual representa a probabilidade do mesmo ser selecionado para o processo de reprodução.

O cálculo da *fitness* relativo é determinado pela equação (2.4).

$$p_i = \frac{f_i}{\sum_{k=1}^{N} f_k} \tag{2.4}$$

onde N é o tamanho da população e f_i é o *fitness* do indivíduo i da população. Na Tabela 2.4 é apresentada uma população com seus indivíduos e seus respectivos fitness. Na Fig. 2.3 é apresentada uma população de A a H com os respectivos *fitness* relativos.

Tabela 2.4: População de Indivíduos e respectivos *fitness*.

Indivíduo	***Fitness***	***Fitness* Relativo**
A	0,135	4,31%
B	0,2398	7,65%
C	0,1556	4,97%
D	0,7655	24,43%
E	0,5901	18,83%
F	0,0364	1,16%
G	0,4032	12,87%
H	0,8074	25,77%

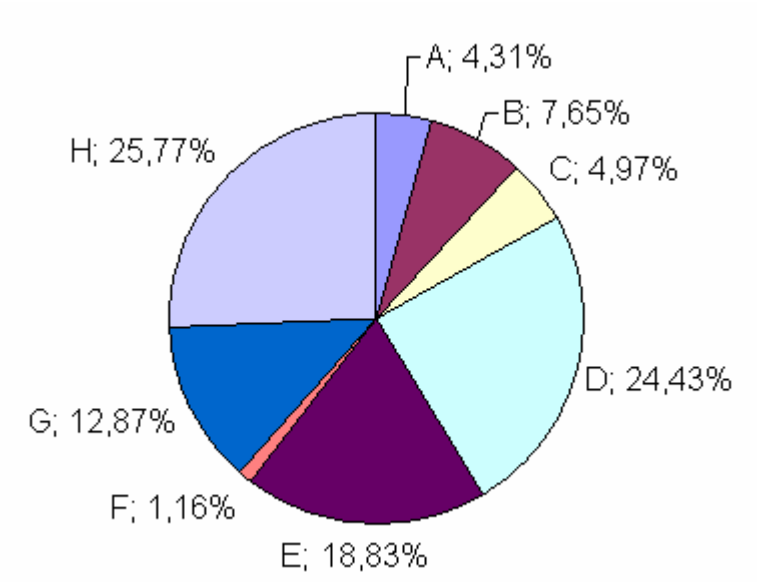

Figura 2.3: *Fitness* relativos de uma população.

Pela Fig. 2.3 nota-se que os indivíduos com maior *fitness* relativa têm maiores chances de serem selecionados, ou seja, maiores chances de se reproduzir. Uma desvantagem presente na utilização deste método de seleção é a possibilidade de existência de uma dominância da roleta por parte de um cromossomo, fazendo com que uma grande quantidade de cópias do mesmo sejam realizadas para a próxima geração, levando o algoritmo a uma convergência prematura.

b. Método do Torneio

A seleção por torneio também é bastante utilizada. Neste método de seleção é determinado um subconjunto aleatório da população, normalmente composto por dois indivíduos, e uma cópia do melhor indivíduo deste subconjunto é selecionado para sofrer a ação dos operadores genéticos.

Como conseqüência deste processo de seleção, os elementos com baixo grau de adequabilidade ao meio possuem maior chance de serem selecionados do que no método da roleta.

c. Seleção Elitista

O processo de seleção elitista normalmente é utilizado em conjunto com outros métodos de seleção (Castro, 2001). Este método de seleção é responsável por realizar a cópia dos *N* melhores cromossomos da população atual diretamente para a população subseqüente, sem sofrerem as ações dos operadores genéticos.

Esta operação garante a sobrevivência destes cromossomos na próxima geração.

2.1.7. Operadores Genéticos

Para a manipulação genética dos elementos da população, são utilizados dois operadores, são eles o cruzamento e a mutação.

a. Cruzamento ou Recombinação

Freqüentemente, os termos recombinação e cruzamento são confundidos, mas deve-se ressaltar uma diferença básica existente entre eles, o operador de recombinação é utilizado para populações que utilizam a representação real e o operador de cruzamento é utilizado para populações que utilizam representação binária ou canônica.

O operador de cruzamento é considerado o operador mais importante de um algoritmo genético (Filho, 1996). Este operador é responsável por tomar dois indivíduos da população e trocar parte de suas informações genéticas gerando dois novos indivíduos, acelerando o procedimento de busca, sendo que este processo foi inspirado no processo de reprodução sexuada existente na natureza. Deve-se ressaltar que a operação de cruzamento pode não ocorrer, pois a ocorrência ou não depende de uma probabilidade de cruzamento (parâmetro de projeto). Caso o cruzamento não ocorra, o par original é transmitido para a próxima geração.

Existem diferentes tipos de operadores de cruzamento, cruzamento de ponto único, de ponto duplo e de *n* pontos (Man, *et al.*, 1996). O operador mais utilizado é o operador de cruzamento de ponto único, um exemplo de funcionamento deste operador é apresentado na Fig. 2.4.

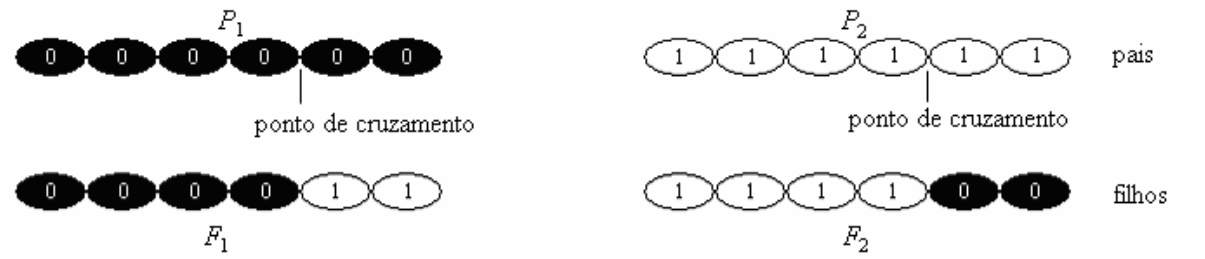

Figura 2.4: Operador de cruzamento com um ponto de corte (Filho e Treleaven, 1994).

O funcionamento deste operador é simples. Seleciona-se aleatoriamente um ponto de cruzamento. A seguir, um dos filhos é gerado com as informações genéticas do pai 1, desde o início da cadeia binária até o ponto de cruzamento, e com as informações do pai 2, do ponto de cruzamento ao final da cadeia binária. O filho 2 é gerado com as informações complementares a esta. A probabilidade associada a este operador normalmente é elevada e da ordem de 60 a 80%.

O operador de cruzamento pega os $n/2$ pares gerados pelo processo de seleção e realiza o cruzamento entre estes pares, respeitando a sua probabilidade de ocorrência, gerando uma nova população que substitui a anterior.

b. Mutação

O operador de mutação é responsável por introduzir variações aleatórias no material genético da população, de forma a aumentar a diversidade. A utilização deste operador é

relevante, pois, caso contrário, a população poderia restringir a sua busca a uma determinada região do espaço.

O operador de mutação modifica aleatoriamente o valor de um dos bits da cadeia binária. Este operador também está associado a uma probabilidade de mutação. Porém, de forma contrária ao operador de cruzamento, o percentual é baixo, normalmente da ordem de 0,5 a 5%. Caso este percentual fosse maior, a busca se tornaria acentuadamente aleatória. Na Fig. 2.5, pode-se ver um exemplo de funcionamento do operador de mutação para o caso de uma cadeia binária.

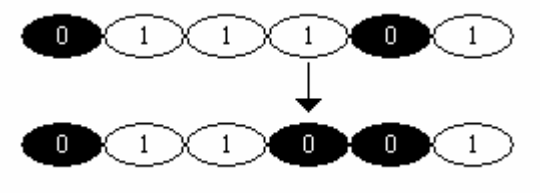

Figura 2.5: Funcionamento do operador de mutação (Filho, 1994).

2.1.8. Pseudocódigo

Na Fig. 2.6, o pseudocódigo de um algoritmo genético é apresentado.

```
Iniciar população;
FAÇA {
        Avaliar indivíduos;
        Selecionar indivíduos;
        Operação de cruzamento;
        Operação de mutação;
}ENQUANTO critério de parada não atingido
```

Figura 2.6: Pseudocódigo de um algoritmo genético.

Capítulo 3

Inteligência coletiva

A inteligência coletiva é uma área com características similares à computação evolucionária, e é utilizada para resolver problemas de diversos tipos. A inteligência coletiva é inspirada na natureza, no fato de que os diversos membros de um determinado grupo contribuem com as suas próprias experiências para o grupo, tornando-o mais forte perante os outros, ou seja, as capacidades do grupo são superiores às capacidades que os membros podem atingir individualmente. Um dos conceitos de inteligência coletiva é a propriedade de sistemas compostos de elementos não inteligentes exibirem comportamentos coletivos inteligentes, conforme apresentado em (White, 2004).

A inteligência coletiva na natureza pode ser baseada em três princípios básicos: a avaliação, a comparação e a imitação.

A avaliação é a capacidade de analisar o que é positivo ou negativo, atrativo ou repulsivo, na natureza. Até mesmo os menores seres vivos possuem esta capacidade. As bactérias têm a capacidade de perceber se o meio em que se encontram é nocivo ou não (Liu e Passino, 2002). O aprendizado não pode ocorrer a não ser que o organismo seja capaz de avaliar características atrativas e repulsivas do meio.

A comparação é a forma com que os seres utilizam outros seres como um padrão para medir a si mesmo. Os resultados destas comparações podem vir a ser uma motivação para aprender e mudar. A imitação, por sua vez, é uma forma efetiva de aprender a fazer coisas. Porém, poucos animais na natureza são realmente capazes de realizar imitações (Kennedy e Eberhart, 2001).

Estes três princípios, avaliação, comparação e imitação, podem ser combinados de forma simplificada em programas de computador, possibilitando que os mesmos se adaptem a problemas complexos.

Quando se analisa a inteligência coletiva, não se pode deixar de analisar um importante aspecto, as estratégias de busca por alimentos, ou seja, os métodos de localização, manipulação e ingestão de alimentos. Diversos animais utilizam estratégias de busca por alimentos como forma de maximizar a energia obtida por unidade de tempo gasto na busca. Com o tempo, a seleção natural tende a eliminar os animais com estratégias de busca pobres e a favorecer a propagação de genes dos animais com estratégias de busca bem sucedidas. Isto ocorre porque os animais com melhores estratégias têm maior probabilidade de se procriarem, pelo fato de estarem com mais energia.

O estudo de estratégias ótimas de busca por alimentos tenta explicar as adaptações das estruturas e comportamentos de organismos a problemas e restrições ambientais na busca por alimentos. Estratégias ótimas de busca por alimentos são utilizadas em algoritmos de busca (Passino, 2002; Pires, *et al.*, 1999).

Uma questão importante a ser relatada quanto à estratégia de busca por alimentos é a possibilidade de formação de colônias. Neste caso, animais da mesma espécie são atraídos uns aos outros, formando colônias de animais. Estas colônias são formadas devido à necessidade de cooperação por parte dos indivíduos para que os mesmos atinjam os seus objetivos. Pode também ocorrer o contrário, a repulsão entre animais da mesma espécie. Esta repulsão pode ser explicada pela competição dos animais por recursos escassos no meio (Adler e Gordon, 2003).

O interesse por mecanismos de busca de alimentos tem crescido recentemente, principalmente quando levando em consideração aplicações de engenharia, robótica, padrões de tráfego em sistemas de transporte inteligentes e aplicações militares (Liu e Passino, 2000). Este crescente interesse pode ser explicado através dos resultados obtidos com a aplicação dos conceitos de inteligência coletiva em problemas de otimização (Liu, *et al.*, 2001).

Ao longo desta dissertação, são abrangidos alguns algoritmos relacionados à inteligência coletiva e também a sistemas *fuzzy*. A opção por algoritmos de inteligência coletiva deve-se a potencialidade destas abordagens na solução de problemas complexos de otimização.

3.1 Nuvem de Partículas

3.1.1. Introdução

A proposta do algoritmo surgiu de alguns cientistas que desenvolveram simulações computadorizadas do movimento de organismos, tais como bandos de pássaros e cardumes de peixes. Estas simulações foram fortemente baseadas na manipulação das distâncias entre os indivíduos, ou seja, o sincronismo no comportamento do bando era compreendido como um esforço dos pássaros em manter uma distância "ótima" entre eles. O sócio-biologista E. O. Wilson fez a conexão entre estas simulações e os problemas de otimização (Brandstätter e Baumgartner, 2002).

Em teoria, ao menos, os membros individuais de um bando (ou nuvem) podem se aproveitar das descobertas e da experiência anterior de todos os outros membros do bando na busca por alimentos. O fundamento de desenvolvimento da otimização por nuvem de partículas (*Particle Swarm Optimization* – PSO) é uma hipótese na qual a troca de informações entre seres de uma mesma espécie oferece uma vantagem evolucionária.

Através destas simulações simplificadas de um comportamento social e, baseando-se nos estudos de Wilson, a técnica de PSO foi desenvolvida por Kennedy e Eberhart (Kennedy e Eberhart, 1995).

De forma semelhante aos algoritmos genéticos, o PSO é baseado em uma população, sendo que cada elemento da população é denominado de partícula, ou seja, cada partícula é uma solução potencial.

Entretanto, diferentemente dos algoritmos genéticos, o PSO não possui operadores como o cruzamento e a mutação. O PSO não implementa a sobrevivência do indivíduo mais adequado ao meio, mas, ao invés disso, implementa a simulação do comportamento social e cognitivo. Este algoritmo pode ser facilmente implementado e possui características de convergência estáveis com boa eficiência computacional (Parsopoulos e Vrahatis, 2002).

3.1.2. Algoritmo

O sistema possui, inicialmente, uma população de soluções candidatas com posições aleatórias. A cada uma destas partículas é atribuída uma velocidade e as partículas passam a se movimentar pelo espaço de busca. As partículas possuem memória, armazenando nesta a sua melhor posição visitada (*pbest*) e também a adequabilidade desta posição.

A melhor posição passada pela nuvem é denominada de melhor posição global da população (*gbest*) (Gudise e Venayagamoorthy, 2003). O conceito básico do algoritmo PSO é acelerar as partículas em direção às posições *pbest* e *gbest*, com um peso de aceleração aleatório a cada passo de tempo. Matematicamente, as partículas são manipuladas conforme as equações (3.1) e (3.2).

$$V_{id}^{t+1} = W * V_{id}^{t} + c_1 * rand_1 * \left(P_{id} - X_{id}^{t}\right) + c_2 * rand_2 * \left(P_{gd} - X_{id}^{t}\right) \tag{3.1}$$

$$X_{id}^{t+1} = X_{id}^{t} + V_{id}^{t+1}\Delta t \tag{3.2}$$

onde $\Delta t = 1$, t representa a iteração atual e t+1 representa a próxima iteração V_{id} e X_{id} representam a velocidade e a posição da partícula i, com dimensão d, respectivamente, $rand_1$ e $rand_2$ são dois números aleatórios com distribuição uniforme dentro do intervalo [0,1], P_{id} é o *pbest* e P_{gd} é o *gbest*.

A equação (3.1) é utilizada para atualizar a velocidade de cada uma das partículas. Para o cálculo da velocidade é utilizada a velocidade na iteração anterior, multiplicada pelo momento de inércia, como um fator que pondera sua velocidade.

O segundo fator é composto pelo componente de cognição c_1, multiplicado por um número aleatório entre 0 e 1, multiplicado pela diferença existente entre a posição atual da partícula e a melhor posição que a partícula já atingiu ao longo da execução do algoritmo (*pbest*).

O último fator da equação é composto pela componente social c_2, multiplicado por um número aleatório entre 0 e 1, multiplicado pela diferença existente entre a posição atual da partícula e a melhor posição já atingida por qualquer partícula ao longo da execução do algoritmo (*gbest*).

A equação (3.2) representa a atualização da posição da partícula, de acordo com a sua posição anterior e sua velocidade, levando em conta que $\Delta t = 1$ foi adotado.

3.1.3. Parâmetros

Uma das razões para que o PSO seja bastante utilizado é a necessidade de ajuste de poucos parâmetros (Xie, *et al.*, 2002).

a. Constantes de Cognição e Social

As constantes c_1 e c_2 são constantes positivas denominadas de componentes de cognição e social, respectivamente. Estas são as constantes de aceleração que variam a velocidade de uma partícula em direção ao *pbest* e *gbest*, de acordo com a experiência passada.

As constantes c_1 e c_2 não são fatores críticos para determinar a convergência do algoritmo. Porém, um ajuste correto destes valores pode levar a uma convergência mais rápida do algoritmo.

Os valores de c_1 e c_2 são tomados como 2.0, segundo (Gaing, 2004). Mas pesquisas mais recentes informam que pode ser ainda melhor uma escolha do parâmetro cognitivo (c_1) maior que o parâmetro social (c_2), desde que se atenda a restrição $c_1 + c_2 \leq 4$ (Parsopoulos e Vrahatis, 2002).

b. Momento de Inércia

A utilização do momento de inércia W foi proposta por Shi *et al*. (Shi e Eberhart, 1998). Este parâmetro é responsável por um ajuste dinâmico da velocidade, sendo, portanto, responsável por balancear a busca realizada pelo algoritmo entre local e global. Quando analisado separadamente dos outros parâmetros do projeto, isso faz com que sejam necessárias menos iterações para a convergência do algoritmo. Um valor elevado do momento de inércia facilita uma busca global, pois uma maior área do espaço de buscas será percorrida, devido à velocidade das partículas. Por outro lado, um pequeno momento de inércia leva a uma busca local, pois a área do espaço de buscas percorrida será menor.

Através de um ajuste dinâmico no momento de inércia, é possível atuar na habilidade de busca. Uma vez que o processo de busca do PSO é não-linear e complexo, é difícil, se não impossível, modelar matematicamente o processo de busca de forma a ajustar dinamicamente o momento de inércia. Portanto, pode-se adotar um momento de inércia fixo, ou um momento de inércia linearmente decrescente. As alternativas para o ajuste dinâmico de W são a adoção de co-evolução ou a meta-otimização.

A aplicação de um valor elevado no momento de inércia no início, fazendo com que o mesmo decaia até um valor pequeno ao longo da execução do PSO, faz com que o algoritmo possua características de busca global no início da execução e características de busca local ao fim da execução. O valor de W decrescendo linearmente de um valor máximo de 0,9 a um valor mínimo de 0,4 ao longo da execução é comumente utilizado. Quando o momento de

inércia linearmente decrescente é adotado, normalmente se adota a equação (3.3) para a atualização de W, onde $iter_{max}$ é o número máximo de iterações e $iter$ é a iteração atual (Shi e Eberhart, 2002).

$$W = W_{\max} - \frac{W_{\max} - W_{\min}}{iter_{\max}} * iter \tag{3.3}$$

c. Velocidade Máxima

O V_{max} é a velocidade máxima permitida para as partículas, ou seja, nos casos em que o módulo da velocidade da partícula seja superior à V_{max}, limita-se a mesma à V_{max}. Quando V_{max} é alto, as partículas poderão passar direto sobre boas soluções, sem percebê-las, e quando o V_{max} é baixo, as partículas podem demorar muito para convergir. O valor de V_{max} é normalmente ajustado em torno de 10 a 20% do alcance máximo das variáveis em cada dimensão (Gaing, 2004).

d. Posição Máxima e Mínima

Outro fator importante na implementação do algoritmo é a utilização do parâmetro X_{max}, este parâmetro é responsável por limitar o espaço de busca analisado. Isto previne que as partículas realizem buscas fora da área de interesse para o problema sendo analisado (Eberhart e Shi, 2000).

3.1.4. Pseudocódigo

O pseudocódigo para o PSO é mostrado na Fig. 3.1 e a sua correspondente explicação é apresentada a seguir.

```
Iniciar populacao;
FAÇA{
        Avaliar indivíduos;
        Atualizar pbest;
        Atualizar gbest;
        Atualizar a velocidade das partículas;
        Atualizar a posição das partículas;
}ENQUANTO critério de parada não atingido
```

Figura 3.1: Pseudocódigo para o PSO.

Como pode ser observado através do pseudocódigo mostrado na Fig. 3.1, o funcionamento de um algoritmo PSO é simples, o qual é composto de uma seqüência determinada de passos.

Inicialmente, cada partícula da população é iniciada com valores aleatórios, mas tomando o devido cuidado ao verificar se os valores aleatórios gerados pertencem às fronteiras de limites máximos e mínimos admissíveis para cada variável e se os valores arbitrados atendem às restrições que o problema pode apresentar.

A seguir, inicia-se um ciclo (*loop*) contínuo que é executado enquanto o critério de parada não for atingido. Este critério de parada pode ser referente à convergência do algoritmo, um número máximo de iterações a serem executadas ou mesmo outro critério imposto pelo projetista.

Dentro deste ciclo, deve-se calcular o valor do *fitness* para cada uma das partículas pertencentes à população e também determinar a posição que possui o melhor *fitness* para cada uma das partículas, armazenando-a na variável *pbest*. Isto é realizado através de uma comparação entre o valor do *fitness* atual e o valor anterior de *pbest*.

Uma vez que todas as partículas tenham sido analisadas, deve-se determinar a posição que possui o melhor valor global para o *fitness*, ou seja, qual o melhor ponto que qualquer uma das partículas já passou. Isto é feito através de uma comparação entre os valores de *pbest*, atribuindo a melhor posição à variável *gbest*.

A seguir, o algoritmo faz a alteração da velocidade para cada uma das partículas segundo a equação (3.1) e, utilizando este novo valor da velocidade, altera a posição de cada uma das partículas, através da equação (3.2).

Através de comparações do algoritmo PSO com outros algoritmos de otimização, principalmente algoritmos evolucionários, Hu *et al*. (Hu, *et al.*, 2003), ao aplicar estes algoritmos a problemas práticos de engenharia, envolvendo restrições múltiplas, não-lineares e não-triviais, afirma que o algoritmo PSO possui algumas vantagens em relação aos outros algoritmos de computação evolucionária. O algoritmo PSO é mais rápido, ou seja, obtém resultados da mesma qualidade com uma necessidade menor de avaliações do *fitness*. É mais eficaz, pois, segundo demonstrações, o mesmo atingiu melhores resultados em relação a outros. E é mais robusto, o algoritmo é intuitivo e não necessita domínio de conhecimentos específicos para a resolução do problema, além de não existir a necessidade de ajuste dos parâmetros para diferentes aplicações.

Devido aos bons resultados encontrados na literatura (Parsopoulos e Vrahatis, 2002; Fourie e Groenwold, 2002), optou-se por utilizar esta técnica como uma das opções para a otimização da trajetória de robôs móveis. No Capítulo 6, os resultados obtidos com esta técnica serão analisados.

3.2 Colônia de Bactérias

A seleção natural tende a eliminar animais com estratégias de busca por alimentos (*foraging strategies*) fracas e a favorecer a propagação dos genes de animais que possuem estratégias de busca por alimentos favoráveis, uma vez que os mesmos têm maior possibilidade de obter sucesso na reprodução. Estes princípios evolucionários levaram os cientistas a desenvolverem a teoria da busca por alimento, baseada na hipótese de que é apropriado modelar a atividade de busca por alimentos como um procedimento de otimização (Passino, 2002).

Os animais que atuam segundo a teoria de busca por alimentos tendem a tomar ações de forma a maximizar a energia obtida por unidade de tempo, levando em conta as suas restrições fisiológicas e também as restrições ambientais.

Uma variedade de modelos já foram propostos para explicar os comportamentos de busca por alimentos dos animais. Todos estes modelos se baseiam em algumas suposições, que são as seguintes (Garber, 1987):

(i) o comportamento da busca é hereditário e diretamente relacionado à adequabilidade e ao sucesso na reprodução;
(ii) o comportamento de busca e a morfologia evoluem mais rapidamente que a taxa de mudança no ambiente e outras condições;
(iii) o tempo gasto processando os alimentos é mínimo comparado com o tempo gasto na busca pelos mesmos;
(iv) tipos de alimentos são encontrados seqüencialmente com uma taxa constante e independente;
(v) as taxas com as quais as presas são localizadas, processadas e consumidas são independentes de experiências passadas;
(vi) animais seguindo estas técnicas não possuem preferências paralelas por diferentes tipos de alimentos;

(vii) animais seguindo estas técnicas maximizam a energia adquirida através de uma hierarquia de tipos de comida e explorando apenas aqueles nas posições hierárquicas mais altas; e

(viii) as escolhas de dieta são constantes.

Através de observações, foram ressaltados alguns aspectos relacionados à estratégia de busca por alimentos em animais. A mais importante das mesmas, aplicada à área de otimização, é que a complexidade do habitat e a heterogeneidade da abundância de suprimentos no espaço e no tempo podem implicar que as estratégias de busca possam não ser constantes, mas sim variantes no espaço e no tempo (Bergman, *et al.*, 2001).

3.2.1. Bactérias

As bactérias são organismos unicelulares, uma das formas mais simples de vida existente na Terra. Apesar da sua simplicidade, elas adquirem informações sobre o meio, se orientam neste meio e utilizam estas informações eficientemente para sobreviver. Em outras palavras, as bactérias possuem um sistema de controle que as permite controlar a sua busca por comida e evitar a presença de substâncias nocivas.

A presença de flagelos nas bactérias permite a sua locomoção e isto é atingido através da sua rotação em um mesmo sentido, a uma velocidade de 100 a 200 rotações por segundo. As bactérias podem se mover de duas formas diferentes: elas podem sofrer translação (nadar por um período em uma mesma direção). Este movimento é obtido quando os flagelos são rotacionados no sentido anti-horário; ou então, elas podem sofrer uma rotação, quando os flagelos são rotacionados no sentido horário. As bactérias operam numa alternância entre estes dois modos de locomoção durante toda a sua vida (raramente os flagelos param de rotacionar). Após um período de translação, a bactéria sofre uma rotação, o período de rotação é de $0{,}14 \pm 0{,}19s$ (Passino, 2002). Após esta rotação, a bactéria estará voltada para uma direção aleatória. Quando os flagelos são rotacionados no sentido anti-horário, a bactéria se desloca na direção em que está voltada, a uma velocidade média de 10 a $20\mu m/s$, ou seja, cerca de 10 vezes o seu comprimento por segundo, por um período médio de corrida de $0{,}86 \pm 1{,}18s$, segundo (Passino, 2002).

É possível que os ambientes onde se localizem as bactérias se modifiquem, tanto de forma gradual quanto de forma repentina. Com isso, as bactérias podem sofrer um processo de eliminação, através do surgimento de uma substância nociva, ou então de dispersão, através da ação de uma outra substância geram-se assim os efeitos de eliminação e dispersão.

3.2.2. Processo de *Chemotaxis*

O processo de *chemotaxis* das bactérias é a forma como elas conseguem evitar substancias nocivas e encontrar alimentos (Passino, 2002). A simplicidade e a robustez do processo de *chemotaxis* das bactérias sugerem um ponto inicial para a construção de um algoritmo de otimização. Apesar de saber que as bactérias compartilham informações umas com as outras, ainda não é conhecido muito sobre os padrões de comunicação. Geralmente, as bactérias são consideradas como indivíduos e a interação social não é utilizada nos modelos (Muller, *et al.*, 2002).

No campo da otimização, muitos pesquisadores têm se inspirado em processos biológicos, como a evolução, para desenvolver novos métodos de otimização (Muller, *et al.*, 2002). Estas técnicas têm-se provado competitivas com os métodos de otimização clássicos, com a utilização de heurística clássica, ou métodos baseados em gradientes (quando possível), especialmente quando utilizadas em problemas de otimização de funções multimodais, não-diferenciáveis ou descontínuas.

Um dos métodos de otimização inspirados na biologia é o algoritmo de *chemotaxis*, desenvolvido por Bremermann, no qual foi proposta uma analogia com a forma como as bactérias reagem à atratividade do meio (Muller, *et al.*, 2002).

3.2.3. Equacionamento

Nesta seção, é considerada a aplicação da teoria de busca por alimentos às bactérias, devido ao fato das mesmas serem mais simples que os outros seres vivos e de poderem se utilizar da teoria de *chemotaxis* de forma a obter benefícios para o grupo.

Como mostrado na seção 3.2.1, as bactérias operam sempre em um de dois modos, deslocamento ou rotação, considerando-se que o primeiro passo a ser executado seja uma rotação. O processo de rotação gera um vetor aleatório, que representa a direção para a qual a bactéria estará voltada após a rotação. Durante o processo de rotação, a posição da bactéria não é alterada.

As posições das bactérias em um determinado instante de tempo, durante o processo de deslocamento, podem ser determinadas através da fórmula apresentada na equação (3.4), onde a posição ($\phi'(j+1,k,l)$) é determinada através da posição no instante anterior mais um valor do tamanho do passo ($C(i)$) a ser aplicado em uma direção aleatória ($\phi(j,k,l)$), gerada pela rotação da bactéria, onde *j* é o índice do passo de *chemotaxis*, *k* é o índice do passo de reprodução e *l* é o índice do processo de eliminação e dispersão.

$$\theta'(j+1,k,l) = \theta'(j,k,l) + C(i) * \phi(j,k,l) \tag{3.4}$$

Para a utilização de tal estratégia como uma forma de resolver problemas de otimização, deve existir uma fórmula que determine o custo de cada uma das posições, de forma a determinar qual a melhor posição. O custo ($J(i,j,k,l)$) de uma determinada bactéria *i* é determinado segundo a equação,

$$J(i,j,k,l) = J(i,j,k,l) + J_{cc}(\theta'(j,k,l), P(j,k,l)) \tag{3.5}$$

onde θ' é a posição da bactéria sendo analisada, P é a população de bactérias para os determinados j, k e l, e J_{cc} é o valor de atratividade/repulsão existente entre a bactéria e os outros indivíduos da população.

Através desta equação pode-se ver que o custo de uma determinada posição ($J(i,j,k,l)$) é também afetado pela atratividade e pela repulsão existente entre as diversas bactérias existentes na população ($J_{cc}(\theta'(j,k,l), P(j,k,l))$).

Um passo *chemotactic* corresponde à rotação, movimentação e determinação do custo da posição para cada uma das bactérias existentes na população. Após a execução de um determinado número de passos *chemotactic* (N_c) um passo de reprodução ocorre.

Neste passo de reprodução, ordenam-se as bactérias através do seu custo acumulado, de forma decrescente, durante os passos *chemotactic*. A metade inferior desta lista morre, ou seja, as bactérias que não conseguiram juntar nutrientes suficientes para se reproduzirem, e a metade superior divide-se em duas novas bactérias, localizadas na mesma posição, ou seja, as bactérias que conseguiram juntar nutrientes suficientes se reproduzem.

3.2.4. Pseudocódigo

Na Fig. 3.2, é apresentado o algoritmo da teoria de *chemotaxis* em pseudocódigo.

```
Iniciar população;
PARA CADA ciclo de eliminação e dispersão{
        PARA CADA ciclo de reprodução{
                PARA CADA ciclo de chemotaxis{
                        Avaliar os indivíduos;
                        Determinar uma direção aleatória;
                        Mover o indivíduo naquela direção;
                        Continuar o movimento enquanto for vantajoso;
                }
                Calcular a saúde dos indivíduos;
                Matar os indivíduos com pior saúde;
                Duplicar os indivíduos com melhor saúde;
        }
        Eliminar e dispersar as bactérias;
}
```

Figura 3.2: Pseudocódigo para a colônia de bactérias.

Nota-se pelo pseudocódigo da Fig. 3.2 que o algoritmo de colônia de bactérias é composto inicialmente por um ciclo de eliminação e dispersão. Dentro deste ciclo, existe um outro que é responsável pela reprodução das bactérias. Dentro deste existe ainda um terceiro que é responsável por determinar a direção na qual cada bactéria se desloca e a determinação do período de deslocamento obtendo assim a sua posição após a execução deste ciclo e também por analisar os custos destas posições.

O ciclo de reprodução é responsável por determinar quais delas devem se reproduzir e quais devem ser exterminadas após as movimentações exercidas no ciclo 3, através da análise da saúde das bactérias, ou seja, dos custos das posições percorridas pelas mesmas ao longo da sua movimentação. O primeiro ciclo é responsável por eliminar algumas bactérias, através de uma probabilidade e posicioná-la em outra posição do espaço analisado.

3.3 Colônia de Formigas

O algoritmo de otimização chamado colônia de formigas é baseado no comportamento natural das colônias de formigas. O primeiro trabalho referente a este algoritmo foi descrito por Colorni *et al.* em 1991 (Colorni, *et al.*, 1991).

3.3.1. Comportamento de Colônias de Formigas

As formigas são insetos sociais que vivem em colônias e que, devido a sua colaboração mútua, são capazes de mostrar comportamentos complexos e realizar tarefas difíceis do ponto de vista de uma formiga individual (Cordón, *et al.*, 2002).

É correto afirmar que as formigas passam boa parte de seu tempo buscando comida. Sabe-se também que as formigas são praticamente cegas e aparentemente vagam sem destino. Porém, quando uma formiga encontra comida e a carrega para o seu ninho, um grande número de formigas logo chegará ao local onde a primeira encontrou a comida, passando a ter um grande número de formigas carregando comida daquele local após um curto espaço de tempo.

Outro fato a ser considerado em relação às formigas é a capacidade das mesmas conseguirem superar obstáculos colocados em seus caminhos, sendo que a maioria das formigas encontrará o caminho mais curto para superar o obstáculo.

A descoberta de E. O. Wilson (1990) que as formigas se comunicavam através de sinais químicos permitiu a compreensão de como as formigas chegavam ao local onde existem os depósitos de comida, após a descoberta do mesmo por uma formiga qualquer, e também de como as mesmas conseguem superar obstáculos colocados em seu caminho.

As formigas depositam marcadores químicos, denominados feromônios, no solo enquanto as mesmas se locomovem. O feromônio é depositado em quantidades variáveis e serve como uma trilha para as outras formigas, ou seja, o feromônio serve para comunicar indiretamente o seu caminho para as outras formigas.

O comportamento de uma formiga é simples. A mesma se locomove em uma direção aleatória até encontrar uma trilha de feromônio. A seguir cabe à formiga decidir se vai ou não seguir a trilha encontrada, esta decisão é baseada na quantidade de feromônio encontrada na trilha. Caso a formiga decida seguir a trilha, ela reforça a trilha de feromônio, ou seja, quanto mais formigas seguirem a trilha maior a probabilidade que as outras formigas sigam esta trilha (Spaulding, 1998).

Uma vez que se entenda como funcionam as trilhas de feromônio torna-se óbvio como as formigas descobrem a comida e como as mesmas cooperam entre si para transportá-la. De forma a entender como as formigas superam obstáculos que venham a aparecer na sua trajetória, está mostrada a Fig. 3.3, em um esquema em que os números representam as unidades de distância a serem percorridas em cada um dos segmentos de trajetória.

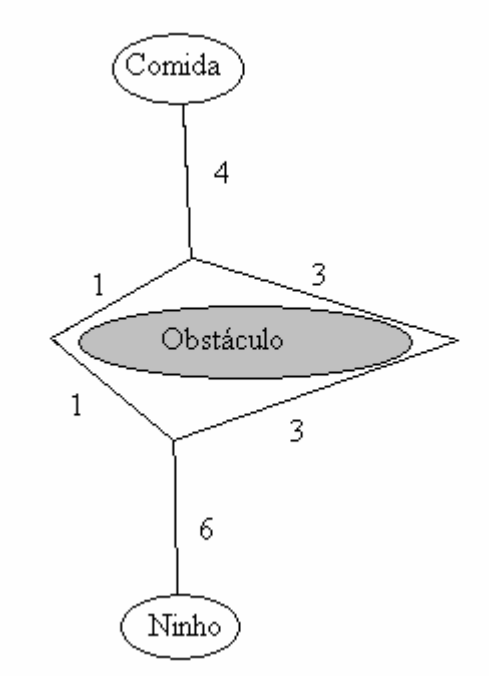

Figura 3.3: Obstáculo presente no caminho das formigas (Spaulding, 1998).

Quando uma formiga se deslocando a partir do ninho atinge um determinado obstáculo em seu caminho, não existe nenhuma trilha de feromônio a ser seguida, restando a ela então decidir por um dos caminhos aleatoriamente, seguindo toda a trajetória necessária para retornar à trilha previamente existente. Uma vez atingida esta trilha, a formiga continua a segui-la até atingir o seu destino (comida). Após atingir o seu destino, a formiga retornará pelo mesmo caminho (para retornar ao ninho), atingindo novamente o obstáculo. Porém, quando a mesma atingir este obstáculo, já existirão trilhas de feromônio pelos dois caminhos possíveis e a formiga optará pelo caminho que possuir a maior concentração de feromônio. Lembrando que as formigas apenas depositam feromônio na trilha durante seu retorno ao ninho. Passaremos a verificar agora porque um dos caminhos possíveis tende a possuir mais feromônio que o outro.

Suponha que as formigas depositem feromônio a uma taxa constante e também que se mova a uma velocidade constante. Analisando a Fig. 3.3, percebe-se que caso as formigas sigam o caminho da esquerda as mesmas terão que percorrer uma distância total de 12 unidades e caso as mesmas sigam o caminho da direita, percorrem uma distância total de 16 unidades. Supondo a existência de apenas duas formigas percorrendo a trilha de feromônio, cada uma seguindo um dos caminhos ao redor do obstáculo, em um determinado período de

tempo, a formiga que contorna o obstáculo seguindo o caminho mais curto percorre mais vezes o caminho entre o ninho e a fonte de alimentos. Com isso, existe uma maior concentração de feromônio ao longo do caminho mais curto ao redor do obstáculo, permitindo que as formigas que venham a seguir consigam saber qual dos caminhos a mesma devem percorrer.

Na Fig. 3.4, pode-se perceber o processo de produção de feromônio para dois caminhos possíveis.

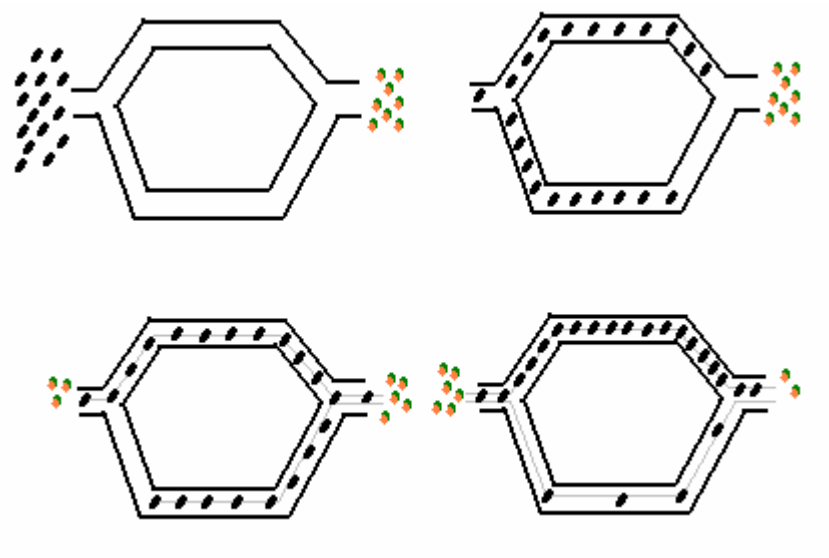

Figura 3.4: Seleção do menor caminho utilizando a trilha de feromônio (Cordón, *et al.*, 2002).

Um aspecto importante a ser mencionado é a possibilidade de exaustão das fontes de alimento. Uma vez que a fonte de alimentos seja exaurida, as formigas não mais percorrerão aquele caminho devido à evaporação do feromônio da trilha, este é um aspecto importante no comportamento das formigas (Spaulding, 1998).

A observação da capacidade das formigas encontrarem o menor caminho levou Colorni *et al.* (Colorni, *et al.*, 1991) a desenvolverem um algoritmo de otimização baseado no comportamento das formigas.

3.3.2. Algoritmo

O primeiro algoritmo desenvolvido sobre *Ant Colony Optimization* (ACO) foi o *Ant System* (AS), que é explicado nesta seção. Ele foi refinado ao longo dos anos através de melhoramento de desempenho por várias heurísticas.

Os algoritmos ACO utilizam diversos agentes, denominados formigas, que constroem soluções iterativamente para um determinado problema de otimização. A solução construída pelas formigas é guiada pelas trilhas de feromônio e por informações heurísticas sobre o problema. Em princípio, os algoritmos ACO podem ser aplicados a problemas de otimização combinatorial (Cordón, *et al.*, 2002).

As formigas constroem soluções candidatas iniciando com uma solução vazia e então adicionando componentes da solução de maneira iterativa, até que uma solução candidata completa seja gerada. Uma vez que a construção da solução esteja concluída, as formigas realizam um reforço nas soluções que elas construíram através do depósito de feromônio nos componentes da solução utilizados na sua solução, proporcional à qualidade da solução (Stützle e Hoos, 2000).

a. *Ant System*

O *Ant System* é um processo heurístico que pode ser utilizado para a solução de problemas de otimização que possam ser representados através de grafos. O processo se baseia no comportamento natural das formigas. O problema mais utilizado para o estudo deste algoritmo é o problema do caixeiro viajante (*Travelling Salesman Problem* – TSP) (Dorigo, *et al.*, 1996).

Este algoritmo possui um determinado número de formigas que percorrem os nós possíveis do grafo e a cada novo nó do grafo as formigas escolhem entre os nós conectados ao nó atual para determinar qual será o próximo nó. A probabilidade de escolha do nó subseqüente depende da distância àquele nó e também da quantidade de feromônio naquele caminho.

O comportamento das formigas pode ser resumido em três premissas básicas. São elas: (i) as formigas tendem a escolher os caminhos que possuírem uma maior quantidade de feromônio e, como as formigas liberam feromônio, (ii) os caminhos escolhidos tendem a acumular cada vez mais feromônio com o tempo, e (iii) assume-se que as formigas comunicam-se apenas através do feromônio.

A equação (3.6) determina a probabilidade de uma formiga *k*, presente no nó *i*, se deslocar para o nó *j*.

$$P_{ij}^{k} = \frac{\left[\tau_{ij}(t)^{\alpha}\right]\left[\frac{1}{D(i,j)}\right]^{\beta}}{\sum_{h \in Permitidos_{k}(t)} \left[\tau_{ih}(t)^{\alpha}\right]\left[\frac{1}{D(i,h)}\right]^{\beta}} \tag{3.6}$$

A equação (3.6) é válida para os caminhos entre *i* e *j* que são permitidos. Para os caminhos não permitidos, a probabilidade é 0. Na equação (3.6), $\tau_{ij}(t)$ representa o nível de concentração de feromônio existente ao longo do caminho entre *i* e *j*, $D(i,j)$ representa a distância entre os nós *i* e *j*, α e β são as constantes de controle relativas ao nível de importância do feromônio e da distância a ser percorrida entre os caminhos. A probabilidade de a formiga *k* ir para um determinado nó *j* é calculada em relação a todos os caminhos possíveis de serem percorridos por ela, a partir do nó *i*. Esta fórmula garante que as formigas seguirão o caminho com maior quantidade de feromônio e a menor distância entre os nós (Kristensen, 2000).

Como pode ser percebido, a quantidade de feromônio em um determinado caminho é importante para que as formigas decidam qual dos diversos caminhos possíveis estas irão percorrer. Portanto, é analisado como a concentração de feromônio é alterada ao longo do tempo, segundo a equação (3.7).

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \tag{3.7}$$

A equação (3.7) é responsável por determinar a concentração de feromônio no caminho entre os nós *i* e *j*, no próximo instante de tempo, sendo que ρ é a constante que governa a taxa de evaporação de feromônio, sendo que $\rho \in [0,1]$ e $\Delta\tau_{ij}$ é o termo responsável pela adição de feromônio por formigas que passem pelo caminho, podendo ser determinado pelas equações (3.8) e (3.9).

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{3.8}$$

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \tag{3.9}$$

A equação (3.9) calcula a quantidade de feromônio que a formiga k adicionou ao caminho entre os nós i e j, sendo que Q é uma constante e L_k é a distância entre os nós i e j. Caso a formiga não tenha passado por aquele caminho, o valor de $\Delta\tau_{ij}$ é tomado como zero.

Com isso, a quantidade de feromônio adicionado à trilha é inversamente proporcional à distância percorrida, ou seja, as rotas menores são mais atrativas que as rotas longas. Com o tempo, as formigas passarão a percorrer as mesmas rotas, convergindo a busca a uma determinada rota.

O algoritmo *Ant System* pode ser resumido pelo pseudocódigo apresentado na Fig. 3.5.

```
Iniciar a população;
FAÇA{
        Selecione a próxima posição para cada formiga;
        Calcule a distância percorrida;
        Calcule a quantidade de feromônio;
}ENQUANTO critério de parada não atingido
```

Figura 3.5: Pseudocódigo para o *Ant System*.

b. Algoritmo para Otimização Contínua

O algoritmo apresentado na seção anterior serve para otimizar problemas combinatórios. Porém, quando se deseja realizar a otimização de uma função real em um espaço contínuo, deve-se realizar algumas alterações no algoritmo. O primeiro trabalho desenvolvido neste sentido é apresentado em (Monmarché, *et al.*, 1999).

O processo de otimização neste algoritmo apresenta um ninho, localizado em uma determinada posição da região de busca do problema, e este ninho é composto por um determinado número de formigas.

Um dos parâmetros utilizados pelo algoritmo é o número de regiões de busca que cada formiga pode memorizar. O comportamento de cada formiga é baseado neste número. Caso a formiga ainda não tenha memorizado a quantidade máxima de regiões de busca possíveis, esta gera uma nova região e a explora.

O processo de geração de uma nova região de busca é responsável por determinar uma nova região circular, dentro da qual a formiga procura alimentos. O ninho está no centro desta nova região de busca representado por um círculo cujo raio é um parâmetro do algoritmo.

O processo de exploração das regiões de busca é composto pela geração aleatória de uma solução dentro desta região de busca e pela comparação do valor da função objetivo neste ponto com a função objetivo no centro daquela região. Caso o valor da função objetivo no ponto seja menor (no caso de minimização) que o valor da função no centro da região de busca, modifica-se o centro da região de busca para aquele ponto e determina-se que aquela formiga obteve sucesso na sua busca. Caso a formiga não obtenha sucesso, o número de fracassos daquela determinada região é incrementado em 1. Quando o número de fracassos de uma determinada região atingir um limite (parâmetro do algoritmo), a região é esquecida e uma nova região é criada.

Voltando ao comportamento da formiga, caso a mesma já tenha memorizado o número máximo de regiões de busca que esta pode memorizar, verifica-se se a mesma obteve sucesso no seu último processo de busca. Caso tenha obtido sucesso, a busca será novamente naquela região. Caso a formiga não tenha obtido sucesso a formiga escolhe aleatoriamente uma das regiões de busca memorizadas e, em seguida, a explora.

Uma vez que todas as formigas tenham realizado os seus procedimentos de busca por um determinado número de iterações (parâmetro do algoritmo), é selecionado o ponto que possui o menor valor da função objetivo (no caso de minimização) encontrado até o momento por um dos centros das regiões de busca. Neste caso, é então avaliado se este ponto possui um valor da função objetivo menor que o valor da função no ninho. Caso afirmativo, a posição do ninho é modificada para este novo ponto.

Caso a posição do ninho não se altere, é utilizado um artifício para modificar o processo de otimização global para um processo de otimização local. Neste caso, o raio ao redor do ninho que as formigas podem gerar as suas regiões de busca, e também os raios das regiões de busca, são reduzidos a cada vez que a posição do ninho é modificada, por um fator de atenuação (parâmetro de projeto do algoritmo). O processo de alteração da posição do ninho é realizado até que o número máximo de alterações seja alcançado.

O pseudocódigo do algoritmo de otimização para problemas contínuos é mostrado na Fig. 3.6.

```
Iniciar posição do ninho;
Iniciar região de exploração;
FAÇA{
        Simular comportamento das formigas;
        Atualizar posição do ninho;
        Atualizar a região de exploração;
} ENQUANTO critério de parada não atingido;
```

Figura 3.6: Pseudocódigo para a otimização de funções reais por colônia de formigas.

Na Fig. 3.7, pode-se ver o pseudocódigo para o comportamento das formigas utilizado no algoritmo de Colônia de Formigas.

```
PARA CADA formiga{
        Criar regiões de busca;
        Explorar regiões de busca;
}
```

Figura 3.7: Pseudocódigo para o comportamento das formigas.

3.4 Sistemas Imunológicos Artificiais

Como já pode ser percebido ao longo desta dissertação, a biologia tem servido muitas vezes como fonte de inspiração para o desenvolvimento de modelos computacionais e métodos de resolução de problemas. O sistema imunológico é uma destas fontes de inspiração que têm atraído atenção. Como resultado disto, os sistemas imunológicos artificiais (SIAs) surgiram. A seguir, é dada uma introdução básica sobre os sistemas imunológicos existentes na natureza (Dasgupta, *et al.*, 2003).

3.4.1. Sistema Imunológico Biológico

O sistema imunológico é responsável pela defesa dos animais contra constantes ataques de microorganismos. O sistema imunológico é fundamental para a sobrevivência dos animais e, por isso, precisa atuar de forma eficiente. A arquitetura do sistema imunológico é composta por três níveis de defesa:

(i) as barreiras físicas, composta pela pele e mucosas;

(ii) pelo sistema imune inato; e

(iii) pelo sistema imune adaptativo.

O sistema imune biológico envolve distintos agentes que interagem entre si, com a função de proteger o organismo de danos causados por antígenos. Para que o sistema imunológico funcione corretamente, este deve estar apto a distinguir entre as células pertencentes ao organismo (definidas como próprias) e as células que não pertencem ao organismo (definidas como não-próprias). Um dos processos utilizados para esta discriminação entre próprias e não-próprias é a seleção negativa (Garrett, 2005).

Sob os pontos de vista tanto biológico quanto de engenharia, dois mecanismos são fundamentais em um sistema imunológico: mecanismos de aprendizagem e de memória, de forma a extrair informações dos agentes infecciosos e também armazenar estas informações para o caso de futuras infecções.

a. Mecanismos de Aprendizagem e Memória

Para que o sistema imunológico seja capaz de proteger nosso organismo, o reconhecimento antigênico não é suficiente. Também é preciso que haja recursos suficientes para realizar uma resposta imunológica efetiva contra os agentes patogênicos. A aprendizagem imunológica envolve o aumento do tamanho da população e afinidade dos elementos que reconheceram determinado antígeno. A eficiência da resposta adaptativa a encontros secundários pode ser consideravelmente aumentada através do armazenamento de células produtoras de anticorpos com alta afinidade àquele antígeno, denominadas de células de memória, de forma que se tenha um "grande" clone inicial nos encontros subseqüentes. Clones são cópias das células existentes na população.

Ao invés de "partir do começo" toda vez que um dado estímulo antigênico é apresentado, existe um ponto de partida adequado para defender o organismo contra aquele microorganismo. Esta estratégia garante que a velocidade e eficácia da resposta imunológica se aperfeiçoem após cada infecção. Este esquema é característico de uma estratégia de aprendizagem por reforço, onde o sistema está continuamente melhorando a capacidade de executar sua tarefa. Como pode ser observado na Fig. 3.8, o período de latência para que o organismo reaja a um antígeno se reduz após uma primeira presença daquele antígeno.

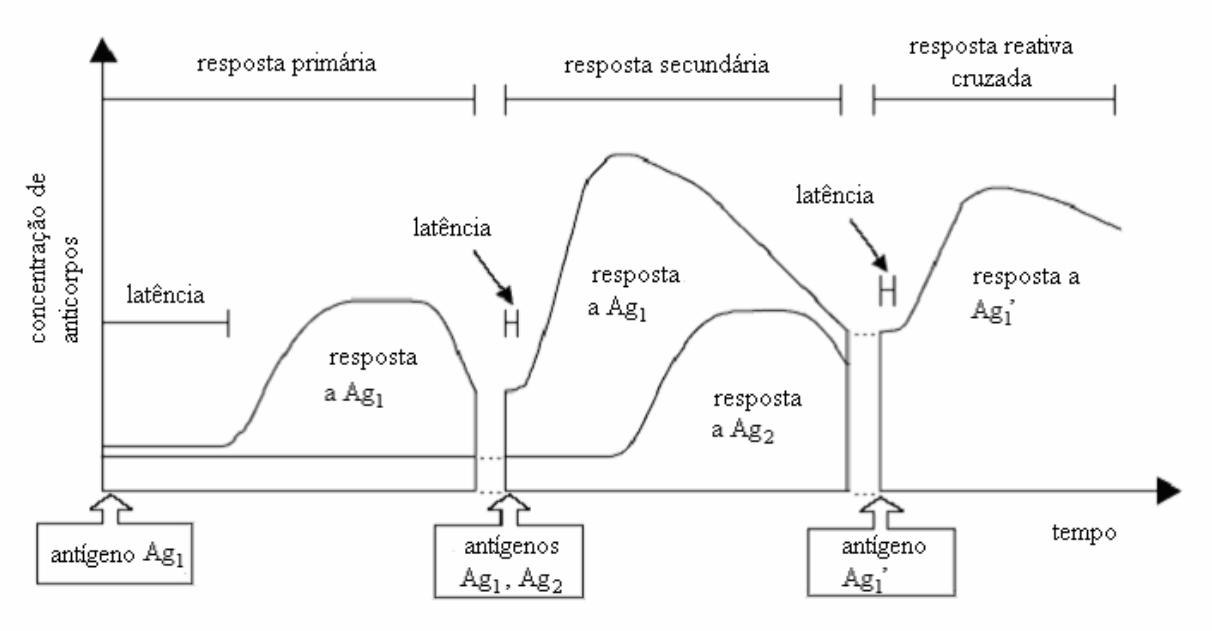

Figura 3.8: Resposta de um sistema imunológico a um antígeno (Silva, 2001).

Existem duas teorias básicas sobre os fenômenos de aprendizagem e memória imunológica, a teoria da seleção clonal e a teoria da rede idiotípica.

a.1. Princípio da Seleção Clonal

O princípio da seleção clonal está associado às características básicas de uma resposta imune adaptativa a um estímulo antigênico. Ele estabelece que apenas aquela célula capaz de reconhecer um determinado estímulo antigênico irá se proliferar, sendo, portanto, selecionada em detrimento das outras.

a.2. Teoria da Rede Idiotípica

Na teoria de rede idiotípica, as células estão interconectadas e as mesmas são responsáveis por estimular e suprimir umas as outras de forma a atingir a estabilização da rede (Dasgupta, *et al.*, 2003). Esta teoria é particularmente interessante para o desenvolvimento de ferramentas computacionais, pois ela fornece uma medida aproximada de propriedades emergentes como aprendizagem e memória, tolerância ao próprio, tamanho e diversidade de populações celulares. Estas propriedades não podem ser compreendidas partindo-se de uma análise de componentes isolados.

b. Mecanismo de Hipermutação Somática

O mecanismo de hipermutação opera seletivamente durante a proliferação celular a uma taxa próxima a $1x10^{-3}$ por par-base, um rápido aumento de mutações é necessário para uma rápida maturação da resposta imunológica, mas a maioria das mudanças introduzidas resultarão em anticorpos auto-reativos ou de baixa afinidade. Se uma célula que acabou de sofrer uma mutação capaz de aumentar sua afinidade antigênica continuar sendo mutada à mesma taxa durante as respostas imunológicas seguintes, então o acúmulo de variações indesejadas pode causar a perda das variações que levaram ao aumento da afinidade. Dessa forma, um curto pico de hipermutação somática, seguido de um intervalo para seleção e expansão clonal devem formar a base do processo de maturação. O mecanismo de seleção deve fornecer uma regulação do processo de hipermutação, que é dependente da afinidade do receptor. As células com receptores de baixa afinidade permanecem sofrendo mutações, enquanto as células com altas afinidades devem ter suas taxas de mutação controladas, e até mesmo inativadas (Silva, 2001).

3.4.2. Sistemas Imunológicos Artificiais

Os sistemas imunológicos artificiais (SIAs) podem ser definidos como sistemas adaptativos inspirados no sistema imunológico e aplicados à resolução de problemas (Silva, 2002). Para o desenvolvimento de um sistema imunológico artificial, são necessários alguns passos, entre eles a escolha de espaço de formas para os componentes do sistema, uma ou mais medidas de afinidade e o algoritmo imunológico. Apesar das técnicas de sistemas imunológicos artificiais serem relativamente antigas, datadas de mais de 15 anos atrás, apenas recentemente as mesmas estão recebendo maior atenção dos pesquisadores (Silva e Timmis, 2002).

Na modelagem de um sistema imunológico artificial, é importante considerar alguns princípios extraídos da imunologia, tais como (Carvalho e Freitas, 2000):

(i) Diversidade: melhora, de forma significativa, na robustez tanto de cada indivíduo como da população como um todo;

(ii) Distribuição: consiste de vários componentes interagindo localmente a fim de prover proteção a nível global, não existindo um controle central;

(iii) Tolerância a erros: a ocorrência de alguns erros de classificação não leva ao fracasso do sistema;

(iv) Dinamicidade: os indivíduos são criados, destruídos e colocados em circulação dinamicamente através do organismo, o que aumenta a diversidade espacial e temporal do sistema imune, permitindo o descarte de alguns elementos já dispensáveis;

(v) Auto-proteção: o mesmo mecanismo que protege o organismo também protege o sistema imune em si;

(vi) Adaptabilidade: ocorre o aprendizado de como reconhecer e responder a novos agentes estranhos.

a. Espaço de Formas

O espaço de formas é um formalismo utilizado para criar representações abstratas dos componentes de um sistema imunológico. A forma de uma célula ou molécula do sistema imunológico corresponde a todas as características necessárias para descrever quantitativamente as interações existentes entre a célula ou molécula e o ambiente ou outros elementos do sistema.

Existem quatro tipos principais de espaço de formas (Silva, 2002): Euclidianos, *Hamming*, Inteiros ou Simbólicos. No espaço de forma Euclidiano, os elementos do sistema são representados por vetores de valores reais. No espaço de forma de *Hamming* aos elementos do sistema são atribuídas *strings* construídas de um alfabeto finito. No espaço de forma inteiro, as células e moléculas são representadas por valores inteiros. E no espaço de forma simbólico, diferentes tipos de atributos são utilizados para representar todos os elementos, por exemplo, um valor inteiro e uma *string* determinando "cor".

b. Interações

Existem dois tipos principais de interações que podem ser realizadas por elementos pertencentes a um sistema imunológico artificial: a interação com o ambiente e a interação com outros elementos do sistema imunológico artificial. Na interação dos elementos com o ambiente, a afinidade entre a célula e o antígeno (elemento pertencente ao ambiente) é medida através de uma função que analisa o grau de semelhança entre ambos. No caso da interação de dois elementos do sistema imunológico artificial, a similaridade pode ser medida de acordo com o espaço de formas adotado. Existem diversos tipos de medidas de afinidade, que normalmente variam de acordo com o tipo de espaço de formas utilizado.

c. Algoritmo Imunológico

O último aspecto necessário para o desenvolvimento de um sistema imunológico artificial é a seleção do algoritmo imunológico. Existe uma grande variedade de algoritmos imunológicos, cada um voltado para aplicações em domínios diferentes. A inspiração para estes algoritmos é dos trabalhos teóricos sobre a imunologia e diversos processos que ocorrem dentro do sistema imunológico.

c.1. Medula Óssea

O modelo de medula óssea se baseia no fato de a medula óssea dos vertebrados ser a responsável pela geração das células sangüíneas. O material genético de uma molécula de anticorpo está armazenado em cinco bibliotecas separadas e distintas, sendo que duas delas são utilizadas para gerar a região variável da cadeia leve (V_L) e três para gerar a região variável da cadeia pesada (V_H). A produção de uma molécula de anticorpo se dá através da concatenação de componentes selecionados aleatoriamente a partir de cada uma das bibliotecas gênicas.

O modelo computacional mais simples de medula óssea é aquele que gera cadeias, ou vetores, de comprimento L utilizando um gerador de números pseudo-aleatórios. Para o caso de espaços reais, basta determinar o intervalo de pertinência do vetor m como, por exemplo, $m \in [0,1]^L$. No caso de espaços de *Hamming*, o vetor que representa a molécula m deve ser composto por elementos pertencentes a um alfabeto finito pré-definido. Para espaços inteiros, um algoritmo de permutação de L pode ser empregado.

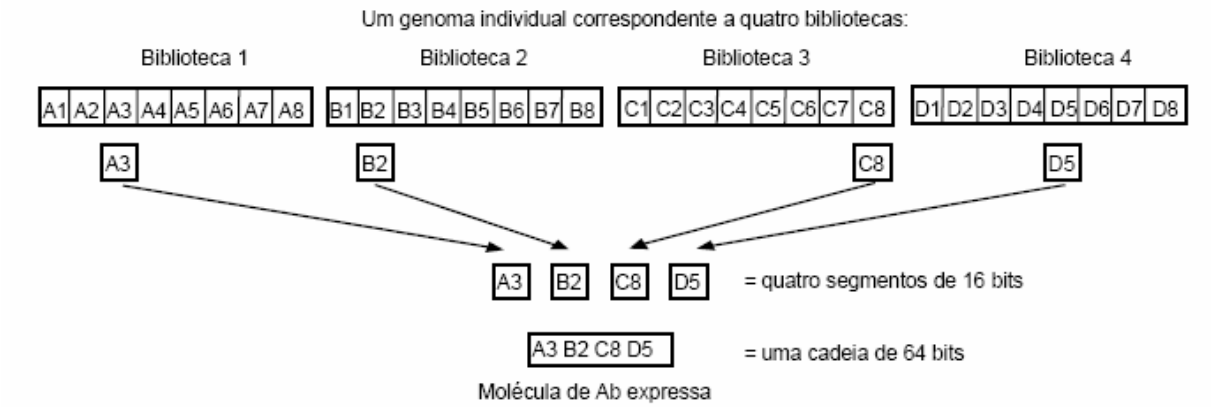

Figura 3.9: Processo de construção de um anticorpo a partir de bibliotecas genéticas (Silva, 2001).

A quantidade de bibliotecas, o tamanho dos segmentos genéticos e o comprimento L das moléculas serão definidos pelo projetista de acordo com o problema a ser tratado. Um sistema imunológico contendo l bibliotecas, cada uma com c componentes, pode produzir c^l moléculas de anticorpo distintas, ou seja, o repertório potencial de anticorpos é composto por c^l moléculas (Silva, 2001).

c.2. Seleção Negativa

Este algoritmo se originou do processo de seleção negativa, sendo responsável pela detecção de anomalias. O algoritmo é executado em duas fases: sensoriamento, geração do grupo de detectores; e monitoramento, detecção de elementos não-próprios.

Assumindo que o conjunto de elementos próprios é conhecido, o algoritmo de seleção negativa padrão é focado em desenvolver um conjunto de células imunológicas, conhecidas como detectoras, que reconhecem qualquer célula, exceto aquelas pertencentes ao conjunto das células próprias. Um pseudocódigo para este algoritmo é mostrado na Fig. 3.10.

```
Gerar conjunto de detectores;
Comparar as cadeias com o conjunto de detectores;
Determinar se elemento próprio ou não;
```

Figura 3.10: Pseudocódigo para o algoritmo de seleção negativa.

c.3. Seleção Clonal

Após a determinação de quais elementos são próprios e não-próprios, o sistema imunológico deve realizar uma resposta imunológica de forma a eliminar as substâncias não-próprias. A seleção clonal é o nome dado à teoria que explica como as células e moléculas imunológicas lidam com elementos invasores, denominados antígenos (Silva, 2002).

O princípio da seleção clonal estabelece que as células que reconhecem um determinado estímulo antigênico devem se proliferar, em detrimento das outras. Estas células capazes de reconhecer o antígeno se reproduzem assexuadamente de uma maneira proporcional ao seu grau de reconhecimento, Quanto maior o reconhecimento, maior o número de clones gerados. Durante o processo de reprodução, as células individuais sofrem mutação de forma que as mesmas se adaptem melhor ao antígeno reconhecido (aumentem a afinidade). Quanto maior a afinidade menor é a mutação sofrida.

Um pseudocódigo para o algoritmo de seleção clonal é mostrado na Fig. 3.11.

```
Gerar população inicial;
Selecionar as células com maior afinidade ao antígeno;
Clonar as células com maior afinidade;
Mutar os clones para modificar a afinidade com o antígeno;
```

Figura 3.11: Pseudocódigo para o algoritmo de seleção clonal.

O processo de funcionamento do algoritmo pode também ser representado pelo diagrama da Fig. 3.12.

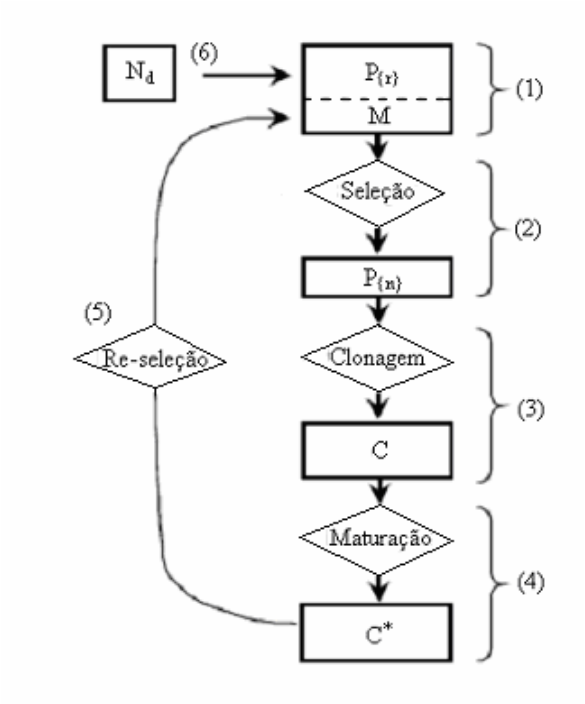

Figura 3.12: Diagrama de blocos para o algoritmo de seleção clonal (Silva, 2001).

No esquema da Fig. 3.12, os números de 1 a 6 apresentados são as ações realizadas pelo algoritmo:

1. Gerar candidatos à solução;
2. Determinar indivíduos com maior afinidade;
3. Reproduzir os melhores indivíduos;
4. Mutar a população de clones;
5. Re-selecionar a população de clones, com maior afinidade;
6. Substituir anticorpos por novos indivíduos.

c.4. Hipermutação Somática

A hipermutação somática é utilizada pelo sistema imunológico para criar e manter a diversidade e também para melhorar a afinidade dos anticorpos aos estímulos aplicados. Como o espaço de formas permite representar computacionalmente qualquer componente do sistema imunológico e seus ligantes através de vetores e/ou cadeias de atributos, é possível utilizar diversos algoritmos para inserir as variações na codificação destes elementos. No caso dos espaços de *Hamming*, uma posição da cadeia é escolhida aleatoriamente e seu elemento trocado por um outro elemento pertencente ao alfabeto. No caso dos espaços de formas Euclidianos, a mutação pode ser gerada através das equações (3.10) e (3.11) (Silva e Timmis, 2002):

$$c' = c + \alpha * randn(0,1) \tag{3.10}$$

$$\alpha = \left(\frac{1}{\beta}\right) * \exp(-f) \tag{3.11}$$

onde c é a célula original e c' é a célula mutada, β é o parâmetro que controla o grau de atenuação da exponencial inversa, α é o parâmetro que controla o grau de variação existente entre as células originais e as células mutadas e f que representa a afinidade do indivíduo ao estímulo aplicado.

3.4.3. Pseudocódigo para Otimização Contínua

Como o presente estudo está voltado para a aplicação em otimização de problemas, devem ser realizadas algumas modificações para adaptar estes algoritmos citados em algoritmos aplicáveis à otimização. Na Fig. 3.13, o pseudocódigo descrito para um problema de otimização contínua é apresentado.

```
Gerar população inicial;
FAÇA {
        Avaliar cada indivíduo;
        Clonar as células
        Mutar os clones;
        Suprimir indivíduos próximos entre si;
        Introduzir novos indivíduos;
} ENQUANTO critério de parada não atingido
```

Figura 3.13: Pseudocódigo para SIAs aplicados à otimização contínua.

O comportamento do algoritmo descrito é simples, inicialmente é gerada uma população aleatória de células, sendo a seguir otimizadas através de um processo de mutação proporcional à afinidade. Uma vez se que tenha atingido uma população relativamente estável, determinado através da variação dos *fitness*, as células passam a interagir umas com as outras em forma de rede e as células similares são eliminadas de forma a evitar redundância. Além disso, é inserida uma quantidade de células aleatórias na população atual e o processo de otimização reinicia.

Capítulo 4

Controle *Fuzzy*

Um propósito da utilização de um controlador *fuzzy* em um determinado processo é capturar o conhecimento de um operador sobre aquele determinado processo e convertê-lo em controle automático. Esta conversão é mais simples através de um controlador *fuzzy* ao invés de outros controladores, pois os controladores *fuzzy* utilizam variáveis lingüísticas. Além disso, estes controladores permitem certo grau de incerteza aos processos, possibilitando sistemas mais robustos, utilizados para modelar processos complexos (Yager e Filev, 1994). O controlador *fuzzy* é estudado em detalhes ao longo deste capítulo.

Para a mais simples compreensão de um controle lógico *fuzzy*, a partir de agora denominado de controle *fuzzy*, serão previamente apresentados um histórico básico, conceitos básicos sobre os sistemas *fuzzy* e alguns modelos de controladores *fuzzy*.

4.1 História

A necessidade de desenvolvimento dos sistemas *fuzzy* foi percebida no início da década de 60 por Lofti A. Zadeh. Este percebeu que as técnicas de análise tradicionais existentes na época eram excessivamente precisas para os problemas complexos existentes no mundo real. Daí surgiu a necessidade de uma nova forma de matemática, que foi desenvolvida por ele em 1961 (Yen e Langari, 1999):

> *"Nós necessitamos de uma forma de matemática radicalmente diferente, a matemática de quantidades nebulosas, que não são descritas em termos de distribuições probabilísticas. De fato, a necessidade de tal matemática está se*

tornando aparente..., para a maioria dos casos práticos os dados conhecidos a priori e também um critério através do qual é conhecido o desempenho do sistema desenvolvido pelo homem é julgado como distante de ser precisamente especificado ou possuindo distribuições probabilísticas conhecidas."

A idéia de graus de pertinência ocorreu a Zadeh em julho de 1964, quando o mesmo estava trabalhando em um problema de classificação de dados. Esta idéia de graus de pertinência se tornou essencial na teoria de conjuntos *fuzzy*, levando ao surgimento da tecnologia dos sistemas *fuzzy* (Zadeh, 1965).

Estes conceitos de lógica *fuzzy* sofreram duras críticas por parte da comunidade acadêmica. Apesar das críticas, diversos pesquisadores ao redor do mundo se tornaram seguidores de Zadeh. Enquanto Zadeh se preocupava em expandir os fundamentos da teoria dos conjuntos *fuzzy*, acadêmicos e cientistas, nos mais diversos campos, desde psicologia, sociologia, filosofia e economia a ciências e engenharia, exploravam este novo paradigma durante a primeira década após o seu nascimento (no período de 1965 a 1975) (Yen e Langari, 1999).

Vários conceitos importantes foram desenvolvidos neste período. Entre eles, o desenvolvedor de decisões de multi-estágios, relações de similaridades *fuzzy*, restrições *fuzzy* e limitações lingüísticas. No final da década de 70, alguns pequenos grupos de pesquisa universitários foram estabelecidos no Japão para estudar os sistemas *fuzzy*.

Em 1976, a primeira aplicação industrial de *fuzzy* foi desenvolvida por *Blue Circle Cement* e SIRA na Dinamarca. O sistema é o controlador de um forno para cimento que incorpora o conhecimento de operadores com experiência de forma a melhorar a eficiência. O sistema entrou em operação em 1982. A partir de então, a utilização de sistemas baseados em lógica *fuzzy* vem sofrendo um crescimento, até os dias atuais.

4.2 Lógica *Fuzzy*

O conceito de lógica *fuzzy* se baseia no pressuposto de que os fatos são relativos, por exemplo, ao uma pessoa perguntar se a temperatura de um determinado ambiente está baixa, média ou alta, esta pode receber respostas diferentes de acordo com a perspectiva da pessoa a que se está perguntando. Uma determinada sala a 20ºC pode estar com uma temperatura baixa para uma pessoa originária de Belém, que está acostumada a temperaturas em uma média de

30ºC diariamente. A temperatura pode estar quente para uma pessoa originária da Patagônia e pode estar em uma temperatura média para uma pessoa originária de Curitiba.

4.2.1. Conjuntos *Fuzzy*

Os conjuntos *fuzzy* se baseiam exatamente no conceito citado, quando se pergunta como se classificar um determinado dado, como, por exemplo, a temperatura, é óbvio que uma temperatura de -5º é uma temperatura baixa e é claro que uma temperatura de 45º é uma temperatura alta, mas quando uma temperatura de 15º ou 25º é tratada, como a classificação pode ser realizada? A resposta foi elaborada por Zadeh, que propôs um grau de pertinência para cada um dos conjuntos *fuzzy*. Com isso, a transição da pertinência para a não pertinência a um determinado grupo ocorre de forma gradual, e não de forma abrupta (Jantzen, 1998).

O grau de pertinência de um item é normalmente uma função que gera um número real entre 0 e 1, geralmente representado pela letra grega μ. Quanto maior este número, maior a pertinência daquele item ao grupo analisado, conforme mostrado na Fig. 4.1.

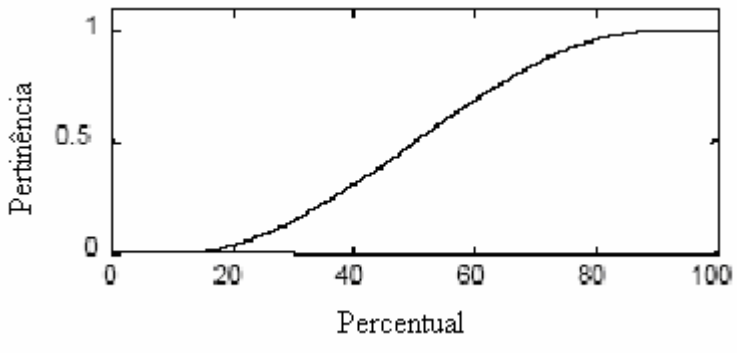

Figura 4.1: Descrição do grau de pertinência para um determinado conjunto *fuzzy*.

Apesar da definição de grau de pertinência para os conjuntos *fuzzy* realizada por Zadeh, este não definiu formalmente como obter os graus de pertinência para determinados conjuntos. Assim, não pode ser solucionado o problema de classificação de uma temperatura de 25ºC. Por isso, pode-se dizer que o grau de pertinência é uma medida precisa, porém subjetiva, ou seja, o grau de pertinência depende do contexto sendo utilizado (Jantzen, 1998).

4.2.2. Universo de Discurso

O universo de discurso, ou apenas universo, contém todos os elementos que podem ser levados em consideração. Como pode ser ressaltado, o próprio universo depende do contexto no qual o problema está situado.

Como forma de melhor descrever o problema de classificação das temperaturas, pode-se dividir o nosso universo de temperaturas em três graus de pertinência: baixa, média e alta, conforme mostrado na Fig. 4.2.

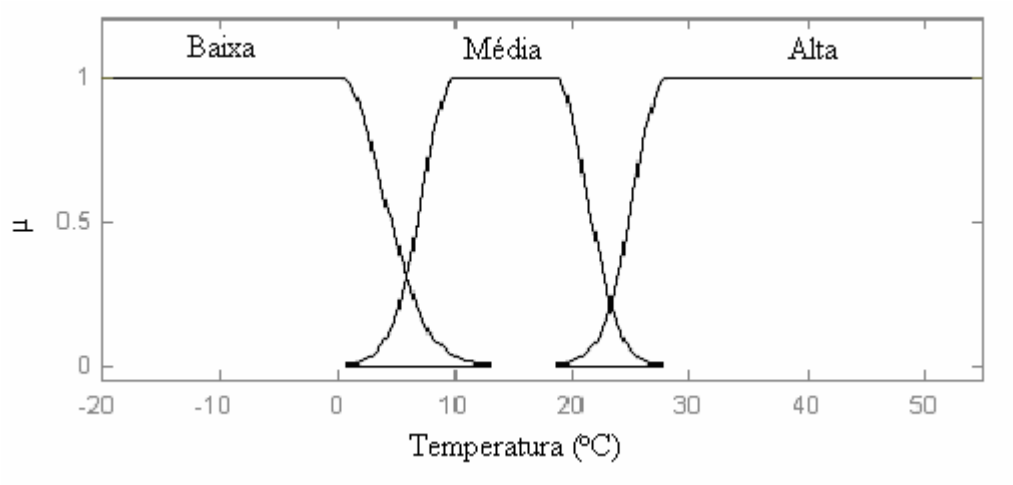

Figura 4.2: Universo para as temperaturas com três graus de pertinência.

A classificação do universo de discurso pode ser realizada em função de qualquer número de graus de pertinência, sendo que a utilização de mais graus aumenta a adequação da variável ao meio real. Isto ocorre, pois a utilização de um maior número de graus de pertinência faz com que a classificação seja mais exata.

4.2.3. Funções de Pertinência

Todos os elementos pertencentes ao universo de discurso estão relacionados a todos os graus de pertinência de algum modo, mesmo que a pertinência seja zero. A função que associa um número a todos os elementos do universo *(x)* é a função de pertinência $\mu(x)$.

Existem duas formas básicas de se representar as funções de pertinência, a contínua e a discreta (Jantzen, 1998).

a. Representação Contínua

Na forma de representação contínua, uma função de pertinência é gerada por uma função matemática. Exemplos destas formas de representação são as representações triangulares, trapezoidais, forma de sino (*bell-shaped*), gaussiana, entre outras. Estes exemplos podem ser vistos na Fig. 4.3 (a), (b), (c) e (d), respectivamente.

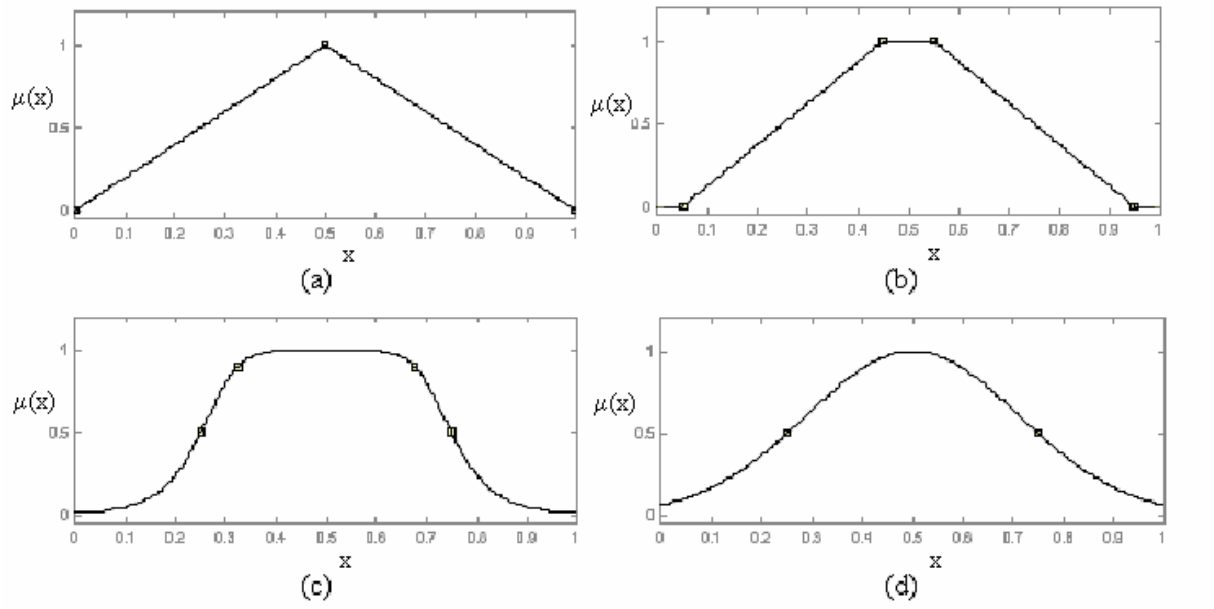

Figura 4.3: Exemplos de formas de funções de pertinência.

b. Representação Discreta

Quando é utilizada a representação discreta, tanto as funções de pertinência quanto o universo de discurso são pontos discretos em uma lista, ou seja, os mesmos são representados por um vetor de pontos. Para se obter estes vetores de pontos, são utilizados os conceitos de amostragem. A representação discreta é utilizada principalmente para reduzir o custo computacional.

4.2.4. Lógica

A lógica é um estudo da linguagem para efeitos de argumentação e persuasão. A lógica pode ser utilizada para avaliar o grau de coerência de uma seqüência de pensamentos. Na lógica *booleana*, cada proposição pode ser apenas Falsa ou Verdadeira. Já na lógica *fuzzy*, uma proposição pode ser falsa, verdadeira, ou então um valor intermediário entre falsa e verdadeira, algo similar a “talvez”. Neste caso, passa a se tratar de uma lógica multivalores.

4.3 Principais Modelos

Existem três tipos de modelos baseados em regras para a aproximação de funções: o modelo lingüístico de Mamdani, o Takagi-Sugeno-Kang (TSK) e o modelo aditivo de Kosko (Yen, 1999). O primeiro modelo desenvolvido foi o modelo de Mamdani (Mamdani e Assilian, 1975) e o modelo TSK foi apresentado em 1985 (Takagi e Sugeno, 1985).

4.3.1. Modelo de Mamdani

O modelo de Mamdani é um dos mais utilizados na prática. Neste modelo, a saída é composta pela superposição das saídas de cada uma das regras individuais. O modelo consiste de uma série de regras lingüísticas com formato:

$$\text{SE } X_1 \text{ é } A_1 \text{ e } X_2 \text{ é } A_2 \text{ e ... e } X_n \text{ é } A_n \text{ ENTÃO } Y \text{ é } C_1 \tag{4.1}$$

onde X_i são as variáveis lingüísticas de entrada, Y é a variável de saída e A_i e C_i são os conjuntos *fuzzy* para X e Y, respectivamente.

A característica básica do modelo de Mamdani é o fato dele utilizar conjuntos *fuzzy* nos conseqüentes das suas regras *fuzzy* (Delgado, 2002). A contribuição de uma determinada regra R_i para a saída do modelo de Mamdani é um conjunto *fuzzy*, cuja função de pertinência pode ser calculada pela equação (4.2),

$$\mu_{C_i'}(y) = (\alpha_{i1} \wedge \alpha_{i2} \wedge ... \wedge \alpha_{in}) \wedge \mu_{C_i}(y) \tag{4.2}$$

onde α_{ij} é o grau de relacionamento existente entre a variável x_j e a condição desta mesma variável na regra R_i, e μ_{C_i} é o grau de pertinência do conjunto C (saída), e o operador ^ é o operador *min*. Uma vez que a fórmula presente na equação (4.2) seja calculada para todas as regras existentes na base de regras do controlador. O passo seguinte é obter o conjunto de saída, a partir dos resultados obtidos, conforme a equação (4.3),

$$\mu_C(y) = \max\left\{\mu_{C_1'}(y), \mu_{C_2'}(y), ..., \mu_{C_L'}(y)\right\} \tag{4.3}$$

sendo que L é a quantidade de regras existentes na base de regras do controlador. Note que utilizando este modelo, a saída é um conjunto *fuzzy*. Para se determinar um valor de controle é

necessário utilizar uma técnica de defuzificação, que será vista na Seção 4.4.4 (Yen e Langari, 1999).

4.3.2. Modelo de Takagi-Sugeno-Kang (TSK)

O modelo de Takagi-Sugeno-Kang (TSK) foi introduzido por T. Takagi e M. Sugeno em 1984 (Takagi e Sugeno, 1985), cerca de uma década após o modelo da Mamdani. A principal motivação para o desenvolvimento deste modelo é reduzir o número de regras necessárias no modelo de Mamdani, especialmente para modelos complexos e de elevada dimensão.

Para se atingir este objetivo, o modelo TSK substitui os conjuntos *fuzzy* existentes no conseqüente (parte após o ENTÃO) das regras de Mamdani por uma equação linear das variáveis de entrada. De forma geral, as regras para um modelo TSK são conforme vistas na equação (4.4):

$$\text{SE } X_1 \text{ é } A_{i1} \text{ e } ... \text{ e } X_r \text{ é } A_{ir} \text{ ENTÃO } y = f_i(x_1,x_2,...,x_r) = b_{i0}+b_{i1}x_1+...+b_{ir}x_r \qquad (4.4)$$

onde f_i é o modelo linear e b_{ij} são parâmetros cujos valores pertencem aos números reais (Yen e Langari, 1999).

A inferência realizada pelo modelo TSK é uma interpolação entre todos os modelos lineares relevantes. O grau de relevância de um determinado modelo linear é determinado pelo grau de associação entre os dados de entrada e o subespaço *fuzzy* associado a este modelo linear. Estes graus de relevância serão os pesos associados no processo de interpolação.

A saída do modelo é determinada pela equação (4.5), onde α_i é o grau de relevância da regra R_i, e L é a quantidade de regras existente na base de regras do controlador.

$$y = \frac{\sum_{i=1}^{L} \alpha_i f_i(x_1, x_2, ..., x_r)}{\sum_{i=1}^{L} \alpha_i} \qquad (4.5)$$

O cálculo de α_i é realizado tanto pelo operador mínimo quanto pelo operador produto, como pode ser visto pelas equações (4.6) e (4.7).

$$\alpha_i = \min\left(\mu_{A_{i1}}(a_1), \mu_{A_{i2}}(a_2), ..., \mu_{A_{ir}}(a_r)\right) \tag{4.6}$$

$$\alpha_i = \mu_{A_{i1}}(a_1) \times \mu_{A_{i2}}(a_2) \times ... \times \mu_{A_{ir}}(a_r) \tag{4.7}$$

4.3.3. Modelo Aditivo Padrão (*Standard Additive Model* – SAM)

O modelo aditivo padrão (SAM) foi introduzido por B. Kosko em 1996. A estrutura deste modelo é idêntica à estrutura do modelo de Mamdani, porém, existem quatro diferenças entre os métodos de inferência destes dois modelos (Yen e Langari, 1999):

(i) SAM assume que as entradas são valores, enquanto que o modelo de Mamdani trata como entradas tanto valores quanto conjuntos *fuzzy*;

(ii) SAM utiliza o método de inferência de escala, ao invés do método visto na seção 4.3.1;

(iii) SAM utiliza a adição para combinar as conclusões das regras *fuzzy*, enquanto que o modelo de Mamdani utiliza o operador *max*;

(iv) SAM inclui a técnica de defuzificação por centróide (a ser vista na seção 4.4.4), enquanto que o modelo de Mamdani não força um método de defuzificação.

A forma das regras que compõem a base de regras de um controlador SAM deve ser segundo a equação (4.8).

$$\text{SE } X_1 \text{ é } A_1 \text{ e } X_2 \text{ é } A_2 \text{ e ... e } X_n \text{ é } A_n \text{ ENTÃO } Y \text{ é } C_1 \tag{4.8}$$

Com isso, a saída do modelo é calculada segundo a equação (4.9).

$$y = Centroide\left(\sum_{i=1}^{n} \mu_{A_i}(x_1) \times \mu_{A_i}(x_2) \times \ldots \times \mu_{A_i}(x_n) \times \mu_{C_i}(y)\right) \tag{4.9}$$

4.4 Controlador *Fuzzy*

Um controlador *fuzzy* é um sistema de controle baseado nos conceitos de lógica *fuzzy*. Um controle *fuzzy* pode ser simplificado pela frase “Controle com sentenças ao invés de com equações”.

Um controlador *fuzzy* típico, segundo o modelo de Mamdani, modelo a ser utilizado nos casos apresentados no Capítulo 5, é composto por regras do tipo *if-then* (se-então), seguindo o padrão descrito na equação (4.10).

$$\begin{aligned} &\text{SE } X_1 \text{ é } A_1 \text{ e } X_2 \text{ é } A_2 \text{ e ... e } X_n \text{ é } A_n \\ &\text{ENTÃO } Y_1 \text{ é } B_1 \text{ e } Y_2 \text{ é } B_2 \text{ e ... e } Y_m \text{ é } B_m \end{aligned} \tag{4.10}$$

Um controlador *fuzzy* é composto por basicamente quatro funções básicas. São elas: a fuzificação, a base de regras, o motor de inferência e a defuzificação. A estrutura básica de um controlador *fuzzy* pode ser vista na Fig. 4.4 (Jantzen, 1998).

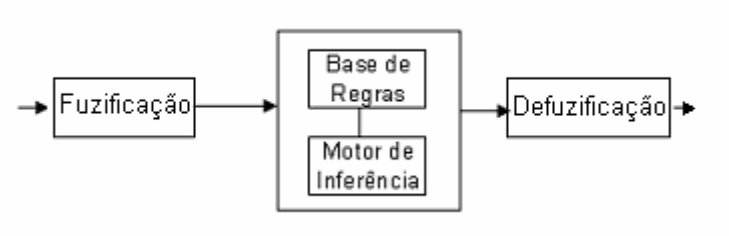

Figura 4.4: Estrutura de um controlador *fuzzy* (Jantzen, 1998).

4.4.1. Fuzificação

O bloco de fuzificação é o bloco responsável pela conversão dos dados de entrada no controlador (dados originários do processo sendo controlado) de valores escalares para valores "*fuzzy*", ou seja, este bloco é responsável por converter os números que aparecem na entrada do controlador (cujos valores devem estar dentro do universo do controlador) para um dos conjuntos *fuzzy* do controlador.

Para realizar esta conversão de escalar para variáveis *fuzzy*, são utilizadas as funções de pertinência de cada um dos conjuntos *fuzzy* do controlador, é calculada a função de pertinência de cada um dos conjuntos para a entrada, o conjunto cuja função de pertinência for maior será o conjunto daquela determinada entrada.

4.4.2. Base de Regras

A construção da base de regras é um fator crucial para o funcionamento de um controlador *fuzzy*. Esta etapa é também o passo mais difícil no desenvolvimento de um projeto de controlador *fuzzy* (Yager e Filev, 1994).

A base de regras é composta por diversas regras no formato *if-then*, podendo utilizar uma ou várias entradas e saídas. Cada uma das regras são compostas segundo a equação (3.1), onde $X_1, X_2, ..., X_n$ são as entradas do sistema, $A_1, A_2, ..., A_n$ são os conjuntos *fuzzy* pertencentes ao universo de discurso do problema. Deve-se ressaltar que o universo de discurso pode variar entre as variáveis; $Y_1, Y_2, ..., Y_m$ são as variáveis de saída do sistema e $B_1, B_2, ..., B_m$ são os conjuntos *fuzzy* pertencentes ao universo. Na Fig. 4.5 pode-se ver a base de regras para um exemplo do controle do erro e da variação do erro da saída do processo em relação ao sinal de referência como entradas do controlador, gerando como saída um sinal de controle.

1. Se erro é *Neg* e variação no erro é *Neg* então saída é *NB*
2. Se erro é *Neg* e variação no erro é *Zero* então saída é *NM*
3. Se erro é *Neg* e variação no erro é *Pos* então saída é *Zero*
4. Se erro é *Zero* e variação no erro é *Neg* então saída é *NM*
5. Se erro é *Zero* e variação no erro é *Zero* então saída é *Zero*
6. Se erro é *Zero* e variação no erro é *Pos* então saída é *PM*
7. Se erro é *Pos* e variação no erro é *Neg* então saída é *Zero*
8. Se erro é *Pos* e variação no erro é *Zero* então saída é *PM*
9. Se erro é *Pos* e variação no erro é *Pos* então saída é *PB*

Figura 4.5: Representação da base de regras de um controlador (Jantzen, 1998).

Neste exemplo os conjuntos *fuzzy* das variáveis erro e variação do erro são: *Neg* (negativo), *Zero* e *Pos* (positivo), e os conjuntos *fuzzy* para a saída são *NB* (negativo grande), *NM* (negativo médio), *Zero*, *PM* (positivo médio) e *PB* (positivo grande).

Como o propósito de simplificar a representação destas regras, existem duas formas alternativas, um formato relacional, apresentado como exemplo na Tabela 4.1 e um formato de tabela, apresentado na Tabela 4.2. O exemplo utilizado nestas tabelas tem como entradas o erro e a variação no erro, gerando um sinal de controle.

Tabela 4.1: Representação da base de regras em forma relacional (Jantzen, 1998).

Erro	Variação no Erro	Saída
Neg	Neg	NB
Neg	Zero	NM
Neg	Pos	Zero
Zero	Neg	NM
Zero	Zero	Zero
Zero	Pos	PM
Pos	Neg	Zero
Pos	Zero	PM
Pos	Pos	PB

Tabela 4.2: Representação da base de regras em forma de tabela (Jantzen, 1998).

		Variação no Erro		
		Neg	Zero	Pos
Erro	Neg	NB	NM	Zero
	Zero	NM	Zero	PM
	Pos	Zero	PM	PB

Estas formas de representação foram desenvolvidas visando tornar mais compacta a representação de problemas complexos. A forma da Tabela 4.2 também é utilizada de modo a verificar a completude da base de regras. Caso uma das células da tabela fique vazia, o controlador não está abrangendo todas as possíveis configurações de entradas.

4.4.3. Motor de Inferência

O motor de inferência de um controlador é composto por duas tarefas básicas. São elas: determinar o grau que cada regra existente na base de regras se adapta às condições de entrada do controlador (casamento); e retirar conclusões utilizando as entradas e a base de regras (passo de inferência).

a. Casamento – *Matching*

Para o processo de casamento, existem dois passos básicos: a combinação das entradas do controlador com as premissas das regras; e a determinação de quais regras estão ativadas.

Como já foi dito, o primeiro passo consiste de combinar os conjuntos *fuzzy* originários da fuzificação com os conjuntos *fuzzy* utilizados em cada uma das premissas das regras, além de determinar a função de pertinência deste conjunto gerado.

No segundo passo, valores de pertinência são gerados para as premissas de cada uma das regras da base de pertinência. Esta função de pertinência representa o grau de exatidão com que cada premissa da regra se relaciona com a entrada.

b. Passo de Inferência

O passo de inferência é responsável por determinar a saída do controlador de acordo com os dados obtidos no processo de casamento. Para isso, são somadas as saídas referentes à função de pertinência de cada uma das regras presentes na base de regras do controlador.

4.4.4. Defuzificação

O conjunto de saída obtido pelo motor de inferência deve ser convertido em um número para que possa ser enviado para o processo sendo controlado. O processo de conversão deste conjunto *fuzzy* para um número é denominado de processo de defuzificação. Existem diversos métodos de defuzificação (Jantzen, 1998), sendo que os métodos de centro de gravidade, média da área, média do máximo e média mais à esquerda e mais à direita são apresentadas a seguir.

a. Centro de Gravidade

A saída u é calculada através do centro de gravidade com conjunto *fuzzy*, seguindo a equação (4.11).

$$u = \frac{\sum_i \mu(x_i) * x_i}{\sum_i \mu(x_i)} \tag{4.11}$$

ou seja, o sinal de saída é uma média ponderada dos elementos presentes no conjunto *fuzzy*.

b. Média da Área

Este método considera a linha vertical que divide a área sob a curva em duas áreas iguais, seguindo a equação (4.12).

$$u = \left\{ x \mid \int_{Min}^{x} \mu(x)dx = \int_{x}^{Max} \mu(x)dx \right\} \tag{4.12}$$

onde *Min* é o ponto mais à esquerda do conjunto *fuzzy* e *Max* é o ponto mais à direita do conjunto.

c. Média do Máximo

Uma aproximação intuitiva é escolher o ponto de máximo, ou seja, o ponto de maior pertinência, dentro do conjunto *fuzzy*. Porém, podem existir vários pontos de máximo. Portanto, uma prática comum é considerar a média de todos os pontos de máximo existentes no conjunto. Deve-se ressaltar que tal método desconsidera a forma do conjunto.

d. Máximo mais à Esquerda e Máximo mais à Direita

Outra possibilidade é adotar o máximo existente no conjunto *fuzzy*, ou seja, o ponto com maior função de pertinência, como a saída do controlador. No caso da existência de vários pontos de máximo, considera-se o ponto mais à direita, ou mais à esquerda do conjunto, que tenha o valor máximo, de acordo com a abordagem utilizada.

Capítulo 5

Estudo de Caso

Neste capítulo da dissertação, são descritos os casos a serem estudados. No presente documento, dois estudos de casos são apresentados. O primeiro estudo de caso é a otimização de uma trajetória planejada entre dois pontos, sendo que o ambiente é conhecido a priori, e o segundo estudo de caso diz respeito à otimização de um controlador *fuzzy* de um robô móvel utilizado para explorar um ambiente previamente desconhecido.

5.1 Planejamento de Trajetórias

Principalmente no último século, a automação tem se tornado um fenômeno de crescimento extremamente rápido, de forma a impactar no cotidiano das pessoas. Atualmente, os robôs têm ocupado a maior parte deste nicho. Portanto, o desenvolvimento de robôs de navegação autônoma é de extrema importância (Sedighi, *et al.*, 2004).

A navegação é uma tarefa importante para qualquer robô móvel. Porém, antes que um robô possa afetar de uma maneira sensível o ambiente no qual está situado, o mesmo deve estar situado no local certo e na hora certa. O problema de navegação pode ser dividido em importantes tarefas: localização, planejamento de trajetos e execução da trajetória planejada (Baltes e Hildreth, 2001).

O problema do planejamento de trajetórias é normalmente formulado como se segue: "Dado um robô e a descrição de um ambiente, planeja-se um trajeto entre duas localizações específicas que é livre de colisão e satisfaz certo critério de desempenho" (Xiao, *et al.*, 1997).

A principal dificuldade em encontrar o trajeto ótimo vem do fato de que os métodos analíticos são complexos para serem utilizados em tempo real e métodos de busca

enumerativos são prejudicados pelo tamanho do espaço de busca (Sierakowski e Coelho, 2004; Sierakowski e Coelho, 2005).

Existem diversas classificações de planejamentos de trajetos, entre elas a classificação de planejamento de trajetos global e local, ou também chamado estático e adaptativo. O planejamento de trajetos global, ou estático, requer que todo o ambiente seja conhecido *a priori* e que o ambiente seja estático. Por outro lado, no planejamento de trajetos local, ou adaptativo dinâmico, o planejamento de trajetórias vai sendo realizado ao longo da movimentação do robô. Com isso, o algoritmo é capaz de produzir novas trajetórias de forma a responder a situações inesperadas enquanto engajado no procedimento de resolução do problema (Sedighi, *et al.*, 2004; Sierakowski e Coelho, 2004).

O problema de planejamento de trajetos é adotado como um problema de otimização nesta dissertação. Neste caso uma seqüência de posições que levem o robô da posição inicial à posição de destino são otimizadas. O planejador deve ser capaz de localizar uma série de posições que evitem a ocorrência de colisões entre o robô móvel e os obstáculos existentes no ambiente e, de forma a ser eficiente, o planejador deve ser capaz de minimizar a distância a ser percorrida pelo robô (Sierakowski e Coelho, 2004).

No presente estudo de caso, apenas o problema de otimização da trajetória é considerado, sem levar em conta o problema do controle do robô móvel. O problema de otimização da trajetória considera um espaço bidimensional, onde o robô móvel *R* é representado pelas suas coordenadas cartesianas *(x, y)*, as posições inicial (x_0, y_0) e final (x_{np}, y_{np}) do robô são conhecidas, onde *np* é um parâmetro de projeto que representa a quantidade de vezes que a direção da trajetória do robô é modificada.

Os obstáculos são considerados como circulares no plano de movimentação do robô e são representados pelas coordenadas cartesianas de seu centro e pelo seu raio. O problema de otimização consiste em minimizar a função custo apresentado na equação (5.1).

$$f(x,y) = \alpha * d_{obj} + \lambda * n_0 \tag{5.1}$$

sendo que o *fitness* é determinado pela equação (5.2).

$$fitness = \frac{1}{1 + f(x,y)} \tag{5.2}$$

Os termos α e λ são fatores de ponderação utilizados na formulação do problema, d_{obj} é a distância percorrida pelo robô, e pode ser determinada a partir da equação (5.3), e n_0 é o número de restrições que ocorrem no trajeto planejado (número de colisões ocorridas).

$$d_{obj} = \sum_{i=1}^{np+1} \sqrt{\left(x(i+1) - x(i)\right)^2 + \left(y(i+1) - y(i)\right)^2} \tag{5.3}$$

No problema sendo analisado, são considerados $\alpha = 1$, $\lambda = 200$, o valor de 200 foi selecionado por se tratar de um valor maior que a distância existente entre o ponto de origem e de destino, $x_i \in [0, 100]$ e $y_i \in [0, 100]$, e $np = 4$. A escolha por $np = 4$ foi originada por testes anteriores, onde esta opção foi a que apresentou a melhor relação custo/benefício entre boas soluções e custo computacional. Além disso, poderia ter se optado por um *np* adaptativo, opção que foi desconsiderada devido a estas simulações previamente realizadas.

O ponto de saída (x_0, y_0) do robô é considerado como sendo [0, 0] e o ponto de destino (x_f, y_f) é [100, 100], os centros e raios adotados para os obstáculos são apresentados na Tabela 5.1, estes obstáculos foram selecionados de forma a dificultar a obtenção da solução pelos algoritmos.

Tabela 5.1: Centros e raios dos obstáculos existentes no ambiente.

Obstáculo	**Posição (*x*, *y*)**	**Raio**
1	(50, 40)	08
2	(75, 75)	10
3	(50, 70)	10
4	(20, 20)	05
5	(40, 15)	06
6	(70, 10)	08
7	(75, 52)	04
8	(20, 60)	08
9	(30, 45)	04
10	(85, 50)	05
11	(15, 85)	06
12	(32, 85)	03

Na Fig. 5.1 é apresentado o ambiente no qual o robô móvel deve otimizar seu trajeto.

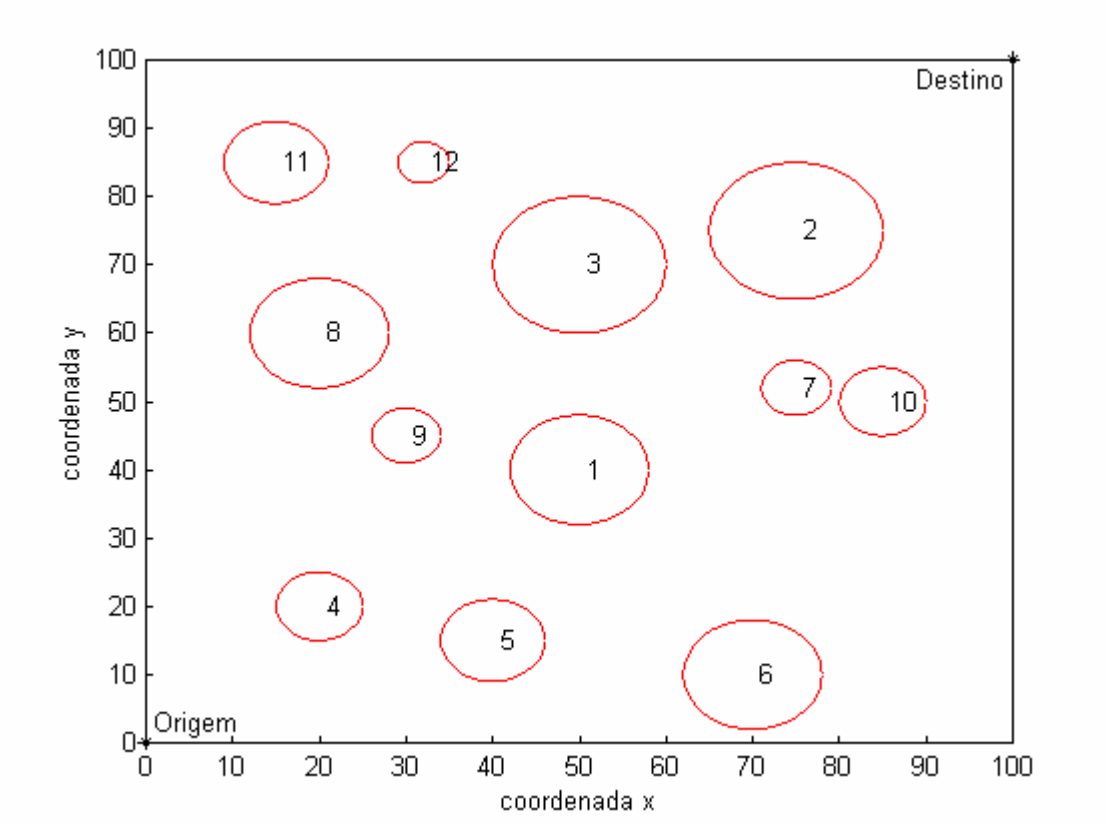

Figura 5.1: Ambiente onde será planejado o trajeto.

Como forma de análise de desempenho, além do *fitness* previamente descrito neste capítulo, serão utilizadas outras duas medidas de desempenho: o desempenho *on-line* e o desempenho *off-line*. O desempenho *on-line* está relacionado à média de todos os *fitness* de todos os indivíduos gerados até uma determinada geração, e o desempenho *off-line* refere-se à média dos melhores *fitness* de cada geração gerados até uma determinada geração, levando em consideração os experimentos realizados (De Jong, 1975; Bobel, 2003).

Os resultados obtidos para o presente estudo de caso e também para os estudos de casos a serem apresentados na seqüência são apresentados no Capítulo 6.

5.2 Otimização de Controladores *Fuzzy*

O presente estudo de caso diz respeito à otimização de um controlador *fuzzy* aplicado a um robô Khepera, com o intuito de exploração de um ambiente previamente desconhecido. De forma a facilitar a compreensão do estudo de caso, primeiramente será apresentada uma descrição do robô móvel Khepera. A seguir, é feita uma descrição da estrutura do controlador

fuzzy utilizado para controlar o Khepera. Será então apresentado o ambiente no qual o robô se locomove e, por fim, como ocorre a otimização deste controlador *fuzzy* através dos algoritmos de inteligência coletiva.

5.2.1. Khepera

O robô móvel Khepera é um robô em miniatura desenvolvido especialmente para ambientes acadêmicos. Este foi originariamente desenvolvido por Andre Guignard, Edoardo Franzi e Francesco Momdada do grupo LAMI da EPFL em Lausanne – Suíça (Köse, 2000), sendo atualmente comercializado pelo K-TEAM S.A. (K-TEAM Website, 2005).

A principal característica do robô móvel Khepera é o seu pequeno tamanho, apenas 55mm de diâmetro e 32mm de altura (Salomon, 2005), como pode ser visto na Fig. 5.2.

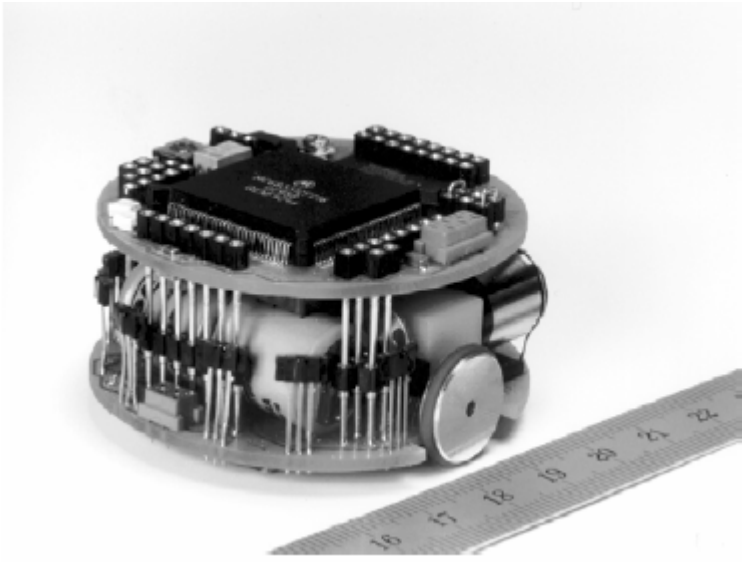

Figura 5.2: Módulo base do robô móvel Khepera (Lundgren, 2003).

O Khepera é composto por duas rodas localizadas uma em cada lado do robô, que permitem a movimentação do mesmo e o direcionamento do mesmo. Estas rodas são controladas de forma independente uma da outra, sendo que o controle é realizado através de dois motores. Cada um destes motores é limitado fisicamente às tensões entre -10 e 10V, permitindo então ao robô a velocidade máxima de 60 cm/s e a velocidade mínima de 2 cm/s, tanto para frente quanto para trás (Ludgren, 2003). Deve-se ressaltar o fato de que o Khepera

pode se rotacionar sem se deslocar através da aplicação de tensões de mesmo valor absoluto, porém de sinais diferentes, em cada um dos motores.

O Khepera possui também um conjunto de oito sensores infra-vermelhos que são utilizados para verificar a distância a obstáculos em uma determinada direção. A cor dos obstáculos influencia diretamente o raio de operação destes sensores de proximidade. O alcance máximo de operação dos sensores é de 50 mm, e os valores retornados pelos sensores variam entre 0 e 1024, sendo que 0 indica que nenhum obstáculo foi encontrado e 1024 indica um obstáculo muito próximo ao robô (Ludgren, 2003). Na Fig. 5.3, pode-se ver como os sensores e os motores são posicionados ao longo do robô.

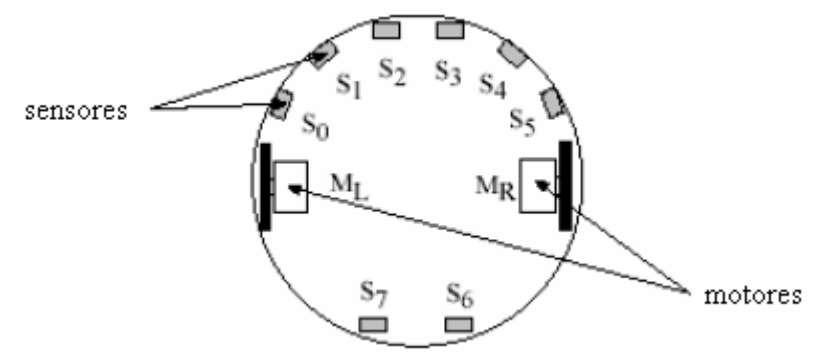

Figura 5.3: Localização dos sensores e motores do Khepera (Ludgren, 2003).

As principais vantagens da utilização de um robô móvel Khepera ao invés de um robô de grande porte são: a possibilidade de trabalhar com o robô nas proximidades de um computador, onde os dados possam estar sendo coletados, facilidade de transporte do robô para outros ambientes e a possibilidade de adição de novas funcionalidades ao robô através da adição de torres sobre o módulo base (Ludgren, 2003). Outra importante vantagem é que, em caso de colisões, o risco tanto para a integridade física do robô quanto àquela do objeto no qual ocorreu a colisão os danos serão menores, devido à menor massa do robô (Intza, 2000).

Devido ao fato de não existir um robô móvel Khepera com o qual pudessem ser realizados os experimentos, foi utilizado um simulador do Khepera desenvolvido pelo próprio autor desta dissertação. Neste simulador, o controlador *fuzzy* descrito na subseção 5.2.2 foi utilizado, sendo que as tomadas de decisão são realizadas a uma freqüência de 50 vezes por segundo, ou seja, a cada 20ms uma nova decisão é tomada pelo controlador do robô.

Ao invés do desenvolvimento de uma nova ferramenta de simulação, poderia-se utilizar uma de diversas ferramentas de simulação já existentes, como, por exemplo, a ExmoR (Lehmann, *et al.*, 1999). Porém, foi realizada a opção de desenvolvimento de uma nova ferramenta, pois nenhuma das ferramentas de simulação existente disponibilizava todas as possibilidades desejadas para o presente estudo de caso. Este simulador é apresentado em mais detalhes na seção 5.2.3 desta dissertação.

5.2.2. Controle *Fuzzy*

O funcionamento do controle *fuzzy* utilizado neste estudo de caso é similar ao desenvolvido por Thongchai (Thongchai, *et al.*, 2000).

O controlador *fuzzy* utilizado neste estudo de caso é do tipo Mamdani e é composto de quatro entradas, sendo estas os valores médios entre os sensores da direita, da esquerda, dos frontais e dos traseiros. O controlador fuzzy é responsável por, a partir destes valores, determinar as tensões a serem aplicadas em cada um dos motores que controlam as rodas.

O processo de fuzzificação das entradas deste controlador ocorre através do operador *min*, como previamente descrito na seção 4.3.1 e o processo de defuzificação ocorre através do método de Centro de Gravidade.

Os valores lingüísticos possíveis para as entradas são: obstáculo longe *(L)*, com proximidade intermediária *(I)* e obstáculo perto *(P)*. Os valores possíveis para as entradas variam de 0 a 1024, conforme limitações dos sensores do Khepera, já apresentadas na seção 5.2.1 desta dissertação. Os valores lingüísticos para as saídas são as tensões: negativa alta *(NA)*, negativa baixa *(NB)*, zero *(Z)*, positiva baixa *(PB)* e positiva alta *(PA)*. Os valores possíveis para os valores de saída estão dentro da faixa [-10; 10], conforme descrito na seção 5.2.1 desta dissertação. Neste controlador *fuzzy*, considera-se todas as funções de pertinência, tanto das entradas quanto das saídas, de formato triangular.

A base de regras proposta para este controlador compreende todas as possíveis combinações de entradas para estes três valores lingüísticos e é apresentada na Tabela 5.2. As entradas 1, 2, 3 e 4 são, respectivamente, os valores médios entre os sensores da esquerda, frente, direita e trás, e as saídas 1 e 2 são, respectivamente, as tensões a serem aplicadas nas rodas esquerda e direita.

Tabela 5.2: Base de regras do controlador *fuzzy* proposto.

Entrada 1	Entrada 2	Entrada 3	Entrada 4	Saída 1	Saída 2
L	L	L	L	PA	PA
L	L	L	I	PA	PA
L	L	L	P	PA	PA
L	L	I	L	Z	PB
L	L	I	I	Z	PB
L	L	I	P	Z	PB
L	L	P	L	Z	PA
L	L	P	I	Z	PA
L	L	P	P	Z	PA
L	I	L	L	Z	NB
L	I	L	I	Z	NB
L	I	L	P	Z	Z
L	I	I	L	NB	Z
L	I	I	I	NB	Z
L	I	I	P	NB	Z
L	I	P	L	NB	Z
L	I	P	I	NB	Z
L	I	P	P	NB	Z
L	P	L	L	NA	NB
L	P	L	I	NA	NB
L	P	L	P	NA	Z
L	P	I	L	NA	Z
L	P	I	I	NA	Z
L	P	I	P	NA	Z
L	P	P	L	NA	Z
L	P	P	I	NA	Z
L	P	P	P	NA	Z
I	L	L	L	PB	Z
I	L	L	I	PB	Z
I	L	L	P	PB	Z
I	L	I	L	PB	PB
I	L	I	I	PB	PB
I	L	I	P	PB	PB
I	L	P	L	Z	PB
I	L	P	I	Z	PB
I	L	P	P	Z	PB
I	I	L	L	Z	NB
I	I	L	I	Z	NB
I	I	L	P	Z	NB
I	I	I	L	NB	NA
I	I	I	I	Z	Z
I	I	I	P	Z	Z
I	I	P	L	Z	NB
I	I	P	I	Z	Z
I	I	P	P	Z	Z

Entrada 1	Entrada 2	Entrada 3	Entrada 4	Saída 1	Saída 2
I	P	L	L	Z	NB
I	P	L	I	Z	NB
I	P	L	P	Z	NB
I	P	I	L	NB	NB
I	P	I	I	Z	Z
I	P	I	P	Z	Z
I	P	P	L	NB	NB
I	P	P	I	Z	Z
I	P	P	P	Z	Z
P	L	L	L	PB	Z
P	L	L	I	PB	Z
P	L	L	P	PB	Z
P	L	I	L	PB	Z
P	L	I	I	PB	Z
P	L	I	P	PB	Z
P	L	P	L	PB	PB
P	L	P	I	PB	PB
P	L	P	P	PB	PB
P	I	L	L	Z	NB
P	I	L	I	Z	NB
P	I	L	P	Z	NB
P	I	I	L	NB	NB
P	I	I	I	Z	Z
P	I	I	P	Z	Z
P	I	P	L	NB	NB
P	I	P	I	Z	Z
P	I	P	P	Z	Z
P	P	L	L	Z	NA
P	P	L	I	Z	NA
P	P	L	P	Z	NA
P	P	I	L	NB	NB
P	P	I	I	Z	Z
P	P	I	P	Z	Z
P	P	P	L	NB	NB
P	P	P	I	Z	Z
P	P	P	P	Z	Z

Uma vez que tanto as entradas e saídas quanto a base de regras e os processos de fuzzificação e defuzificação sejam conhecidos, resta apenas compreender como será realizado o processo de otimização dos controladores. Isto será descrito na seção 5.2.4.

5.2.3. Ambiente

Uma vez determinado o controlador *fuzzy* é necessário testá-lo.Com tal intuito foi utilizado o simulador do Khepera para testar quão bem aquele controlador conseguia atender os requisitos do estudo de caso, ou seja, explorar um ambiente. Nesta seção está descrito o ambiente no qual o Khepera é colocado de forma a testar o seu desempenho na exploração.

O ambiente é uma região quadrada de 100x100 cm, circundado por paredes de 1 cm. O ambiente é composto por obstáculos retangulares e circulares, posicionados no meio do ambiente de forma a dificultar a movimentação do robô. Na Fig. 5.4, pode ser visto a interface do Simulador Khepera com o ambiente utilizado para realizar a simulação.

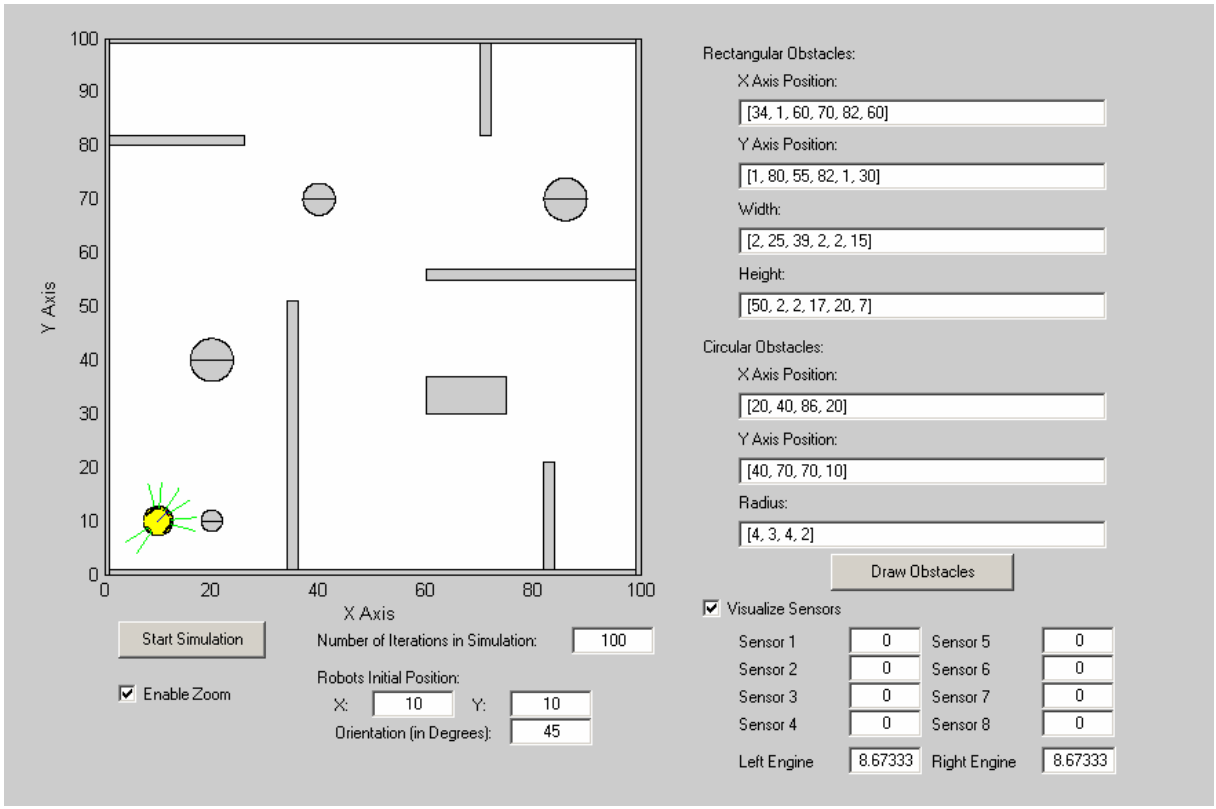

Figura 5.4: Interface do simulador Khepera com o ambiente utilizado para a simulação.

A Fig. 5.4 representa as condições iniciais de simulação para cada experimento, ou seja, o posicionamento do robô ocorre sempre na posição $(x, y) = (10, 10)$ e com orientação de 45 graus ($\pi/4$ rad.). Os obstáculos são apresentados em cinza e o Khepera é apresentado em amarelo, sendo que os seus sensores são apresentados em vermelho e suas rodas em preto. Além disso, são apresentadas retas que demonstram o raio de alcance de cada sensor, estas

retas podem ser representadas em verde (quando o sensor não capta nenhum obstáculo) ou em vermelho (quando o sensor capta um obstáculo). Na Fig. 5.5, pode-se ver um zoom da representação do robô Khepera. O lado da frente do Khepera é o lado com a maior quantidade de sensores.

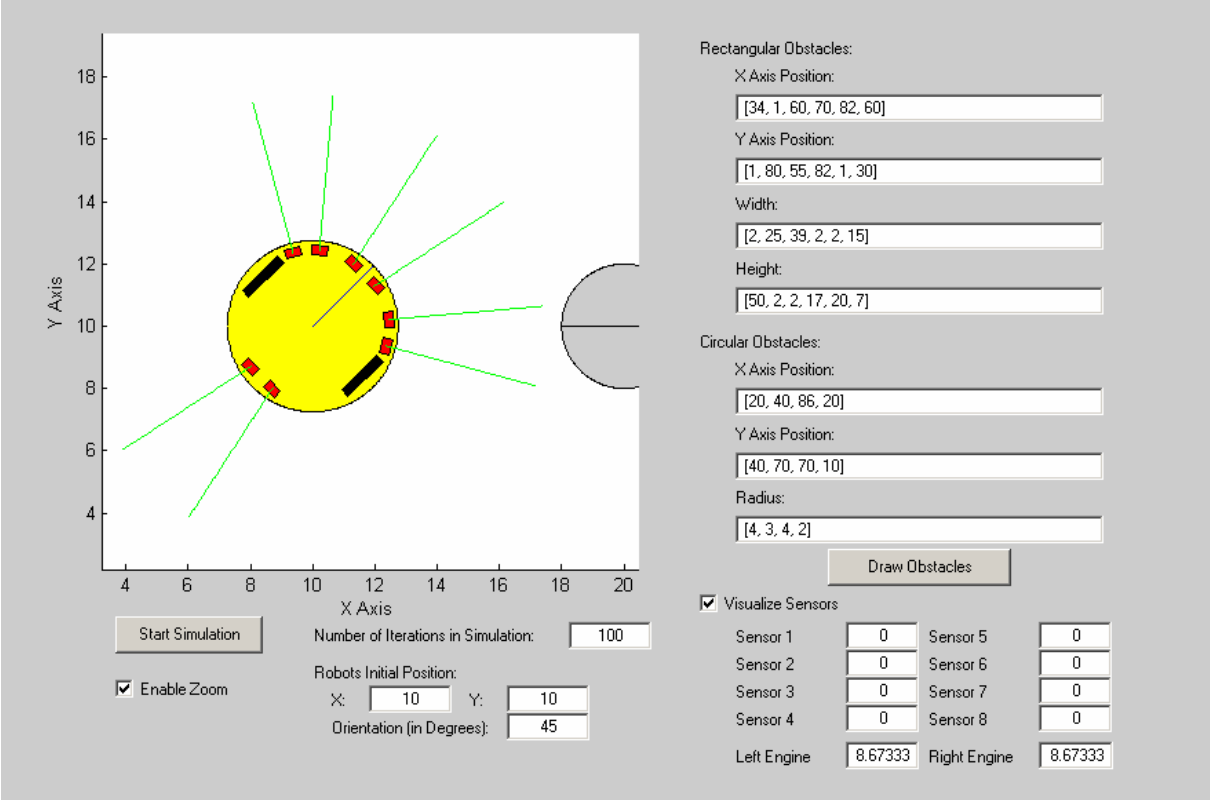

Figura 5.5: Zoom do robô Khepera utilizado para as simulações.

Pode ser visto ainda na interface apresentada na Fig. 5.5, que o simulador apresenta para o usuário informações referentes aos valores de leitura de cada sensor, e o valor de atuação sendo aplicado em cada um dos motores.

5.2.4. Inteligência Coletiva

Nesta subseção, é apresentado como os algoritmos de inteligência coletiva são utilizados para otimizar o controlador *fuzzy* apresentado na seção 5.2.2. No presente estudo de caso, optou-se por modificar, através dos algoritmos de inteligência coletiva, as funções de pertinência do controlador *fuzzy*. Outra opção não inclusa neste estudo de caso poderia ser a otimização da base de regras do controlador.

Como foi previamente dito, as funções de pertinência utilizadas tanto para as entradas quanto para as saídas, são triangulares. As funções de pertinência triangulares possuem dois parâmetros a serem otimizados: primeiro a posição do centro do triângulo, ou seja, a posição onde a pertinência é igual a 1, e segundo, a largura da base do triângulo, ou seja, a velocidade de queda da pertinência daquela função.

Para cada entrada existem três variáveis lingüísticas e para cada saída existem cinco variáveis lingüísticas é necessário um total de 44 (quarenta e quatro) parâmetros a serem otimizados pelo algoritmo de inteligência coletiva (4 entradas X 3 variáveis lingüísticas X 2 parâmetros + 2 saídas X 5 variáveis lingüísticas X 2 parâmetros).

Cada um destes parâmetros determinados pelos algoritmos de inteligência coletiva é normalizado entre 0 e 1, de forma a facilitar a manipulação dos valores, pois os valores de entrada dos sensores variam entre 0 e 1024, enquanto que os valores de saída para os motores variam entre -10 e 10V. Também para futuras alterações nas restrições físicas do robô móvel que sejam implementadas, para o caso de se optar por simular as condições para um outro robô.

Com isso, é possível determinar como os indivíduos dos algoritmos de inteligência coletiva serão configurados. Porém, era ainda necessário desenvolver uma ferramenta para a avaliação de cada um dos indivíduos da população, ou seja, a função de *fitness*, de forma a conseguir a melhor exploração possível do terreno onde se localiza o robô móvel Khepera.

No caso, a função de *fitness* foi baseada em três componentes: a velocidade absoluta de movimentação do robô (V), a movimentação em linha reta (Δv) e desvio de obstáculos (i) (Nolfi, 2000).

No caso a função de *fitness* é determinada segundo a equação (5.4).

$$F = V * \left(1 - \sqrt{\Delta v}\right) * \left(1 - i\right) \tag{5.4}$$

onde V é a soma das velocidades de rotação das duas rodas do robô, ou seja, quanto mais rápido o robô se locomover para a frente, maior será o *fitness*, Δv é o valor absoluto da diferença entre as velocidades de rotação das rodas do robô, ou seja, quanto menor a diferença de velocidade entre as rodas maior o *fitness*, e i é o valor normalizado do sensor com maior com maior índice de atividade no momento. Esta seleção do *fitness* é motivada pelo fato de serem considerados os três principais fatores de exploração de territórios e desvio de

obstáculos, são eles: movimentação em linha reta, com a maior velocidade possível, e sem encontrar obstáculos.

Cada um dos componentes utilizados para calcular a função de *fitness* permanece dentro do limite [0, 1], exceto o componente V que pode assumir valores negativos caso o robô se movimente para trás. Esta normalização ocorre de forma a ponderar a importância entre os três componentes, fazendo com que os três componentes possuam a mesma importância no valor final da função de *fitness*.

De forma a realizar a avaliação do *fitness* de cada indivíduo da população do algoritmo de inteligência coletiva, uma simulação daquele indivíduo é realizada através do simulador do Khepera. No caso, é realizada uma simulação com 50 iterações, este valor representa o suficiente para o robô se locomover até 60 cm, praticamente metade do ambiente de simulação, caso se locomovesse entre os dois pontos mais distantes (141 cm) à velocidade máxima. A utilização de um número maior de iterações seria preferível como forma de testar melhor o controlador para aquele determinado indivíduo. Porém, esta opção não foi adotada devido ao elevado esforço computacional necessário, e em função da aproximação não ser ruim.

Capítulo 6

Resultados da Simulação

No presente capítulo são apresentados os resultados obtidos para os estudos de caso descritos no Capítulo 5. Serão empregadas todas as técnicas de inteligência coletiva descritas ao longo desta dissertação. Para a obtenção dos presentes resultados, as técnicas e o estudo de caso foram implementados no ambiente computacional Matlab®, versão 5.2, da MathWorks, utilizando um computador com processador Intel® Pentium® IV 2.66 GHz, com 1024MB de memória RAM, operando sob o Sistema Operacional Microsoft® Windows® XP Home Edition.

6.1 Planejamento de Trajetórias

O problema de planejamento de trajetórias foi previamente descrito na seção 5.1. De forma a facilitar a análise comparativa dos resultados para o estudo de caso, os resultados são separados em subseções.

6.1.1. Configuração dos Algoritmos

Como a pesquisa sendo apresentada nesta dissertação é uma pesquisa em largura e não em profundidade para os algoritmos, não foi realizada uma análise paramétrica para os algoritmos, portanto, os parâmetros selecionados foram baseados na literatura.

a. Algoritmos Genéticos

O algoritmo genético utilizado para o presente estudo de caso utiliza representação binária, sendo que 16 bits são utilizados para representar cada variável. Isto garante uma

precisão de $1{,}5259 \cdot 10^{-3}$. O tamanho da população é de 100 indivíduos, e o número máximo de gerações é 200. O operador de cruzamento selecionado foi o cruzamento com um ponto de corte, com probabilidade de ocorrência de 80% e a probabilidade de mutação de 3%. Ainda vale ressaltar que o método de seleção selecionado é a seleção por roleta e que o algoritmo utiliza estrutura elitista. Estes parâmetros foram selecionados a partir da literatura (De Jong, 1975).

b. Nuvem de Partículas

O algoritmo de otimização por nuvem de partículas foi configurado com os seguintes parâmetros para o estudo de caso a ser apresentado: tamanho da população de 50, o número máximo de iterações (critério de parada) foi 100, os componentes cognitivo (c_1) e social (c_2) foram escolhidos como 2.0. A velocidade máxima de deslocamento das partículas foi limitada a 10% do tamanho do espaço de buscas, ou seja, o valor máximo permitido é 10, o peso inercial foi adotado como dinâmico e atualizado segundo a equação (3.3), onde o valor máximo permitido é 0,9 e o valor mínimo é 0,4. Esta seleção de parâmetros foi baseada na literatura, conforme descrito na seção 3.1.3 desta dissertação.

c. Colônia de Bactérias

O algoritmo de colônia de bactérias utilizado para gerar os resultados para o estudo de caso foi configurado com um tamanho de população 50, com 2 passos de eliminação e dispersão, 4 passos de reprodução, 15 passos *chemotáticos*. O número máximo de passos que podem ser dados seguidos em uma mesma direção é de 10, sendo que os valores do tamanho de passo variam entre 0 e 2,5.

O valor da quantidade de atraentes expelidos pelas bactérias ($d_{atração}$) é 0,1; a largura onde estes atraentes têm efeito ($w_{atração}$) é de 0,2; o valor da quantidade de repelente emitido pelas bactérias ($h_{repulsão}$) é 0,1 e a faixa onde estes repelentes têm efeito ($w_{repulsão}$) é de 10. Estes valores são escolhidos de forma a ilustrar o comportamento de bactérias reais, segundo (Passino, 2002).

d. Colônia de Formigas

Antes de ser possível aplicar o algoritmo de colônia de formigas ao estudo de caso descrito na seção 5.1, deve-se selecionar diversos parâmetros do algoritmo. Os valores selecionados são número de formigas presentes na colônia é 30, quantidade de regiões de

busca que podem ser memorizadas por cada formiga é 3, limite de falhas que podem ocorrer em uma região de busca antes de a mesma ser esquecida pela formiga é 3, número máximo de iterações onde as formigas buscam por alimentos dentro das regiões de busca é 10, número máximo de mudanças de ninho permitidas é 50. O raio ao redor do ninho dentro do qual as formigas podem gerar as suas regiões de busca é 30, e o fator de atenuação deste raio é 0,95.

e. Sistemas Imunológicos Artificiais

Os seguintes parâmetros foram adotados para os Sistemas Imunológicos Artificiais, tamanho da população inicial é 30, número máximo de iterações permitidas é 30, limite de supressão é 35, percentual de acréscimo de novas células é 40%, número de clones gerados a partir de cada célula 10 e $\beta = 20$. Foram utilizados estes parâmetros com base nas informações apresentadas em (Silva e Timmis, 2002).

6.1.2. Tabelas de Resultados

Na Tabela 6.1, é apresentada uma sumarização entre os resultados obtidos de todas as técnicas previamente apresentadas, sendo que foram realizados 50 experimentos para cada técnica.

Tabela 6.1: Comparação entre os resultados das diversas técnicas.

Método	Média	Mínimo	Máximo	Desvio Padrão	Número de Parâmetros	Tempo Computacional
AG	556,3625	150,6544	2860,6750	340,1439	5	2’52”
PSO	527,5382	164,3050	2679,4416	280,3746	6	0’53”
Colônia de Bactérias	773,0888	150,7608	2829,4322	315,9810	10	2’53”
Colônia de Formigas	621,5343	148,1873	2275,3242	278,3047	7	2’19”
SIAs	739,8635	152,8009	2705,3042	281,5169	6	1’27”

Como pode ser percebido através dos resultados apresentados na Tabela 6.1, o AG, que é a técnica mais difundida no meio acadêmico, possui dificuldades de convergência para a solução em problemas onde o número de parâmetros a serem otimizados é grande, no caso são necessários aperfeiçoar oito parâmetros.

O PSO é a técnica que possui melhores características de convergência para uma solução, ele possui grande capacidade de otimização principalmente quando são considerados pequenos espaços de busca, quando o espaço de buscas é muito amplo o algoritmo possui certas dificuldades na otimização, o que ocorre devido ao fato de o mesmo convergir prematuramente. Isto pode fazer com que o mesmo fique "preso" em mínimos locais. Este fato pode ser ressaltado pelo fato de que em três dos experimentos realizados não foi possível obter soluções factíveis.

Além disso, o PSO é o algoritmo que necessita de menor esforço computacional para a obtenção da resposta e, quando aplicado para otimizar soluções iniciais respectivamente boas, pode ser de grande utilidade.

Além destes, as técnicas de colônia de bactérias e colônia de formigas também conseguem atingir boas soluções, sendo que o algoritmo de colônia de bactérias é capaz de encontrar a vizinhança do ótimo, porém não é capaz de melhorar localmente esta solução de forma a atingir o ótimo nesta vizinhança. Outro ponto a ser ressaltado é a exigência computacional requerida por este algoritmo. Dentre as técnicas analisadas, esta é a que necessita de maior esforço computacional, muito próximo ao AG. O algoritmo de colônia de formigas é o algoritmo que atinge a melhor solução para o experimento.

Percebe-se que o algoritmo de SIA possui uma grande capacidade de otimização e se aproveita da inserção de novas células para aumentar a diversidade da população, que pode vir a melhorar a solução obtida até o momento.

Levando-se em conta o número de parâmetros a serem configurados nos algoritmos testados, pode-se verificar que os algoritmos genéticos, PSO e SIA são os mais facilmente ajustados. Analisando-se este fato em conjunto com as soluções obtidas e com o custo computacional dos algoritmos, conclui-se que, neste caso, os dois melhores algoritmos a serem utilizados são o de colônia de formigas que é o mais adequado para convergir para a solução, e o PSO que é o que possui melhor custo beneficio, entre custo computacional e solução obtida.

6.1.3. Gráficos

Nesta subseção são apresentados os gráficos obtidos a partir das simulações realizadas.

a. Melhores Trajetos

Na Fig. 6.1 são apresentados os melhores trajetos encontrados para cada um dos algoritmos apresentados.

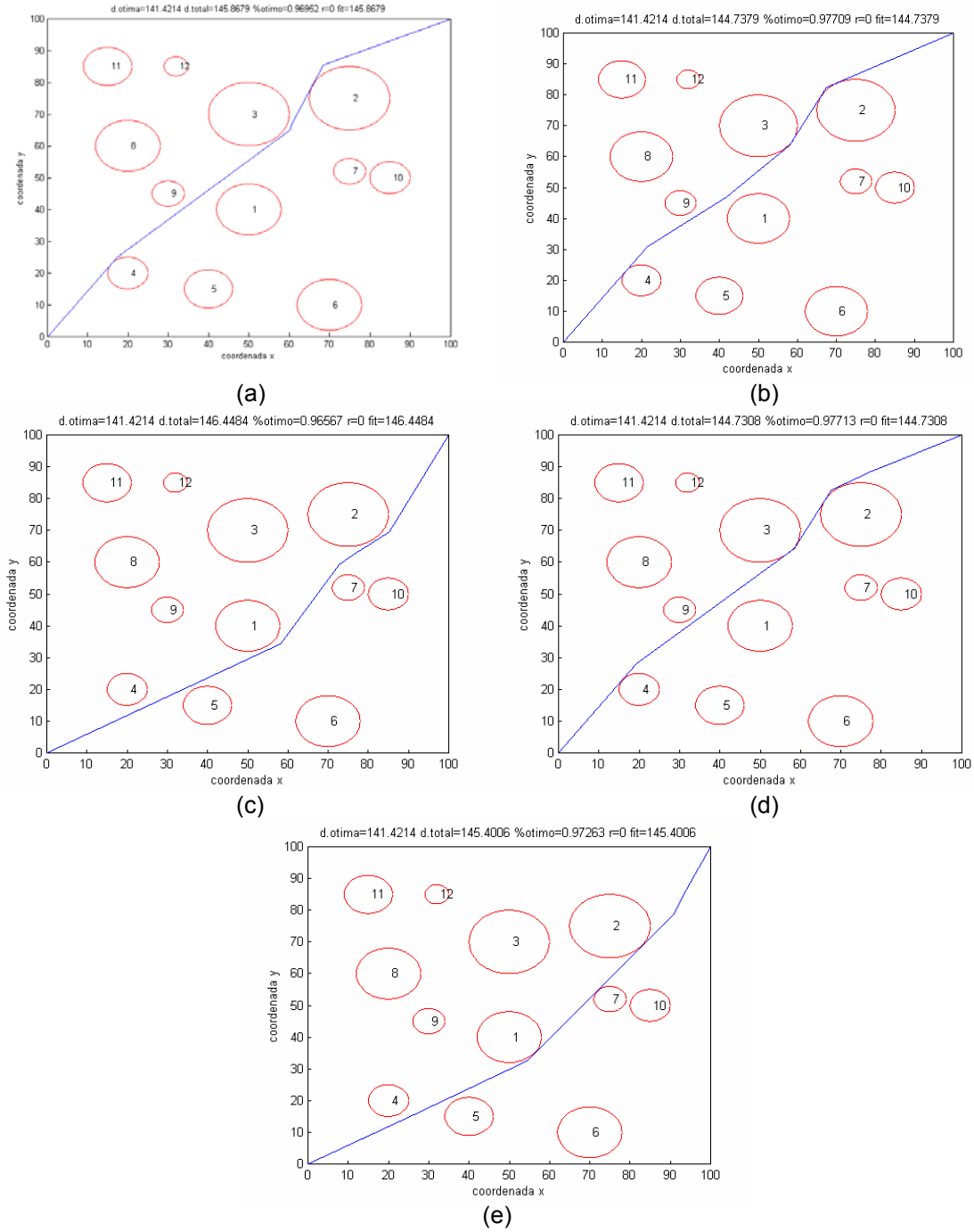

Figura 6.1: Melhores trajetos obtidos pelos algoritmos.

O experimento que atingiu a melhor resposta para o AG, atingiu a solução cujas coordenadas são: (x_1, y_1) = (0,062562; 3,92157), (x_2, y_2) = (1,149; 8,02777), (x_3, y_3) = (7,73632; 25,5467) e (x_4, y_4) = (43,418; 79,3103). Resultando na trajetória apresentada na Fig 6.1 (a).

O experimento que atingiu a melhor solução para o algoritmo PSO, a atingiu com as seguintes coordenadas: (x_1, y_1) = (21,4253; 30,8051), (x_2, y_2) = (41,7935; 46,8964), (x_3, y_3) = (58,0152; 63,7053) e (x_4, y_4) = (67,5319; 82,3712). Estes pontos representam a trajetória apresentada na Fig. 6.1 (b).

O experimento que convergiu para a melhor solução para o algoritmo de colônia de bactérias, a obteve com as seguintes coordenadas: (x_1, y_1) = (58,1957; 34,3295), (x_2, y_2) = (72,6905; 59,2914), (x_3, y_3) = (76,4051; 62,5056) e (x_4, y_4) = (85,1183; 69,5212). Estes pontos representam a trajetória apresentada na Fig. 6.1 (c).

As coordenadas obtidas para o melhor experimento de colônia de formigas foram as seguintes: (x_1, y_1) = (19,3429; 28,2088), (x_2, y_2) = (58,4135; 64,1124), (x_3, y_3) = (67,5946; 82,6675) e (x_4, y_4) = (77,5876; 88,5364). Estes pontos representam a trajetória mostrada na Fig. 6.1 (d).

A melhor solução encontrada para o algoritmo de sistemas imunológicos artificiais é representada pelas coordenadas (x_1, y_1) = (26,3013; 15,6858); (x_2, y_2) = (54,547; 32.6946); (x_3, y_3) = (91,06; 78.9673) e (x_4, y_4) = (93,9988; 86.6236), correspondendo à trajetória mostrada na Fig. 6.1 (e).

b. Evolução *Fitness*

Na Fig. 6.2 são apresentados os gráficos referentes à evolução dos *fitness* mínimo e médio para os experimentos que atingiram as melhores soluções para cada um dos algoritmos apresentados. Sendo que o gráfico (a) representa a evolução para o AG, (b) para o PSO, (c) para o algoritmo de colônia de bactérias, (d) para colônia de formigas e (e) para o algoritmo SIA.

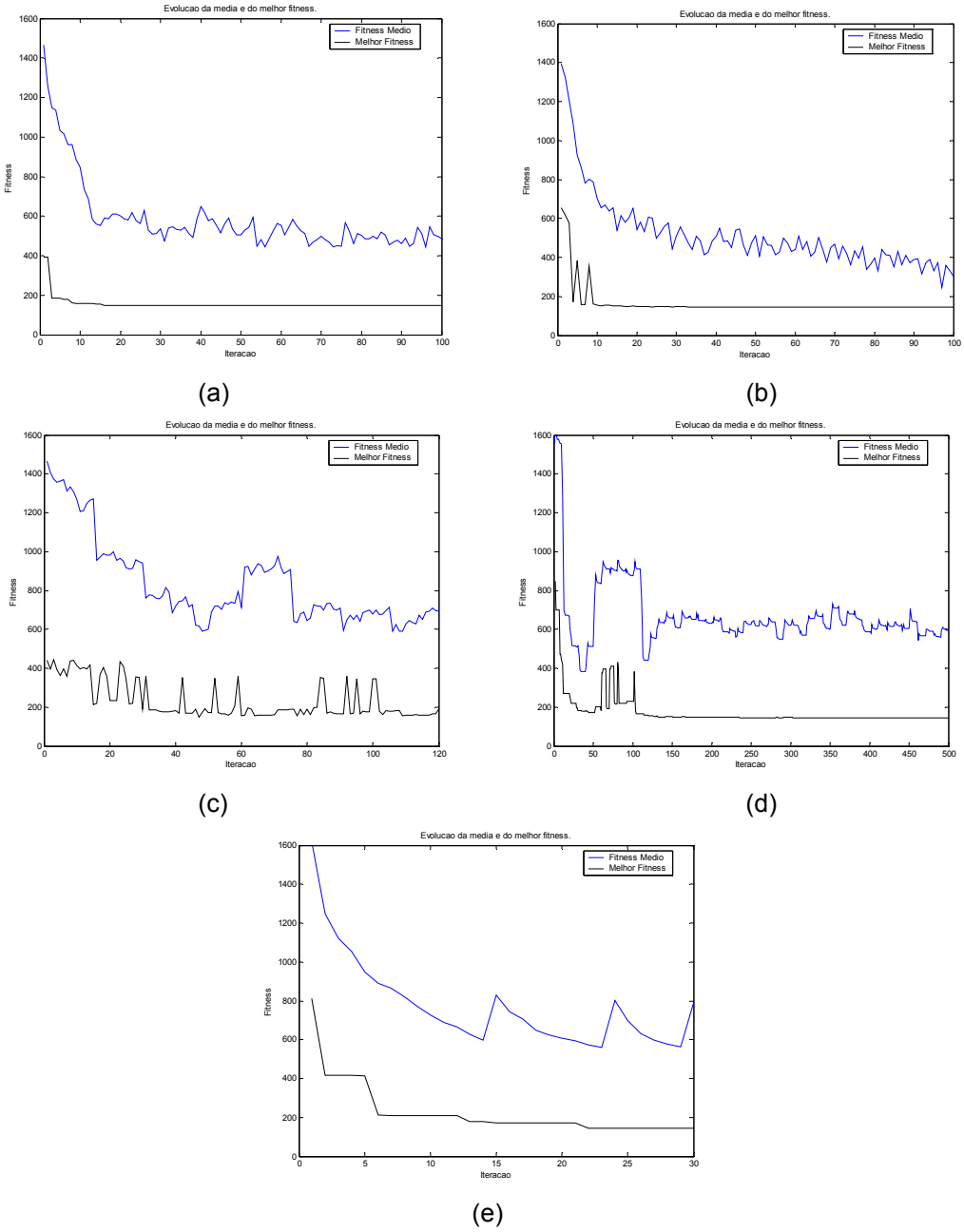

Figura 6.2: Evolução do *fitness* mínimo e médio para o melhor experimento de cada algoritmo.

Como pode ser analisada através da Fig. 6.2 (a), a variância existente entre os indivíduos presentes na população é muito grande. Isto faz com que o valor da média dos *fitness* varie muito ao longo das iterações, sinal de que a diversidade da população está sendo mantida. Em relação à evolução do *fitness* mínimo, percebe-se que a melhor solução para o

experimento já é atingida próximo à iteração 15, e a sua manutenção dentro da população ocorre devido à utilização da estrutura elitista.

Na Fig. 6.2 (b), pode-se verificar que o algoritmo PSO converge para uma solução. Na Fig. 6.2 (c), pode-se ver que o algoritmo de colônia de bactérias não possui uma estrutura elitista, mas que a melhor solução em alguns momentos é “esquecida”, mas em algumas iterações já é recuperada.

Na Fig. 6.2 (d), percebe-se que o algoritmo de colônia de formigas obtém a melhor solução logo nas primeiras iterações e a partir de então não a melhora, mas continua com um valor médio bastante grande, fato importante para manter a diversidade na população.

Na Fig. 6.2 (e), é apresentado o gráfico de evolução dos *fitness* para o algoritmo SIA, que, em determinados momentos, o *fitness* médio possui alguns picos, estes ocorrem nos momentos que o algoritmo começa a convergir para uma solução. Quando a diferença entre a média dos *fitness* da população em duas iterações seguidas é menor do que dez, ocorre a inserção de novas células na população em posições aleatórias. Com isso, aumenta o *fitness* médio da população novamente, este processo ocorre de forma contínua.

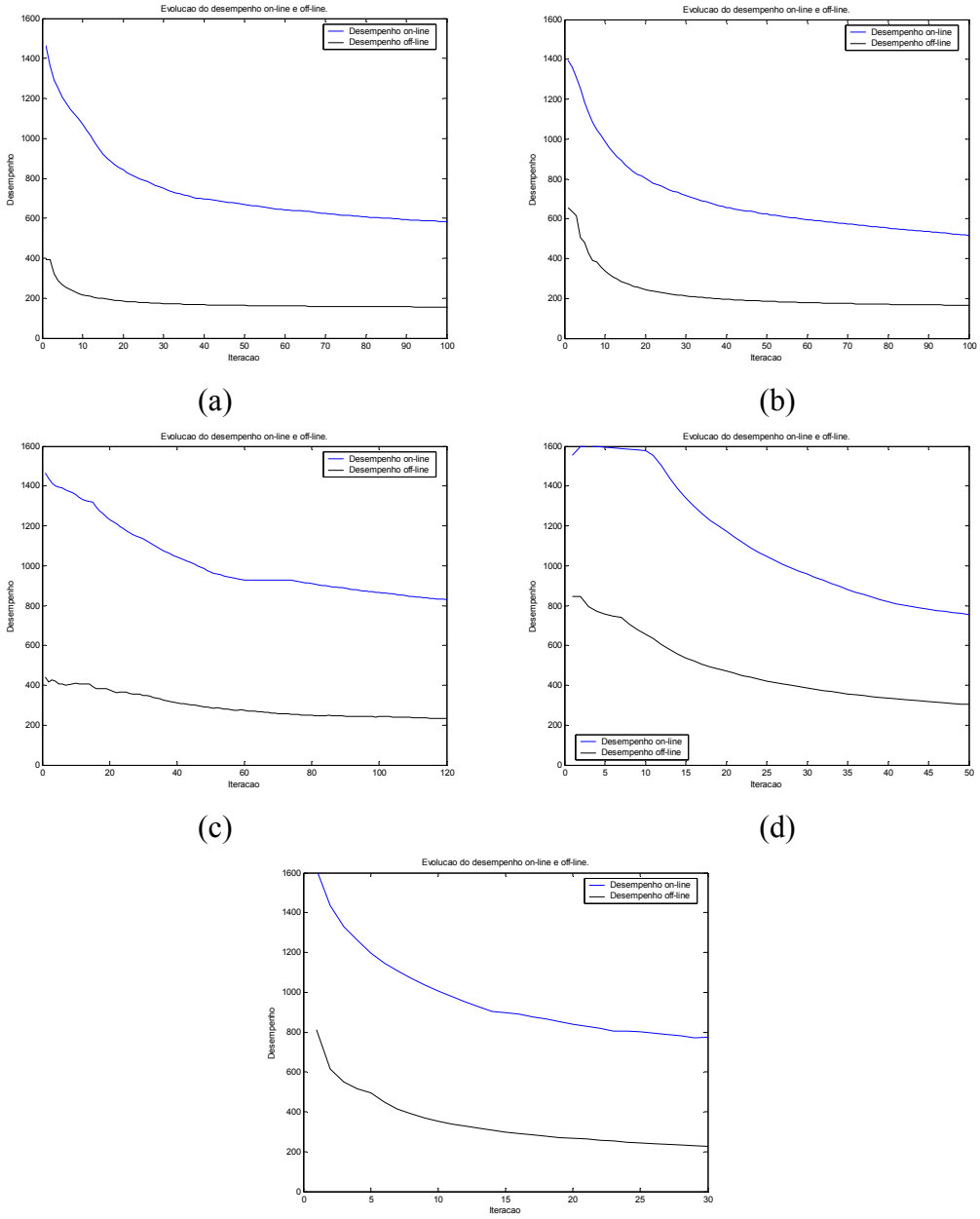

Figura 6.3: Desempenho on-line e off-line dos algoritmos.

Na Fig. 6.3 são apresentados os gráficos referentes ao desempenho *on-line* (média dos *fitness* de todos os indivíduos gerados até o momento) e *off-line* (média dos melhores *fitness* de cada geração gerados até uma determinada geração), conforme mencionado na seção 5.1 desta dissertação, para os algoritmos apresentados, sendo que o gráfico (a) refere-se ao AG, (b) ao PSO, (c) ao colônia de bactérias, (d) ao colônia de formigas e (e) ao algoritmo SIA.

O gráfico dos desempenhos *on-line* e *off-line* é de grande valia para a análise da diversidade genética existente dentro da população. Quanto maior for a diferença existente entre os dois desempenhos, maior é a diversidade genética. Caso os dois desempenhos sejam iguais, significa que a população evolui para uma única solução (Bobel, 2003). Pelos gráficos de desempenho apresentados, é possível ressaltar a afirmação de uma grande diversidade genética na população para todos os algoritmos.

Pode-se perceber, através dos gráficos apresentados nas Figs. 6.3 (b) e (d) que, principalmente, os algoritmos PSO e colônia de formigas são os que mais se aproximam de convergir para uma solução, pois nestes gráficos percebe-se que as linhas referentes aos desempenhos *on-line* e *off-line* estão se aproximando com o decorrer das iterações.

6.2 Otimização de Controladores *Fuzzy*

Como já previamente apresentado na seção 5.2 desta dissertação, este estudo de caso é referente à simulação para a otimização do comportamento de um robô móvel regido por um controlador *fuzzy* na exploração de um ambiente previamente desconhecido. Esta otimização do comportamento do robô móvel é realizada através dos algoritmos de inteligência coletiva apresentados nos capítulos 2 e 3, manipulando as funções de pertinência do controlador *fuzzy*. A seguir, são apresentados os resultados obtidos com cada um dos algoritmos para as simulações.

Para os algoritmos utilizados neste estudo de caso, foi adotada uma solução inicial não aleatória, que representa um controlador *fuzzy* básico, não otimizado. As funções de pertinência deste controlador são representadas pelas seguintes coordenadas: [0 0.45 1 0 0.45 1 0 0.45 1 0 0.45 1 0.2 0.7 0.8 0.2 0.7 0.8 0.2 0.7 0.8 0.2 0.7 0.8 0 0.3 0.5 0.7 1 0 0.3 0.5 0.7 1 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9] (Figs. 6.4 e 6.5).

A codificação utilizada para representar os indivíduos é a seguinte, primeiramente aparecem os três pontos referentes aos três centros das funções de pertinência (obstáculo longe, perto ou com proximidade intermediária) para a entrada 1 (valor médio entre os sensores da esquerda do robô Khepera), seguidos pelos três pontos referentes aos centros das funções de pertinências para as entradas 2, 3 e 4. A seguir, são colocadas as larguras das bases para as três funções de pertinência para as entradas 1, 2, 3 e 4, respectivamente. Esta ordem dos parâmetros para a codificação foi selecionada empiricamente.

Uma vez todas as entradas codificadas, os dados referentes às funções de pertinência da saída passam a ser codificados, primeiramente são inseridos no vetor os valores referentes aos centros das cinco funções de pertinência (Negativa alta, negativa baixa, zero, positiva baixa e positiva alta) para a saída 1 (roda da esquerda), seguido pelos valores para os centros das funções de pertinência para a saída 2 (roda da direita), seguido pelas larguras das bases das funções de pertinência para as saídas 1 e 2, respectivamente.

No caso do indivíduo acima apresentado, o controlador codificado refere-se às funções de pertinência apresentadas nas Figs. 6.4 e 6.5.

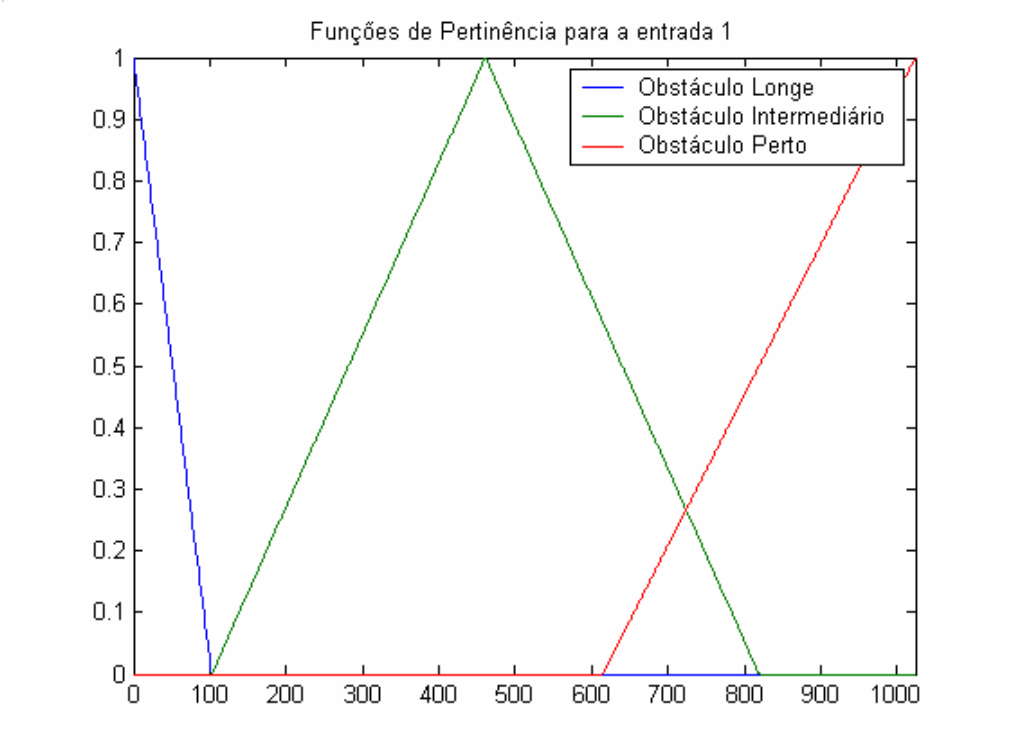

Figura 6.4: Funções de pertinência para as entradas 1, 2, 3 e 4.

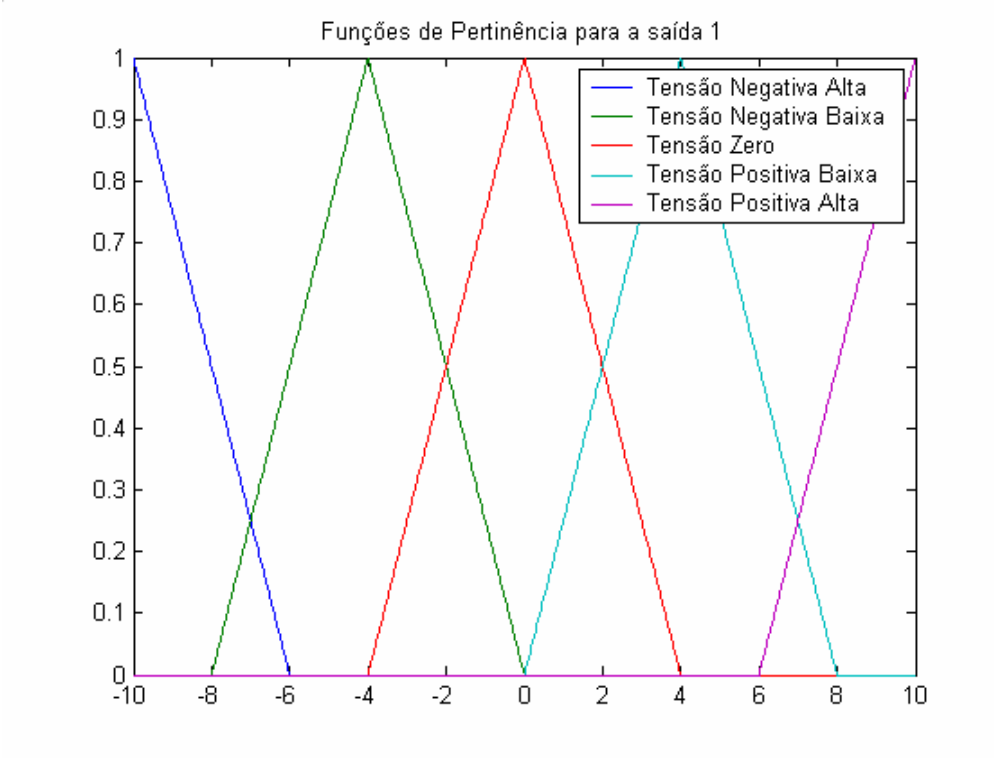

Figura 6.5: Funções de pertinência para as saídas 1 e 2.

6.2.1. Configuração dos Algoritmos

Nesta seção são apresentados os parâmetros selecionados para os algoritmos, de forma a obtermos os resultados apresentados nesta seção. A seleção dos parâmetros foi baseada na seleção feita no primeiro estudo de caso, apresentado na seção 6.1.1.

O algoritmo genético foi configurado conforme o seguinte, algoritmo genético binário, com 10 bits para representar cada uma das variáveis, tamanho máximo da população de 50 indivíduos, com probabilidade de cruzamento de 80% e probabilidade de mutação de 3%. O número máximo de gerações permitidas para o algoritmo é de 80, o algoritmo ainda foi configurado com estrutura elitista de forma que a melhor solução não é perdida ao longo das iterações, e seleção via roleta.

O algoritmo de Nuvem de Partículas (PSO), utilizado para este estudo de caso, foi configurado com os seguintes parâmetros de projeto: tamanho da população de 30 indivíduos, 50 iterações como o critério de parada do algoritmo, os componentes cognitivo (c_1) e social (c_2) foram ambos escolhidos como 2,0. A velocidade máxima de deslocamento das partículas foi limitada a 10% do tamanho do espaço de busca, ou seja, o valor máximo permitido é 0,1, o

peso inercial foi adotado como dinâmico e atualizado segundo a equação (3.3), onde o valor máximo permitido é 0,9 e o valor mínimo é 0,4.

O algoritmo de colônia de bactérias foi configurado com: tamanho da população de 30 bactérias, com 2 passos de eliminação e dispersão, 2 passos de reprodução e 10 passos *chemotáticos*, sendo que o número máximo de passos que uma determinada bactéria pode dar em uma mesma direção foi limitado a 5, sendo o tamanho dos passos limitados entre 0 e 2,5.

Os outros valores relativos à atração e repulsão entre as bactérias foram deixados semelhantes aos apresentados no primeiro estudo de caso, de forma a ilustrar da forma mais aproximada possível o comportamento de bactérias reais.

Como forma de testar o algoritmo de colônia de formigas para o presente estudo de caso, o mesmo foi configurado com os seguintes parâmetros de projeto: número de formigas igual a 30, número máximo de regiões de busca memorizadas por cada formiga de 2, 1 sendo o número máximo de fracassos que podem ocorrer em uma determinada região de busca antes de a mesma ser esquecida, o número máximo de iterações onde as formigas podem explorar as regiões de busca é 3.

O raio ao redor do ninho onde as regiões de busca são criadas é de 0,35, considerando o fator de atenuação deste raio de 95%, sendo que o número máximo de mudanças de ninho permitido é de 30 iterações.

A configuração do algoritmo de Sistemas Imunológicos Artificiais para o presente estudo de caso foi feita conforme os seguintes parâmetros: tamanho inicial da população de 20 indivíduos, número máximo de iterações permitidas de 25, limite de supressão 0.1, quantidade de clones gerados a partir de cada célula 3.

6.2.2. Tabela de Resultados

Utilizando os algoritmos descritos nos Capítulos 2 e 3, configurados conforme a seção 6.2.1, 30 experimentos são realizados para cada técnica como forma de termos uma análise comparativa entre o desempenho dos mesmos no presente estudo de caso.

Como forma de realizar esta análise comparativa as técnicas apresentadas ao longo desta dissertação, primeiramente é feita uma análise dos resultados obtidos para cada uma destas técnicas. Uma síntese desta análise é apresentada na Tabela 6.2. Na Tabela 6.2, é também apresentada uma análise comparativa do custo computacional entre as técnicas.

Tabela 6.2: Análise comparativa entre os algoritmos para o estudo de caso 2.

Algoritmo	Média	Mínima	Máxima	Desvio Padrão	Tempo Médio
AG	0,2077	0	0,6561	0,1757	5h28min
PSO	0,1555	0	0,6421	0,2126	4h52min
Bactérias	-0,1536	-0,4341	0,4060	0,0877	9h03min
Formigas	0,2250	0	0,7073	0,2161	8h32min
SIA	0,0304	0	0,5404	0,0941	12h47min

Analisando os resultados obtidos, pode-se verificar que os algoritmos genético, PSO e colônia de formigas atingem boas soluções para o presente estudo de caso, porém os mesmos não possuem características de convergência.

Os outros algoritmos acabam não conseguindo atingir uma solução satisfatória. Porém, o desvio padrão para estas técnicas é menor. Isso ocorre devido aos mesmos não conseguirem uma maior diversidade nas suas soluções. Com isso, pode-se concluir que a não convergência para uma solução pode vir a ser um fator importante para a obtenção de boas soluções nos algoritmos de inteligência computacional, sendo que para que estas boas soluções não acabem sendo descartadas os algoritmos devem possuir algum mecanismo de "elitização" dos indivíduos da população.

O fato dos algoritmos não conseguirem boas soluções ou então de não conseguirem convergir para uma determinada solução ocorre devido à elevada complexidade do presente estudo de caso, fato comprovado pela quantidade de parâmetros a serem otimizados e também devido ao custo computacional dos algoritmos, apresentado na última coluna da Tabela 6.2.

6.2.3. Gráficos

Nesta subseção são apresentados alguns gráficos referentes aos trajetos dos robôs utilizando os controladores projetados pelos algoritmos e também alguns gráficos relacionados à convergência dos mesmos.

a. Sinal de Controle e Trajetos

Nesta seção são apresentados os sinais de controle e os trajetos que os robôs seguem durante uma simulação dos mesmos, para todos os algoritmos testados, estes são apresentados nas Figs. 6.6 a 6.10, para os algoritmos genético, PSO, colônia de bactérias, colônia de

formigas e SIA, respectivamente. Nos gráficos das trajetórias percorridas, as linhas representam o trajeto percorrido, e as bolas em cima das trajetórias estão posicionadas como forma de facilitar a ligação com o gráfico dos sinais de controle, as bolas representam o ponto onde o robô estava localizado nas iterações múltiplas de 10.

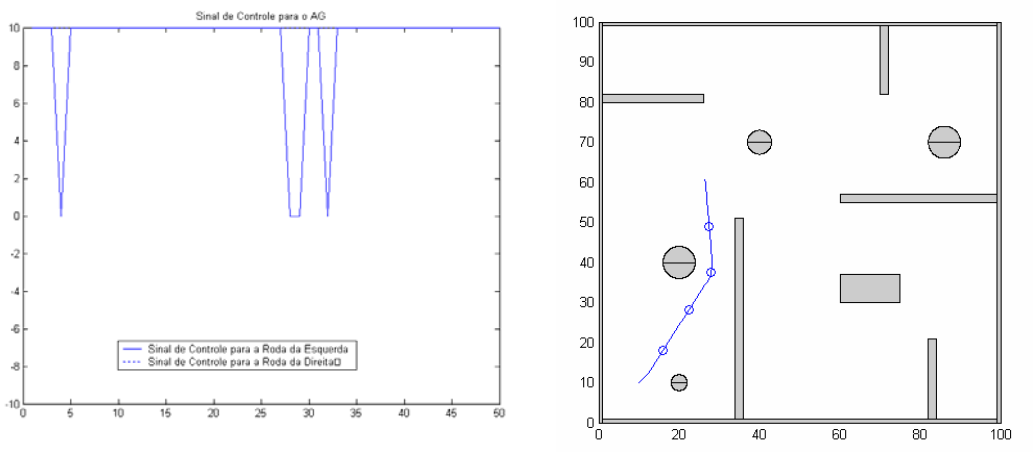

Figura 6.6: Sinal de controle e trajeto percorrido pelo AG.

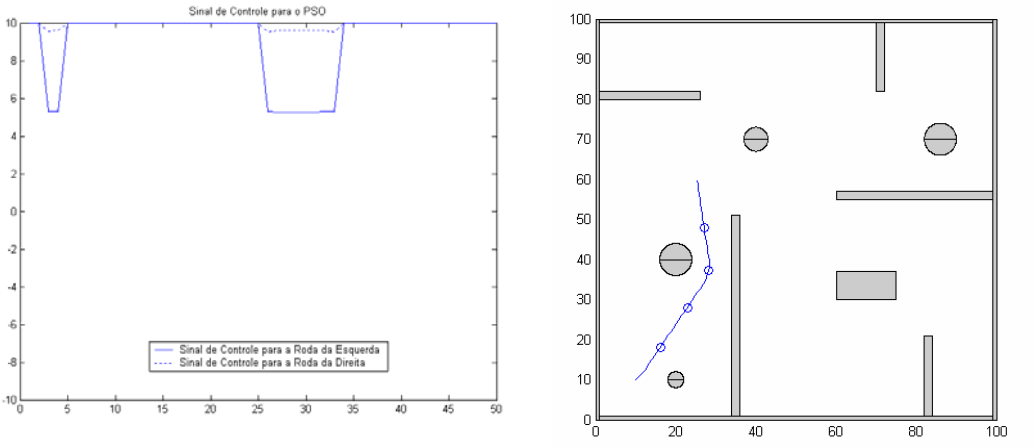

Figura 6.7: Sinal de controle e trajeto percorrido pelo PSO.

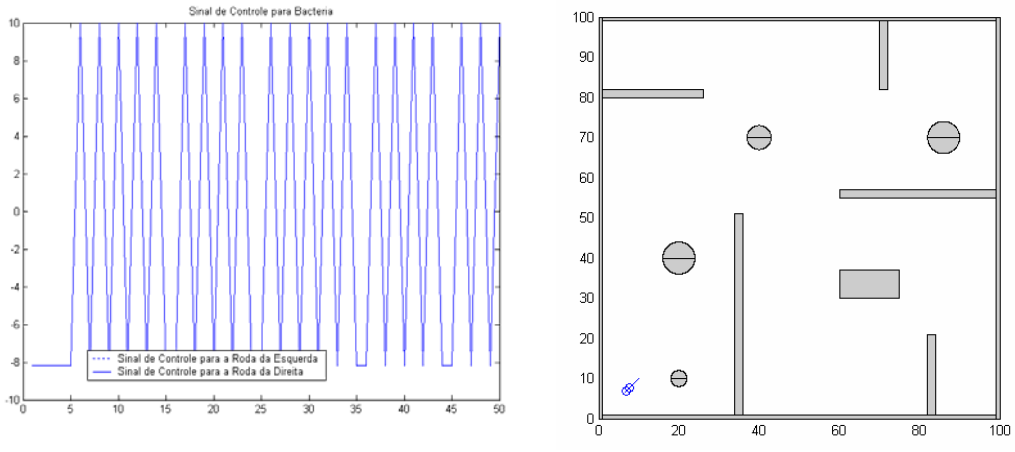

Figura 6.8: Sinal de controle e trajeto percorrido pelo algoritmo colônia de bactérias.

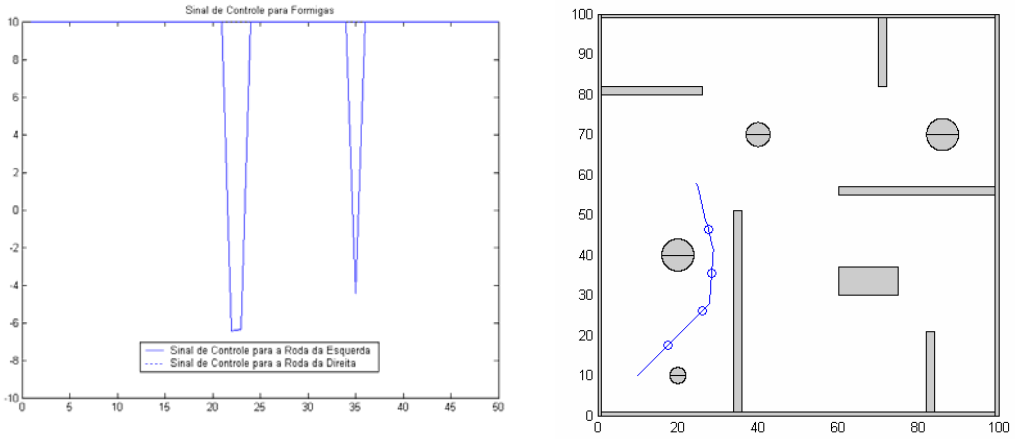

Figura 6.9: Sinal de controle e trajeto percorrido pelo algoritmo colônia de formigas.

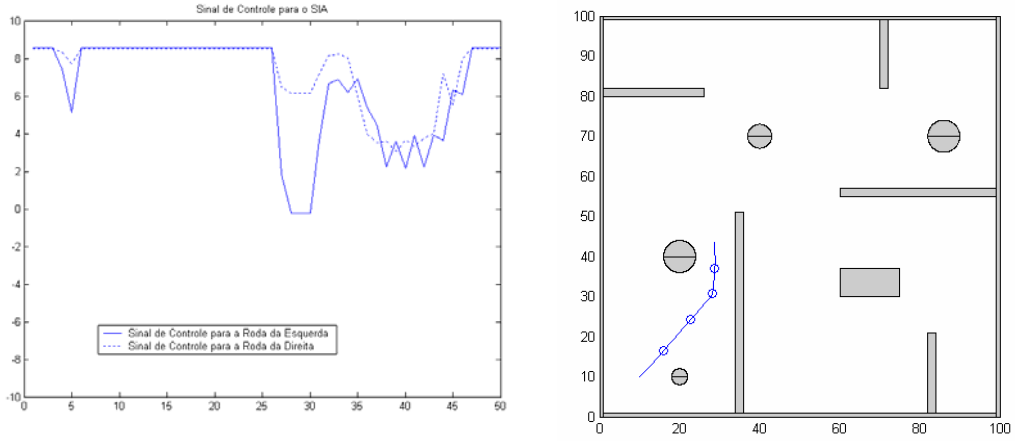

Figura 6.10: Sinal de controle e trajeto percorrido pelo algoritmo SIA.

Como pode ser visto pelos sinais de controle e pelos trajetos percorridos, pode-se perceber que os robôs móveis se deslocam sempre em frente (valores sempre positivos) em praticamente todos os algoritmos, exceto no algoritmo de colônia de bactérias, onde os valores positivos e negativos se alternam.

Isso quer dizer que em praticamente todos os algoritmos, os robôs se deslocam para a frente explorando o território, exceto no colônia de bactérias, onde o robô fica alternando o deslocamento para a frente e para trás.

Um fato importante deve ser ressaltado no controlador projetado pelo algoritmo PSO (Fig. 6.7), o robô, neste caso, se desloca para frente em grande velocidade, inclusive durante suas curvas, fato que não ocorre nos controladores projetados pelos outros algoritmos, onde durante as curvas uma das rodas sempre tem valor zero ou negativo. Pode-se perceber que o controlador obtido é eficaz para deslocamentos rápidos do robô, mas, em ambientes que possuam muitos obstáculos, este controlador pode fazer com que o robô necessite dar a ré muitas vezes, pelo fato de se aproximar de maneira muito rápida dos obstáculos e não ter espaço suficiente para fazer a curva e contorná-los, causando colisões.

Analisando a Fig. 6.8, onde são apresentados os sinais de controle e o trajeto percorrido pelo robô controlado pelo controlador projetado pelo algoritmo de colônia de bactérias, pode-se perceber que o controlador obtido não é eficaz, pois se percebe que os valores para os dois sinais de controle são sempre iguais, ou seja, o robô está sempre se deslocando em linha reta, e também que os sinais estão se alternando entre sinais positivos e negativos de forma cíclica. Ou seja, o robô está se deslocando um pouco para frente e um pouco para trás. Com isso é possível perceber que este controlador não é um controlador desejado para o presente estudo de caso, cujo objetivo é obter a maior exploração possível de um terreno previamente desconhecido.

Um fator a ser ressaltado no controlador obtido pelo algoritmo de colônia de formigas é o fato de o mesmo fazer com que o robô envie sinais de controle de sinais opostos para cada uma das rodas em um mesmo instante de tempo, como pode ser visto na Fig. 6.9, isto faz com que o robô possa fazer curvas de forma mais rápida e também sem se deslocar.

b. Evolução do *Fitness*

Nesta subseção são apresentados os gráficos de evolução do *fitness* e também gráficos contendo a performance *on-line* e *off-line* para os algoritmos. Na Fig. 6.11 são apresentados os gráficos referentes à evolução dos *fitness* médio e máximo para os algoritmos: genético,

PSO, colônia de bactérias, colônia de formigas e sistemas imunológicos artificiais, respectivamente.

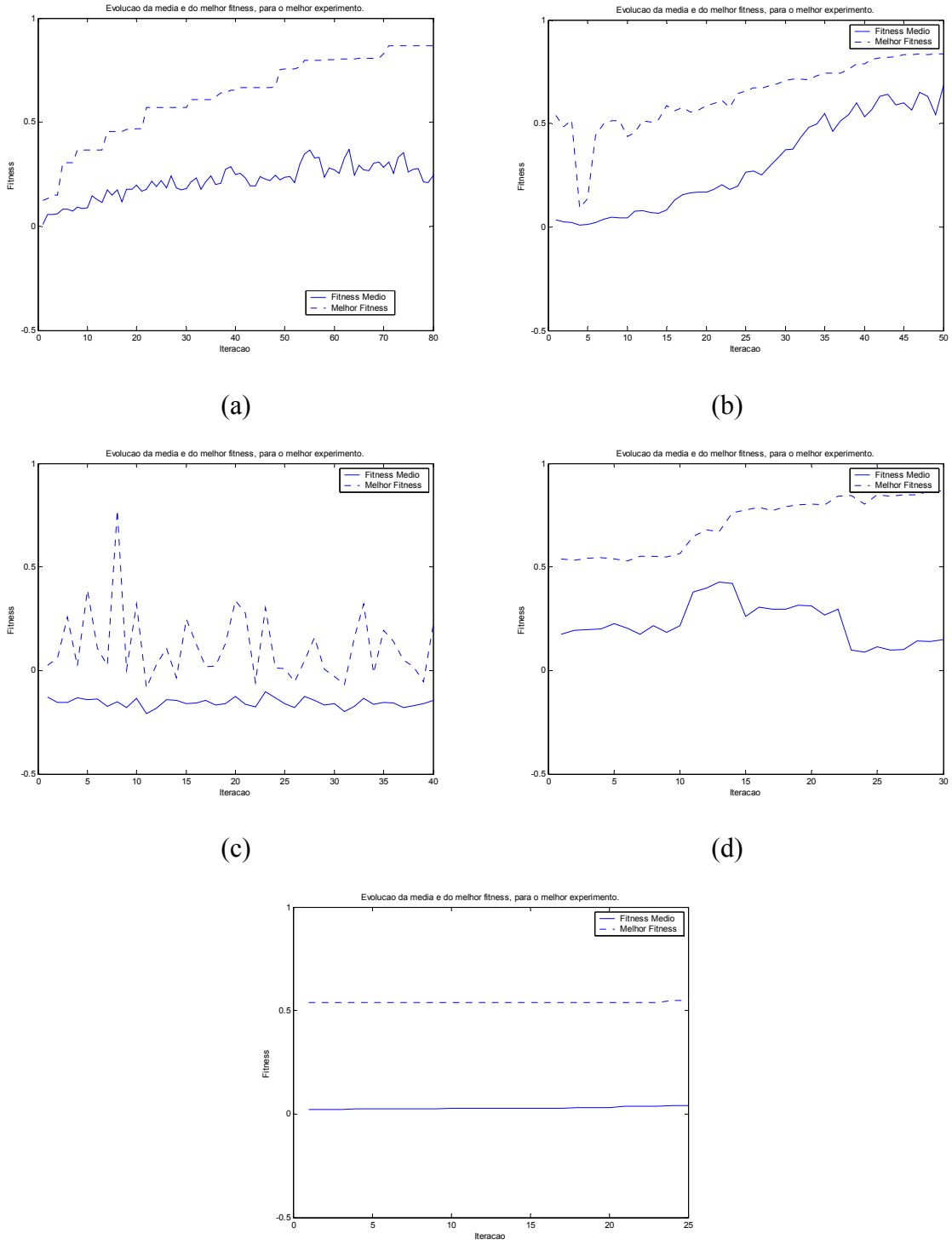

Figura 6.11: Evolução do *fitness* médio e máximo para os algoritmos.

Como pode ser visto através dos gráficos apresentados na Fig. 6.11, nenhum dos algoritmos está convergindo para uma solução, o que mais se aproxima de uma convergência é o algoritmo PSO (b). No AG (a) percebe-se que a grande diversidade da população prejudica a convergência da população, pois a média dos *fitness* começa a cair após um determinado momento, fato que se repete no algoritmo de colônia de formigas (d). A não convergência dos algoritmos pode ser em parte explicada pelo grande número de parâmetros a serem otimizados pelos algoritmos (44 parâmetros).

Analisando-se o conjunto de parâmetros utilizados para este estudo de caso, pode-se perceber que o AG (a) não consegue atingir a solução ótima para o estudo de caso. Porém, percebe-se que a sua utilização é de grande valia no que diz respeito a diversificar a população, pois permite uma exploração de grande parte do espaço de buscas, não ficando restrito a ótimos locais, ou seja, o algoritmo consegue encontrar a vizinhança do ponto de ótimo global, mas o mesmo não consegue atingir o mesmo.

Um fato importante a ser mencionado nos gráficos apresentados é o fato de o algoritmo de colônia de bactérias possuir valores de *fitness* médio negativos, como previamente descrito na seção 5.2.4 desta dissertação, a única possibilidade de o *fitness* ser negativo é através da primeira componente da equação (5.4), que é a soma da velocidade das duas rodas do robô, ou seja, neste caso a soma das rodas do robô está dando negativa. Conseqüentemente o robô está se deslocando em marcha ré. Além disso, percebe-se que as soluções obtidas por este algoritmo, em geral, não são boas, com poucas exceções onde alguns picos no *fitness* podem ser notados. Com isso, conclui-se que este algoritmo, com as seguintes configurações de parâmetros, não se possui bom desempenho para este estudo de caso. Uma análise paramétrica mais apurada é necessária de forma a verificar o grau de adequabilidade deste algoritmo com o presente estudo de caso.

No algoritmo de colônia de formigas (d) pode-se perceber que boas soluções são obtidas, mas ainda mantendo uma boa diversidade na população, que ajuda na busca de melhores soluções para o problema.

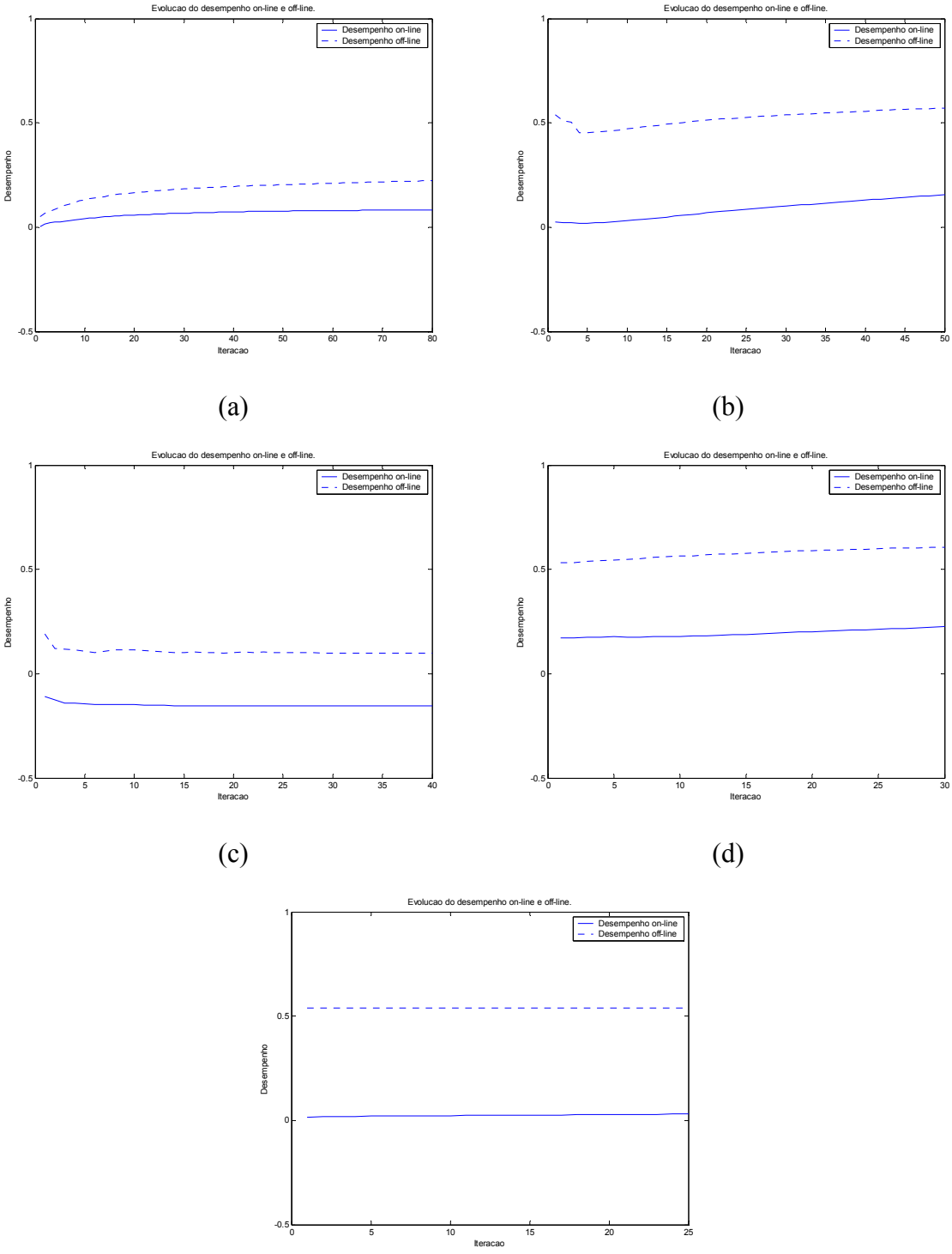

Figura 6.12: Desempenho *on-line* e *off-line* para os algoritmos.

As informações presentes nos gráficos de desempenho *on-line* e *off-line* apresentados na Fig. 6.12 apenas confirmam as conclusões retiradas da Fig. 6.11. Porém, devemos ressaltar alguns pontos referentes ao algoritmo de SIA.

Como pode ser visto nas Figs. 6.11 (e) e 6.12 (e), praticamente não há evolução nas soluções do algoritmo SIA ao longo das iterações. Uma possível explicação para esta falta de evolução é o parâmetro da quantidade de clones que são gerados a partir de cada indivíduo da população, como para este parâmetro foi selecionado um valor baixo, poucas soluções potenciais são geradas a cada iteração. Como o espaço de buscas do presente estudo de caso é muito grande (44 variáveis), estes clones acabam não encontram boas soluções. Com isso, a melhor solução atingida acaba permanecendo a solução inicialmente proposta.

A razão para a seleção deste valor baixo para o número de clones gerados por indivíduo é o fato de o mesmo influenciar diretamente, e com grande peso, no custo computacional do algoritmo. Como o ponto de maior custo computacional neste algoritmo é a análise da função *fitness* (simulação do movimento do robô), cada indivíduo da população a cada iteração gera um determinado número de clones, e cada um destes clones tem seu *fitness* avaliado. A opção por 3 clones por indivíduo da população foi tomada com base no custo computacional do mesmo, um comparativo do custo computacional destes algoritmos pode ser visto na Tabela 6.2 (pg. 98).

6.2.4. Simulações com o Controlador Projetado

Como pode ser visto na Tabela 6.2 (pg. 98), o algoritmo que atingiu a melhor solução foi o algoritmo de Colônia de Formigas, nesta seção é apresentado o controlador obtido nesta simulação e algumas simulações de movimentação de um Khepera sendo controlado pelo mesmo.

A melhor solução obtida pelo algoritmo de colônia de formigas é representada pelas funções de pertinência apresentadas nas Figs. 6.13 e 6.14.

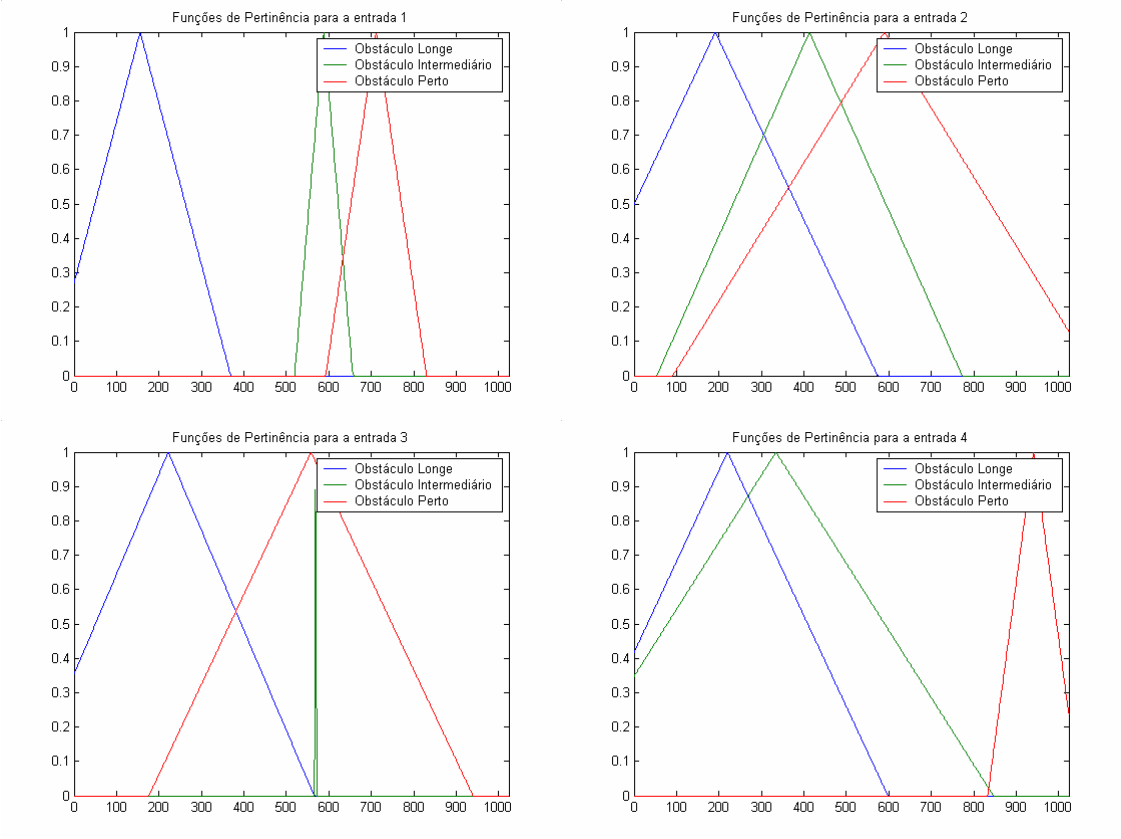

Figura 6.13: Funções de pertinência para as entradas da melhor solução de Col. de Formigas.

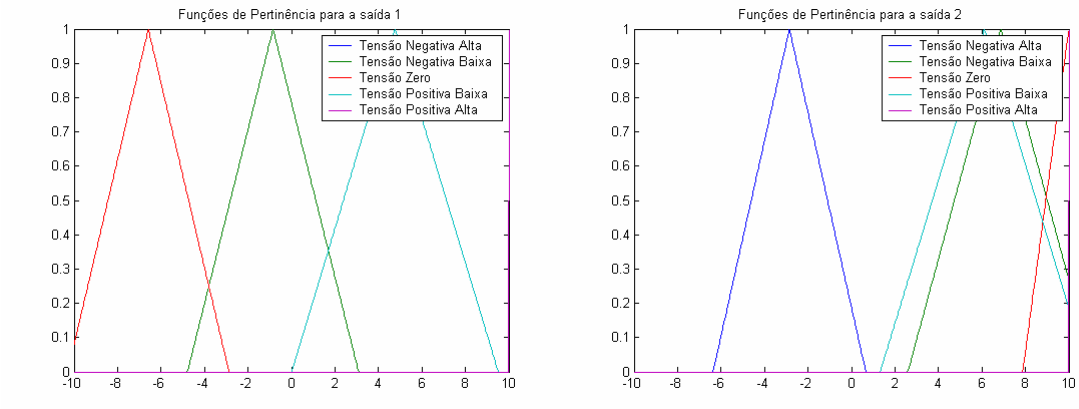

Figura 6.14: Funções de pertinência para as saídas da melhor solução de Col. de Formigas.

Como pode ser visto, existe o problema de superposição das funções de pertinência para o tanto para as entradas quanto para as saídas deste controlador obtido, a implementação

de uma ferramenta de restrição a estas superposições é encorajada como forma de melhorar a solução, a mesma é deixada como uma perspectiva de trabalho futuro nesta dissertação.

Esta controlador obtido foi então implantado no simulador do Khepera como forma de verificar a sua performance tanto no ambiente onde o controlador foi projetado quanto em ambiente desconhecidos. Nas Figs. 6.15 a 6.17 são apresentados alguns dos trajetos percorridos pelo Khepera ao longo destas simulações.

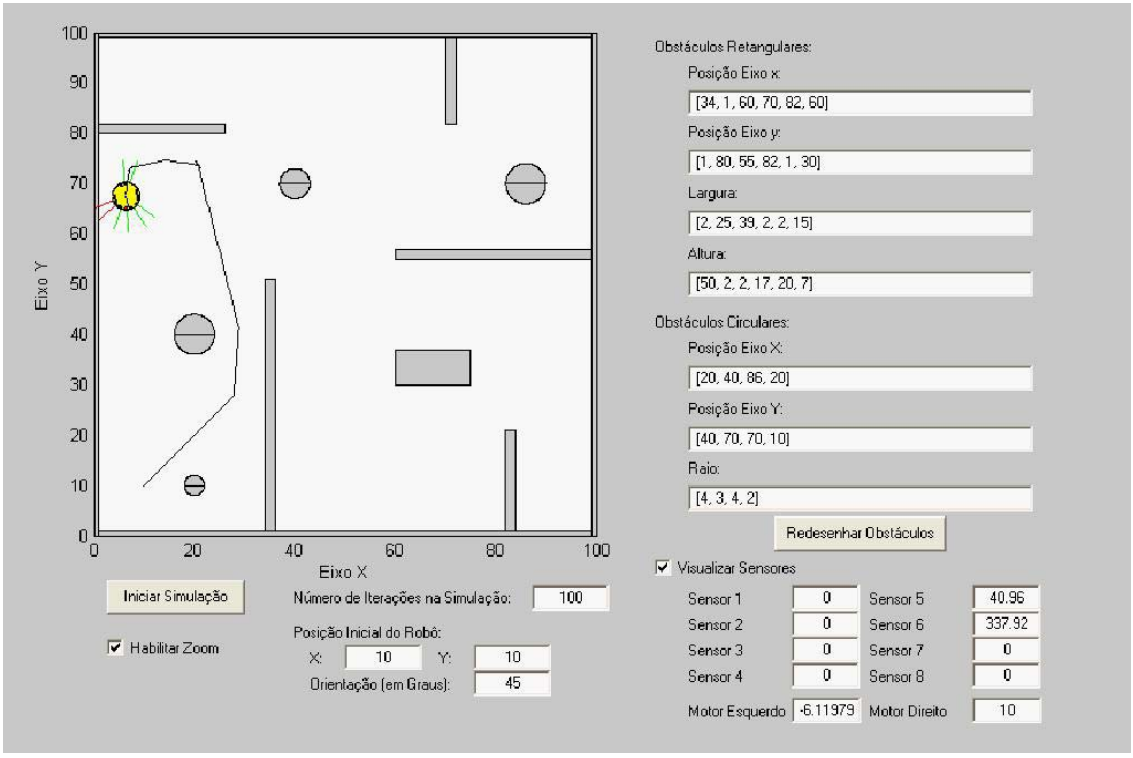

Figura 6.15: Caminho do robô no ambiente utilizado para desenvolver o controlador.

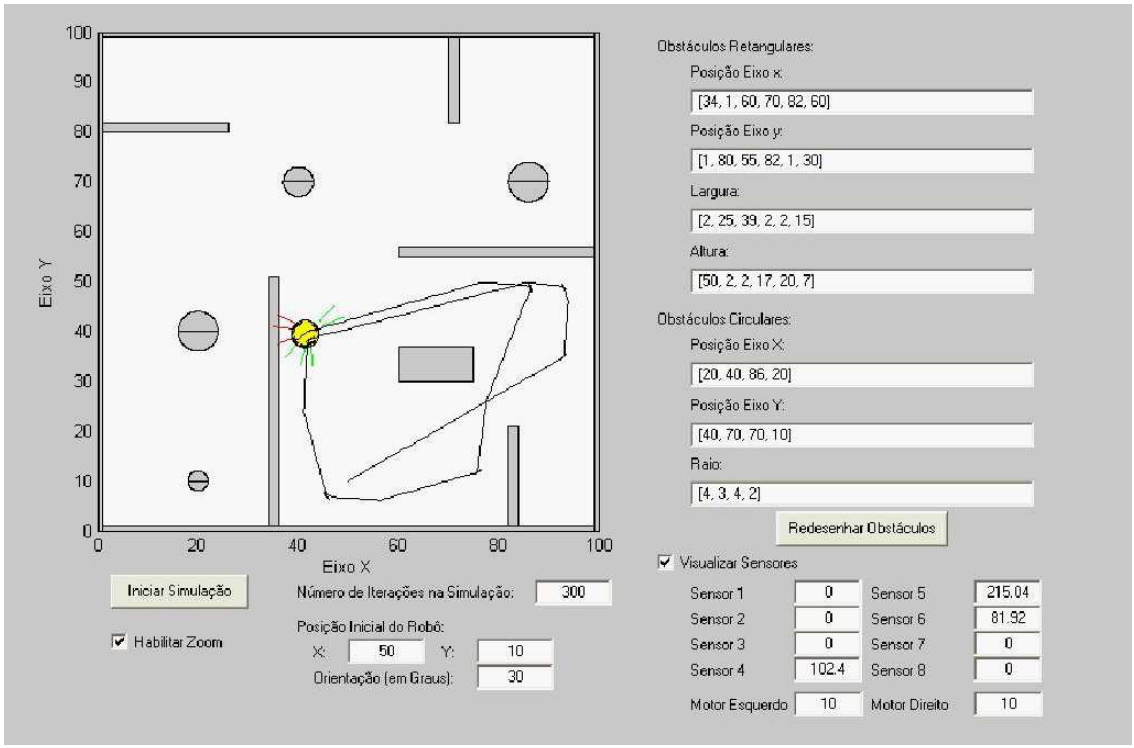

Figura 6.16: Caminho do robô no mesmo ambiente, com outro ponto de início.

Foi optado pelo teste também em um ambiente totalmente desconhecido pelo controlador, portanto, foi desenvolvido outro ambiente de forma a verificar se o desempenho do robô na exploração é satisfatório. Na Fig. 6.17, é apresentado um novo ambiente, com mais obstáculos, para verificar se o robô consegue desviar das paredes do mesmo.

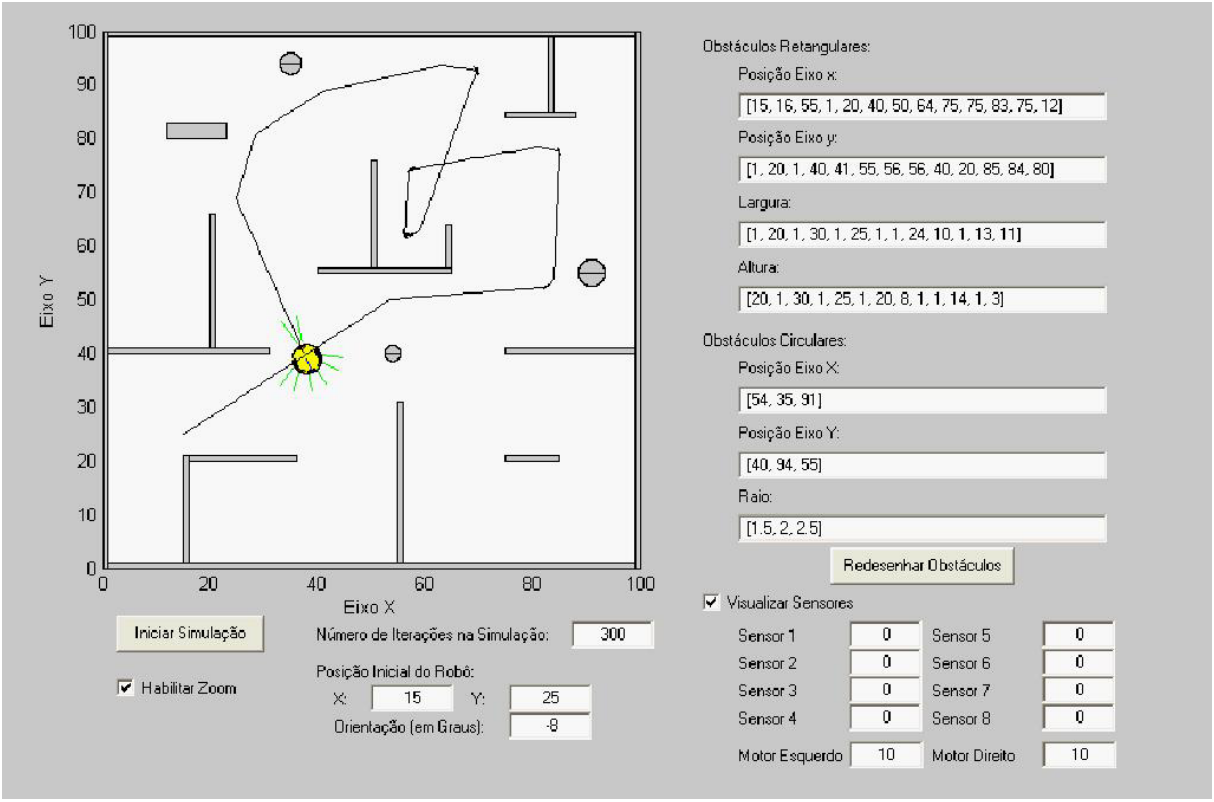

Figura 6.17: Caminho do robô em um ambiente não utilizado para desenvolver o controlador.

Analisando os gráficos apresentados nas Figs. 6.15 a 6.17, pode-se afirmar que o controlador desenvolvido funciona adequadamente para controlar um robô móvel Khepera com o intuito de desviar de obstáculos e de explorar o território.

Outra análise pode ser feita de forma a verificar o custo computacional de cada um dos algoritmos utilizados para otimizar o controlador. Analisando os resultados apresentados na Tabela 6.2, pode-se verificar que qualquer um dos algoritmos possui um elevado custo computacional. O algoritmo mais rápido levou, em média, um tempo próximo a 5 horas (PSO), pode-se ver também que o algoritmo de Sistemas Imunológicos Artificiais leva em média 12 horas e 47 minutos para cada execução.

Quando se leva em conta tanto a complexidade computacional quanto os resultados obtidos, podemos concluir que dois algoritmos, principalmente, são recomendados para este estudo de caso, o algoritmo de colônia de formigas e o algoritmo PSO. O algoritmo de colônia de formigas consegue obter melhores soluções, mas possui um custo computacional maior. Por outro lado, caso se opte pelo menor custo computacional o algoritmo PSO é preferível,

pois seu custo computacional é quase a metade do custo do algoritmo de colônia de formigas, e mesmo assim suas soluções não ficam tão distantes das obtidas por este. Deve-se ressaltar, porém, que estas conclusões são válidas para os algoritmos configurados com os parâmetros apresentados na seção 6.2.1 desta dissertação, para se obter conclusões mais gerais, uma análise paramétrica é requerida.

Capítulo 7

Conclusão e Pesquisas Futuras

O planejamento de trajetos é um dos principais temas a serem estudados na área de automação de robôs móveis, sendo que este planejamento de trajetos abrange a análise da menor trajetória a ser percorrida ao longo de um ambiente conhecido e também o desvio de obstáculos.

O objetivo desta dissertação é contribuir com uma análise de diversas técnicas de inteligência computacional para problemas de planejamento de trajetórias em robótica móvel. Para esta análise comparativa foram usados os algoritmos genéticos, nuvem de partículas, colônia de bactérias, colônia de formigas e sistemas imunológicos artificiais.

Analisando os resultados apresentados no Capítulo 6, é possível perceber que cada uma das técnicas utilizadas possui seus pontos fortes e fracos. Seguem algumas considerações sobre as mesmas. Deve-se ressaltar, porém, que todas as conclusões apresentadas são referentes aos conjuntos de parâmetros selecionados para os algoritmos, a seleção destes parâmetros de forma diferenciada pode causar grandes diferenças nas soluções obtidas e, consequentemente, nas conclusões inferidas.

O algoritmo genético tem como principal característica realizar a busca de soluções através de transições probabilísticas, sendo que esta característica pode ser considerada tanto como uma potencialidade como uma limitação do algoritmo. Esta “aleatoriedade” permite uma maior diversidade da população, fato extremamente útil em problemas não-lineares. Por outro lado, esta aleatoriedade faz com o algoritmo possua algumas dificuldades em convergir para uma solução em problemas de elevada complexidade, ou seja, em problemas com grandes espaços de busca.

O algoritmo de nuvem de partículas tem sua fundamentação teórica no fato de elementos do grupo seguirem o comportamento do melhor indivíduo do mesmo. Isto representa uma grande potencialidade para a convergência do algoritmo para uma solução. O algoritmo é muito bom para realizar otimizações em pequenas vizinhanças. Porém, este mesmo fato se torna uma limitação quando o problema é composto por um espaço de busca grande, pois o algoritmo pode acabar convergindo para um ótimo local. Outra grande potencialidade deste algoritmo frente aos outros apresentados nesta dissertação é o pequeno número de parâmetros a serem otimizados e o reduzido custo computacional do algoritmo, quando comparado com as outras técnicas.

O algoritmo colônia de bactérias, como o próprio nome diz, simula o comportamento das bactérias no meio ambiente. Nesta simulação, os indivíduos da população são avaliados diversas vezes, permitindo que os indivíduos não acabem presos em ótimos locais do espaço de buscas. Por isso mesmo, pode-se considerar este algoritmo adequado para realizar otimizações em grandes espaços de busca. Porém, devido à forma como é realizada a simulação das bactérias, "continuar o movimento do indivíduo enquanto for vantajoso" (Fig. 3.2) o *fitness* do último ponto será sempre pior que o do penúltimo, isso faz com que o algoritmo não sirva para otimizações cujo objetivo sejam o "afinamento" de boas soluções já obtidas. Outra grande limitação deste algoritmo é o elevado número de parâmetros de projeto a serem configurados no algoritmo.

O algoritmo de colônia de formigas em ambos os estudos de caso foi o algoritmo que conseguiu obter as melhores soluções. Porém, o mesmo não consegue convergir para a solução, como pode ser visto nos resultados apresentados no Capítulo 6. Isto demonstra o grande potencial de otimização desta técnica. Porém, devido às suas limitações de convergência, pode-se levar a concluir que este algoritmo seria mais bem aproveitado quando desenvolvido um algoritmo híbrido com outro algoritmo com boas características de otimização local.

O algoritmo de SIA leva em consideração que os indivíduos que têm maior afinidade do objetivo (*fitness*) sofrerão menos mutações do que aqueles com menor *fitness*, fato que possibilita uma boa diversidade nos indivíduos da população. Esta grande diversidade de indivíduos possibilita a obtenção de bons resultados em problemas de complexidade relativamente baixa. Porém, em problemas de elevada complexidade, o algoritmo necessitaria de uma população muito grande para conseguir obter boas soluções, como pode ser visto nos resultados obtidos no segundo estudo de caso.

Com base nestas informações referentes a cada um dos algoritmos apresentados nesta dissertação, é possível elaborar uma proposta de algoritmo de otimização híbrido entre estas técnicas. As duas possibilidades que aparentam ter maiores probabilidades de sucesso são a utilização dos conjuntos de algoritmos: “Colônia de Formigas + Nuvem de Partículas” e “Colônia de Bactérias + Nuvem de Partículas”, com maior ênfase na primeira opção. O que nos leva a acreditar nesta possibilidade é o fato do primeiro algoritmo possuir boas características de encontrar uma boa solução, porém sem convergência para esta, e do segundo algoritmo possuir boas características de convergência, tomando como base a solução previamente obtida pelo primeiro algoritmo. Porém, deve-se ressaltar uma limitação básica para esta solução híbrida, como ambos os algoritmos possuem um custo computacional que não pode ser desprezado, conforme Tabela 6.2, o custo computacional do algoritmo híbrido será bastante elevado.

O que foi apresentado até o momento é apenas parte do que pode ser descoberto na área abordada nesta dissertação. Existem ainda diversos campos de pesquisa que podem ser abrangidos, alguns exemplos são apresentados a seguir.

Pode-se utilizar os algoritmos apresentados ao longo desta dissertação para otimizar problemas com múltiplos objetivos; desenvolver técnicas de otimização híbridas entre os algoritmos apresentados; implementar os estudos de caso apresentados em robôs móveis reais; desenvolvimento de um novo estudo de caso mais complexo englobando toda a simulação real de um robô móvel, i.e. encontrar o melhor trajeto a ser seguida pelo robô em um ambiente previamente desconhecido, com reconhecimento do ambiente através de sensores; problemas de cooperação de robôs para atingir um determinado objetivo; e desenvolvimento de uma análise paramétrica para os algoritmos apresentados, como forma de garantir o seu melhor desempenho para cada estudo de caso individualmente.

Referências Bibliográficas

Adler, F. R.; Gordon, D. M. (2003) **Optimization, Conflict and Nonoverlapping Foraging Ranges in Ants**, *Proceedings of the American Naturalist*, Chicago, Estados Unidos, V. 162, no. 5, pp. 529-543.

Agassounon, W.; Martinoli, A.; Easton, K. (2004) **Macroscopic Modeling of Aggregation Experiments using Embodied Agents in Teams of Constant and Time-Varying Sizes**, *Autonomous Robots*, V. 17, pp. 163-192.

Aguirre, A. H.; Coello, C. A. (2003) **Evolutionary Synthesis of Logic Circuits Using Information Theory**, *Artificial Intelligence Review*, V. 20, pp. 445-471.

Aickelin, U.; White, P. (2004) **Building Better Nurse Scheduling Algorithms**, *Annals of Operations Research*, V. 128, pp. 159-177.

Alcalá, R.; Benitez, J. M.; Casillas, J.; Cordón, O.; Pérez, R. (2003) **Fuzzy Control of HVAC Systems Optimized by Genetic Algorithms**, *Applied Intelligence*, V. 18, pp. 155-177.

Alves, F. O.; Osório, F. S. (2003) **Aplicação da Abordagem SMPA para a Navegação de um Robô Móvel Autônomo**, *IV Encontro Nacional de Inteligência Artificial – IV ENIA*, Campinas, SP, V. 1, pp. 1-6.

Ando, S.; Iba, H. (2003) **Artificial Immune System for Classification of Cancer**, *EvoWorkshops* 2003, Essex, Inglaterra, pp. 1-10.

Avila, S. L. (2001) **Algoritmos Genéticos Aplicados na Otimização de Antenas Refletoras**, *Dissertação de Mestrado em Engenharia Elétrica – Universidade Federal de Santa Catarina*, Florianópolis, SC.

Back, T.; Hammel, U.; Schwefel, H-P. (1997) **Evolutionary Computation: Comments on the History and Current State**, *IEEE Transactions on Evolutionary Computation***,** V. 1, no. 1, pp. 3-17.

Baltes, J.; Hildreth, N. (2001) **Adaptive Path Planner for Highly Dynamic Environments**, *RoboCup-2000: Robot Soccer World Cup IV*, Melbourne, Austrália, pp. 76-85.

Banzhaf, W. (1998) **Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications**, San Francisco: Morgan Kaufmann Publishers, pp. 470.

Barberá, H. M.; Skarmeta, A. F. G. (2001) **Una Arquitectura Distribuida para el Control de Robots Autónomos Móviles**, *Tese de Doutorado – University of Murcia*, Murcia, Espanha.

Bergh, F.; Engelbrecht, A. P. (2000) **Cooperative Learning in Neural Networks using Particle Swarm Optimizers**, *South African Computer Journal*, V. 26, pp. 84-90.

Bergman, C. M.; Fryxell, J. M.; Gates, C. C.; Fortin, D. (2001) **Ungulate Foraging Strategies: Energy Maximizing or Time Minimizing?**, *Proceedings of the Journal of Animal Ecology*, V. 70, pp. 289-300.

Bezdek, J. C. (1994) **What is Computational Intelligence?**, *Computational Intelligence Imitating Life*, New York, Estados Unidos, pp. 1-12.

Blackwell, T. M. (2003) **Swarm in Dynamic Environments**, *Genetic and Evolutionary Computation Conference – GECCO 2003*, Chicago, Estados Unidos, pp. 1-12.

Bobel, E. (2003) **Dimensionamento e Locação de Equipes de Manutenção em Redes de Distribuição de Energia Elétrica**, *Dissertação de Mestrado – CEFET-PR*, Curitiba, PR.

Bonabeau, E.; Dorigo, M.; Theraulaz, G. (1999) **Swarm Intelligence: From Natural to Artificial Systems**, New York: Oxford University Press, pp. 307.

Bonissone, P. P.; Chen, Y.; Goebel, K.; Khedkar, P. S. (1999) **Hybrid Soft Computing Systems: Industrial and Commercial Applications**, *Proceedings of the IEEE*, V. 87, no. 9, pp. 1641-1667.

Boughanem, M.; Chrisment, C.; Tamine, L. (1999) **Genetic Approach to Query Space Exploration**, *Information Retrieval*, V. 1, pp. 175-192.

Bradley, D. W.; Tyrrell, A. M. (2000) **Immunotronics: Hardware Fault Tolerance Inspired by the Immune System**, *International Conference on Evolvable Systems – ICES 2000*, Edinburgh, Escócia, pp. 11-20.

Braendler, D.; Hendtlass, T. (2002) **The Suitability of Particle Swarm Optimisation for Training Neural Hardware**, *The Fifteenth International Conference on Industrial & Engineering Application of Artificial Intelligence & Expert Systems – IEA/AIE 2002*, Cairns, Austrália, pp. 190-199.

Brandstätter, B.; Baumgartner, U. (2002) **Particle Swarm Optimization – Mass-Spring Systems Analogon**, *IEEE Transactions on Magnetics*, V. 38, pp. 997-1000.

Buckmann, O.; Krömker, M.; Berger, U. (1998) **An Application Platform for the Development and Experimental Validation of Mobile Robots for Health Care Purposes**, *Journal of Intelligent and Robotic Systems*, V. 22, pp. 331-350.

Cai, Z.; Peng, Z. (2002) **Cooperative Coevolutionary Adaptative Genetic Algorithm in Path Planning of Cooperative Multi-Mobile Robot Systems**, *Journal of Intelligent and Robotic Systems*, V. 33, pp. 61-71.

Canham, R. O.; Tyrrell, A. M. (2003) **A Hardware Artificial Immune System and Embryonic Array for Fault Tolerant Systems**, *Genetic Programming and Evolvable Machines*, V. 4, pp. 359-382.

Cao, Y. U.; Fukunaga, A. S.; Kahng, A. B. (1997) **Cooperative Mobile Robotics: Antecedents and Directions**, *Autonomous Robots*, V. 4, pp. 7-27.

Carvalho, D. R.; Freitas, A. A. (2000) **O que são Sistemas Imunológicos Artificiais e suas Aplicações**, *Tuiuti: Ciência e Cultura*, Curitiba, PR, V. 19, pp. 63-74.

Carruthers, B.; McGookin, E. W.; Murray-Smith, D. J. (2005) **Adaptive Evolutionary Search Algorithm with Obstacle Avoidance for Multiple UAVs**, *IFAC 2005*, Pague, República Checa.

Castro, R. E. (2001) **Otimização de Estruturas com Multi-Objetivos Via Algoritmos Genéticos de Pareto**, *Tese de Doutorado, Programa de Engenharia Civil* – COPPE/UFRJ, Rio de Janeiro, RJ, pp. 202.

Chang, S. J.; Li, T-H. S. (2002) **Design and Implementation of Fuzzy Parallel-Parking Control for a Car-Type Mobile Robot**, *Journal of Intelligent and Robotic Systems*, V. 34, pp. 175-194.

Chao, D. L.; Forrest, S. (2003) **Information Immune System**, *Genetic Programming and Evolvable Machines*, V. 4, pp. 311-331.

Chang, B. C. H.; Ratnaweera, A.; Halgamuge, S. K.; Watson, H. C. (2004) **Particle Swarm Optimisation for Protein Motif Discovery**, *Genetic Programming and Evolvable Machines*, V. 5, pp. 203-214.

Ciocirlan, B. O.; Marghitu, D. B.; Beale, D. G.; Overfelt, R. A. (1998) **Dynamics and Fuzzy Control of a Levitated Particle**, *Nonlinear Dynamics*, V. 17, pp. 61-76.

Coelho, L. S.; Sierakowski, C. A. (2005) **Bacteria Colony Approaches with Variable Speed Applied to Path Planning Optimization of Mobile Robots**, *International Congress on Mechanical Engineering – COBEM 2005*, Ouro Preto, MG, V. 1, pp. 1-8.

Colorni, A.; Dorigo, M.; Maniezzo, V. (1991) **Distributed Optimization by Ant Colonies**, *Proceedings of ECAL91 – European Conference on Artificial Life*, Paris, França, pp. 134-142.

Cordón, O.; Herrera, F.; Stützle, T. (2002) **A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends**, *Mathware & Soft Computing*, V. 9, no. 2-3, pp. 141-175.

Corno, F.; Sanchez, E.; Reorda, M. S.; Squillero, G. (2004) **Code Generation for Functional Validation of Pipelined Microprocessors**, *Journal of Electronic Testing: Theory and Applications*, V. 20, pp. 269-278.

Corry, P.; Kozan, E. (2004) **Ant Colony Optimisation for Machine Layout Problems**, *Computational Optimization and Applications*, V. 28, pp. 287-310.

Darwin, C. (1859) **The Origins of Species**, London: John Murray, pp. 501.

Dasgupta, D.; Michalewicz, Z. (2001) **Evolutionary Algorithms in Engineering Problems**, pp. 554.

Dasgupta, D.; Ji, Z.; González, F. (2003) **Artificial Immune Systems (AIS) Research in the Last Five Years**, *Proceedings of the International Conference on Evolutionary Computation Conference*, V. 1, pp. 131-138.

Dasgupta, D.; Krishnakumar, K.; Wong, D.; Berry, M. (2004) **Negative Selection Algorithm for Aircraft Fault Detection**, *3rd International Conference on Artificial Immune Systems – ICARIS 2004*, Catania, Itália, pp. 1-13.

Dasgupta, D. (1998) **Evolving Neuro-Controllers for a Dynamic System Using Structured Genetic Algorithms**, *Applied Intelligence*, V. 8, pp. 113-121.

Davison, A. J.; Kita, N. (2002) **Active Visual Localization for Multiple Inspection Robots**, *Advanced Robotics*, V. 16, no. 3, pp. 281-295.

De Jong, K. (1975) **An Analysis of the Behaviour of a Class of Genetic Adaptaive Systems**, *Tese de PhD – University of Michigan*.

Delgado, M. R. B. S. (2002) **Projeto Automático de Sistemas Nebulosos: Uma Abordagem Co-Evolutiva**, *Tese de Doutorado – Universidade Estadual de Campinas*, Campinas, SP.

Dhariwal, A.; Sukhatme, G. S.; Requicha, A. A. G. (2004) **Bacterium-inspired Robots for Environmental Monitoring,** *Proceeding of the 2004 IEEE International Conference on Robotics & Automation*, V.2, pp. 1436-1443.

Dias, A. H. F. (2000) **Algoritmos Genéticos Aplicados a Problemas com Múltiplos Objetivos**, *Dissertação de Mestrado em Engenharia Elétrica – Universidade Federal de Minas Gerais*, Belo Horizonte, MG, pp. 136.

Diéguez, A. R.; Sanz, R.; López, J. (2003) **Deliberative On-Line Local Path Planning for Autonomous Mobile Robots**, *Journal of Intelligent and Robotic Systems*, V. 37, pp. 1-19.

Doctor, S.; Venayagamoorthy, G. K.; Gudise, V. G. (2004) **Optimal PSO for Collective Robotic Search Applications**, *Evolutionary Computation*, V. 2, pp. 1390-1395.

Doerner, K.; Gutjahr, W. J.; Hartl, R. F.; Strauss, C.; Stummer, C. (2004) **Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection**, *Annals of Operations Research*, V. 131, pp. 79-99.

Dorigo, M.; Maniezzo, V.; Colorni, A. (1996) **Ant System: Optimization by a Colony of Cooperating Agents**, *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, V. 26, no. 1, pp. 29-41.

Dorigo, M.; Di Caro, G.; Gambardella, L. M. (1999) **Ant Algorithms for Discrete Optimization**, *Artificial Life*, Brussels, Belgium, V. 5, no. 3, pp. 137-172.

Driessen, B. J.; Feddema, J. T.; Kwok, K. S. (1999) **Decentralized Fuzzy Control of Multiple Nonholonomic Vehicles**, *Journal of Intelligent and Robotic Systems*, V. 26, pp. 65-78.

Duarte, C. Tome, J. A. B. (2000) **An Evolutionary Strategy for Learning in Fuzzy Network**, *19th International Conference of the North American Fuzzy Information Processing Society*, Atlanta, Estados Unidos, pp. 24-28.

Durand, N.; Alliot, J-M.; Médioni, F. (2000) **Neural Nets Trained by Genetic Algorithms for Collision Avoidance**, *Applied Intelligence*, V. 13, pp. 205-213.

Eberhart, R. C.; Shi, Y. (2000) **Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization**, *Proceedings of the 2000 Congress on Evolutionary Computation*, San Diego, Estados Unidos, V. 1, pp. 84-88.

Ellabib, I.; Basir, O. A.; Calamai, P. (2002) **An Experimental Study of a Simple Ant Colony System for the Vehicle Routing Problem with Time Windows**, *Proceedings of the Third International Workshop on Ant Algorithms*, Brussels, Bélgica, pp. 53-64.

Erden, M. S.; Leblebicioglu, K.; Halici, U. (2004) **Multi-Agent System-Based Fuzzy Controller Design with Genetic Tuning for a Mobile Manipulator Robot in the Hand Over Task**, *Journal of Intelligent and Robotic Systems*, V. 39, pp. 287-306.

Esmin, A. A. A.; Aoki, A. R.; Lambert-Torres, G. (2002) **Particle Swarm Optimization for Fuzzy Membership Functions Optimization**, *Proceedings of the IEEE International Conference on System, Man and Cybernetics*, V. 3, pp. 108-113.

Ferentinos, K. P.; Arvanitis, K. G.; Sigrimis, N. (2002) **Heuristic Optimization Methods for Motion Planning of Autonomous Agricultural Vehicles**, *Journal of Global Optimization*, V. 23, pp. 155-170.

Filho, J. L. R.; Treleaven, P. C. (1994) **Genetic-Algorithm Programming Environments**, *IEEE Computer*, V. 27, no. 6.

Filho, J. L. R. (1996) **Algoritmos Genéticos**, *Tutorial do III Simpósio Brasileiro de Redes Neurais*, Recife, PE.

Floreano, D.; Mondada, F. (1996) **Evolution of Homing Navigation in a Real Mobile Robot**, *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, V. 26, no. 3, pp. 396-407.

Fogel, D. B.; Fogel, L. J. (1990) **Optimal Routing of Multiple Autonomous Underwater Vehicles Through Evolutionary Programming**, *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology*, Washington, Estados Unidos, pp. 44-47.

Fong, T.; Nourbakhsh, I.; Dautenhahn, K. (2003) **A Survey of Socially Interactive Robots**, *Robotics and Autonomous Systems*, pp.143-166.

Fourie, P. C.; Groenwold, A. A. (2002) **The Particle Swarm Optimization Algorithm in Size and Shape Optimization**, *Structure Multidisciplinary Optimization*, V. 23, pp. 259-267.

Fujimori, A.; Nikiforuk, P. N.; Gupta, M. M. (1997) **Adaptative Navigation of Mobile Robots with Obstacle Avoidance**, *IEEE Transactions on Robotics and Automation*, V. 13, no. 4, pp. 596-602.

Gaing, Z. (2004) **A Particle Swarm Optimization Approach for Optimum Design of PID Controller in AVR System**, *IEEE Transactions on Energy Conversion*, V. 19, no. 2, pp. 384-391.

Gallina, P.; Gasparetto, A. (2000) **A Technique to Analytically Formulate and to Solve 2-Dimensional Constrained Trajectory Planning Problem for a Mobile Robot**, *Journal of Intelligent and Robotic Systems*, V. 27, pp. 237-262.

Garrett, S. M. (2005) **How do We Evaluate Artificial Immune Systems**, *Evolutionary Computation*, Cidade, País, V. 13, no. 2, pp. 145-178.

Garber, P. A. (1987) **Foraging Strategies Among Living Primates**, *Proceedings of the Annual Review of Anthropology*, V. 16, pp. 339-364.

Ge, S. S.; Cui, Y. J. (2002) **Dynamic Motion Planning for Mobile Robots Using Potential Field Method**, *Autonomous Robots*, V. 13, pp. 207-222.

Glibovets, N. N.; Medvid, S. A. (2003) **Genetic Algorithms Used to Solve Scheduling Problems**, *Cybernetics and System Analysis*, V. 39, no. 1, pp. 81-90.

Goldbarg, M. C.; Luna, H. P. L. (2000) **Otimização Combinatória e Programação Linear**, Rio de Janeiro: Campus.

González, F. A.; Dasgupta, D. (2003) **Anomaly Detection Using Real-Valued Negative Selection**, *Genetic Programming and Evolvable Machines*, V. 4, pp. 383-403.

Gudise, V. G.; Venayagamoorthy, G. K. (2003) **Evolving Digital Circuits Using Particle Swarm**, *Proceedings of the International Joint Conference on Neural Networks*, Portland, Estados Unidos, V. 1, pp. 468-472.

Haït, A.; Simeon, T.; Taïx, M. (2002) **Algorithms for Rough Terrain Trajectory Planning**, *Advanced Robotics*, V. 16, no. 8, pp. 673-699.

Heck, P. S.; Ghosh, S. (2002) **The Design and Role of Synthetic Creative Traits in Artificial Ant Colonies**, *Journal of Intelligent Robotic Systems*, V. 33, pp. 343-370.

Hendzel, Z. (2004) **Fuzzy Combiner of Behaviors for Reactive Control of Wheeled Mobile Robot**, *ICAISC – International Conference on Artificial Intelligence and Soft Computing*, Zakopane, Polônia, pp. 774-779.

Holland, J. H. (1975) **Adaptation in Natural and Artificial Systems**, *Ann Arbor, University of Michigan Press*, Michigan, Estados Unidos, 1975.

Hoppenot, P.; Colle, E. (2002) **Mobile Robot Command by Man-Machine Co-operation – Application to Disabled and Elderly People Assistance**, *Journal of Intelligent and Robotic Systems*, V. 34, pp. 235-252.

Howard, D. W.; Zilouchian, A. (1998) **Application of Fuzzy Logic for the Solution of Inverse Kinematics and Hierarchical Controls of Robotic Manipulators**, *Journal of Intelligent and Robotic Systems*, V. 23, pp. 217-247.

Hu, X.; Eberhart, R. C.; Shi, Y. (2003) **Engineering Optimization with Particle Swarm**, *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, Indianapolis, Estados Unidos, pp. 53-57.

Huntsberger, T. L.; Trebi-Ollennu, A.; Aghazarian, H.; Schenker, P. S.; Pirjanian, P.; Nayar, H. (2004) **Distributed Control of Multi-Robot Systems Engaged in Tightly Coupled Tasks**, *Autonomous Robots*, V. 17, pp. 79-92.

Hunter, A. (1998) **Crossing Over Genetic Algorithms: The Sugal Generalised GA**, *Journal of Heuristics*, V. 4, pp. 179-192.

Ijspeert, A. J.; Martinoli, A.; Billard, A.; Gambardella, L. M. (2001) **Collaboration Through the Exploitation of Local Interactions in Autonomous Collective Robotics: The Stick Pulling Experiment**, *Autonomous Robots*, V. 11, pp. 149-171.

Intza, J. U. (2000) **Evolutionary Adaptive Robots: Artificial Evolution of Adaptation Mechanisms for Autonomous Agents**, *Département d'informatique – École Polytechnique Fédérale de Lausanne*, Switzerland.

Ionita, S.; Sofron, E. (2002) **The Fuzzy Model for Aircraft Landing Control**, *AFSS International Conference on Fuzzy Systems*, Calcutta, Índia, pp. 47-54.

Janson, S.; Middendorf, M. (2004) **A Hierarchical Particle Swarm Optimizer for Dynamic Optimization Problems**, *EvoWorkshop 2004*, Coimbra, Portugal, pp. 513-524.

Jantzen, J. (1998) **Design of Fuzzy Controllers**, *Curso on-line de Fuzzy*, http://fuzzy.iau.dtu.dk/tedlib.nsf/htmlmedia/library.html, data de acesso: 07/12/2004, Agosto, 1998.

Jantzen, J. (1998) **Tutorial on Fuzzy Logic**, *Curso on-line de Fuzzy*, http://fuzzy.iau.dtu.dk/tedlib.nsf/htmlmedia/library.html, data de acesso: 07/12/2004, Agosto, 1998.

Jin, Y. (2004) **A Definition of Soft Computing**, http://www.soft-computing.de/, data de acesso: 07/04/2004.

Jung, D.; Zelinski, A. (2000) **Grounded Symbolic Communication between Heterogeneous Cooperating Robots**, *Autonomous Robots*, V. 8, pp. 269-292.

Karr, C. L.; Zeiler, T. A.; Mehrotra, R. (2004) **Determining Worst-Case Gust Loads on Aircraft Structures Using an Evolutionary Algorithm**, *Applied Intelligence*, V. 20, pp. 135-145.

Kennedy, J.; Eberhart, R C. (2001) **Swarm Intelligence,** San Francisco, Estados Unidos: Morgan Kaufmann Publishers.

Kennedy, J.; Eberhart, R. C. (1995) **Particle Swarm Optimization**, *IEEE International Conference on Neural Networks*, Perth, Austrália, V. 4, pp. 1942-1948.

Khoury, G. M.; Saad, M.; Kanaan, H. Y.; Asmar, C. (2004) **Fuzzy PID Control of a Five DOF Robot Arm**, *Journal of Intelligent and Robotic Systems*, V. 40, pp. 299-320.

Kodagoda, K. R. S.; Wijesoma, W. S.; Teoh, E. K. (2002) **Fuzzy Speed and Steering Control of an AGV**, *IEEE Transactions on Control Systems Technology*, V. 10, no. 1, pp. 112-120.

Köse, H. (2000) **Towards a Robust Cognitive Architecture for Autonomous Mobile Robots**, *Master of Science in Computer Engineering – Bogaziçi University*, Turquia.

Kristensen, L. K. (2000) **Aintz: Collective problem solving by artificial ants**, *Master Thesis in Computer Science – Arhus University, Computer Science Departament*, Dinamarca.

Krohling, R. A.; Hoffman, F.; Coelho, L. S. (2004) **Co-evolutionary Particle Swarm Optimization for min-max Problems Using Gaussian Distribution**, *International Congress on Evolutionary Computation – CEC 2004*, Piscataway, Estados Unidos, pp. 959-964.

K-TEAM website, http://www.k-team.com, data de acesso: 14/03/2005.

Kube, C. R.; Bonabeau, E. (2000) **Cooperative Transport by Ants and Robots**, *Robotics and Autonomous Systems*, V. 30, pp. 85-101.

Kulitz, H. R.; Ferreira, E. P. (2004) **Fuzzy Inference Propagation Applied to the Control of Robots and/or Multi-Linked Vehicles**, *VI Conferência Internacional de Aplicações Industriais – VI Induscon*, Joinville, SC, pp. 1432-1437.

Lee, S. H.; Howlett, R. J.; Walters, S. D. (2003) **A Fuzzy Control System for a Small Gasoline Engine**, *7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems – KES 2003*, Oxford, Inglaterra, pp. 722-732.

Lehmann, F.; Ritzschke, M.; Meffert, B. (1999) **ExmoR – A Testing Tool for Control Algorithms on Mobile Robots**, *1st International Khepera Workshop*, Paderborn, Alemanha.

Lenas, P.; Kitade, T.; Watanabe, H.; Honda, H. (1997) **Adaptative Fuzzy Control of Nutrients Concentration in Fed-batch Culture of Mammalian Cells**, *Cytotechnology*, V. 25, pp. 9-15.

Li, X. (2004) **Better Spread and Convergence: Particle Swarm Multiobjective Optimization Using the Maximin Fitness Function**, *Genetic and Evolutionary Computation Conference – GECCO 2004*, Seattle, Estados Unidos, pp. 117-128.

Liu, Y.; Passino, K. M. (2000) **Swarm Intelligence: Literature Overview**, *Dept. of Electrical Engineering – The Ohio State University*, Estados Unidos.

Liu, Y.; Passino, K. M.; Polycarpou, M. (2001) **Stability Analysis of M-dimensional Asynchronous Swarms with a Fixed Communication Topology**, *IEEE Transactions on Automatic Control*, V. 48, no. 1, pp. 76-95.

Liu, Y.; Passino, K. M. (2002) **Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles, and Emergent Behaviours**, *Journal of Optimization Theory and Applications*, V. 115, no. 3, pp. 603-628.

Liu, Y; Zheng, Q.; Shi, Z.; Chen, J. (2004) **Training Radial Basis Function Networks with Particle Swarms**, *IEEE International Symposium on Neural Networks – ISNN 2004*, Dalian, China, pp. 317-322.

Louarn, F. X.; Gendreau, M.; Potvin, J. Y. (2004) **GENI Ants for the Travelling Salesman Problem**, *Annals of Operations Research*, V. 131, pp. 187-201.

Lu, W. Z.; Fan, H. Y.; Leung, A. Y. T.; Wong, J. C. K. (2002) **Analysis of Pollutant Levels in Central Hong Kong Applying Neural Network Method with Particle Swarm Optimization**, *Environmental Monitoring and Assessment*, V. 79, pp. 217-230.

Lundgren, M. (2003) **Path Tracking and Obstacle Avoidance for a Miniature Robot**, *Master Thesis – Department of Computer Science – Umea University*, Suécia.

Mamdani, E.; Assilian, S. (1975) **An Experimental in Linguistic Synthesis with a Fuzzy Logic Controller**, *International Journal of Man Machine Studies*, V. 7, no. 1, pp.1-13.

Man, K. F.; Tang, K. S.; Kwong, S. (1996) **Genetic Algorithms: Concepts and Applications**, *IEEE Transactions on Industrial Electronics*, V. 43, no. 5, pp. 519-534.

Martinet, P.; Thibaud, C. (2000) **Automated Guided Vehicles: Robust Controller Design in Image Space**, *Autonomous Robots*, V. 8, pp. 25-42.

Maruyama, N.; Furukawa, C. M.; Souza, E. C. (2004) **Development of an Underwater Robotic System for Deep Oil Water Fields**, *VI Conferência Internacional de Aplicações Industriais – VI Induscon*, Joinville, SC, pp. 1438-1443.

Mavridou, T. D.; Pardalos, P. M. (1997) **Simulated Annealing and Genetic Algorithms for the Facility Layout Problem: A Survey**, *Computational Optimization and Applications*, V. 7, pp. 111-126.

Messom, C. (2002) **Genetic Algorithms for Auto-Tuning Mobile Robot Motion Control**, *Research Letters in the Information and Mathematical Science*, V. 3, pp. 129-134.

Meyer, J. A. (1998) **Evolutionary Approaches to Neural Control in Mobile Robots**, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, Estados Unidos, V. 3, pp. 2418-2423.

Miller, J. F.; Job, D.; Vassilev, V. K. (2000) **Principles in the Evolutionary Design of Digital Circuits – Part II**, *Genetic Programming and Evolvable Machines*, V. 1, pp. 259-288.

Minchev, Z.; Manolov, O.; Noykov, S.; Witkowski, U.; Rückert, U. (2004) **Fuzzy Logic Based Intelligent Motion Control of Robot Swarm Simulated by Khepera Robots**, *Second IEEE International Conference on Intelligent Systems*, Melbourne, Austrália, pp. 305-310.

Mitchell, M. (1996) **An Introduction to Genetic Algorithms**, Cambridge: MIT Press, pp. 209.

Miwa, M.; Matsuzaki, M.; Okuma, S.; Inoue, T.; Furuhashi, T. (2002) **Nurse Scheduling System using Bacterial Evolutionary Algorithm Hardware**, *Proceedings of the 28th Annual Conference of the Industrial Electronics Society 2002 (IECON 02)*, Seville, Espanha, V. 3, pp. 1801-1805.

Mondana, F.; Pettinaro, G. C.; Guignard, A.; Kwee, I. W.; Floreano, D.; Deneubourg, J-L.; Nolfi, S.; Gambardella, L. M.; Dorigo, M. (2004) **Swarm-Bot: A New Distributed Robotics Concept**, *Autonomous Robots*, V. 17, pp. 193-221.

Monmarché, M.; Venturini G.; Slimane, M. (1999) **On How the Ants Pachycondyla apicalis Are Sugesting a New Search Algorithm**, *Laboratoire d'Informatique – Université de Tours*, França, Internal Report 214, E3i, 17p.

Moudgal, V. G.; Kwong, W. A. (1995) **Fuzzy Learning Control for a Flexible-Link Robot**, *IEEE Transactions on Fuzzy Systems*, V. 3, no. 2, pp. 199-210.

Müller, S. D.; Marchetto, J.; Airaghi, S.; Koumoutsakos, P. (2002) **Optimization Based on Bacterial Chemotaxis**, *IEEE Transactions on Evolutionary Computation*, V. 6, no. 1, pp. 16-29.

Muscato, G. (2000) **Introduction to the Special Issue on Soft-Computing Techniques for the Control of Walking Robots**, *Soft Computing*, V. 4, pp. 193-194.

Nadimi, S.; Bhanu, B. (2004) **Cooperative Coevolution Fusion for Moving Object Detection**, *Genetic and Evolutionary Computation Conference – GECCO 2004*, Seattle, Estados Unidos, pp. 587-589.

Nanayakkara, D. P. T.; Watanabe, K.; Kiguchi, K.; Izumi, K. (2001) **Evolutionary Learning of a Fuzzy Behavior Based Controller for a Nonholonomic Mobile Robot in a Class of Dynamic Environments**, *Journal of Intelligent and Robotic Systems*, V. 32, pp. 255-277.

Nawa, N. E.; Furuhashi, T. (1997) **A Study on Nonlinear Model Identification Using Pseudo-Bacterial Genetic Algorithm**, *Proceedings of the 4th International Conference on Neural Information Processing (ICONIP'97)*, Dunedin, Nova Zelândia.

Nawa, N. E.; Furuhashi, T. (1998) **Bacterial Evolutionary Algorithm for Fuzzy System Design**, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, Estados Unidos, V. 3, pp. 2424-2429.

Nawa, N. E.; Furuhashi, T. (1999) **Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm**, *IEEE Transactions on Fuzzy Systems*, V. 7, no. 5, pp. 608-616.

Nolfi, S. (2000) **Evolutionary Robotics: The Biology Intelligence, and Technology of Self-Organizing Machines**, *The MIT Press*, Massachusetts, Estados Unidos.

Novkovic, S. (1998) **A Genetic Algorithm Simulation of a Transition Economy: An Application to Insider-Privatization in Croatia**, *Computational Economics*, V. 11, pp. 211-243.

Ogata, K. (2003) **Engenharia de Controle Moderno**, 4 ed., Rio de Janeiro:Prentice Hall, pp. 708.

Onwubolu, G. O.; Mutingi, M. (2003) **A Genetic Algorithm Approach for the Cutting Stock Problem**, *Journal of Intelligent Manufacturing*, V. 14, pp. 209-218.

Orhan, A.; Akin, E.; Karaköse, M. (2001) **DSP Implementation of Fuzzy Controlled Magnetic Levitation System**, *Proceedings of the International Conference, 7th Fuzzy Days on Computational Intelligence, Theory and Applications,* Dortmund, Alemanha, pp. 950-958.

Ostergaard, E. H.; Lund, H. H. (2003) **Co-Evolving Complex Robot Behavior**, *ICES 2003*, Cidade, País, pp. 308-319.

Papageorgiou, E. I.; Spyridonos, P. P.; Stylios, C. D.; Ravazoula, P.; Nikiforidis, G. C.; Groumpos, P. P. (2004) **The Challenge of Soft Computing Techniques for Tumor Characterization**, *International Conference on Artificial Intelligence and Soft Computing – ICAISC 2004*, Zakopane, Polônia, pp. 1031-1036.

Parsopoulos, K. E.; Vrahatis, M. N. (2002) **Recent Approaches to Global Optimization Problems through Particle Swarm Optimization**, *Natural Computing*, V. 1, pp. 235-306.

Parsopoulos, K. E.; Papageorgiou, E. I.; Groumpos, P. P.; Vrahatis, M. N. (2004) **Evolutionary Computation Techniques for Optimizing Fuzzy Cognitive Maps in Radiation Therapy Systems**, *Genetic and Evolutionary Computation Conference – GECCO 2004*, Seattle, Estados Unidos, pp. 402-413.

Passino, K. M. (2002) **Biomimicry of Bacterial Foraging for Distributed Optimization and Control**, *IEEE Control Systems Magazine*, V. 22, no. 3, pp. 52-67.

Pereira, G. A. S.; Campos, M. F. M.; Kumar, V. (2001) **Distributed Sensing for Cooperative Robotics**, *Anais da V Semana de Pós-Graduação em Ciência da Computação – SPG'2001*, Belo Horizonte, MG, pp. 1-5.

Pires, G.; Nunes, U. (2002) **A Wheelchair Steered through Voice Commands and Assisted by a Reactive Fuzzy-Logic Controller**, *Journal of Intelligent and Robotic System,* V. 34, pp. 301-314.

Pires, E. J. S.; Oliveira, P. B. M.; Machado, J. A. T. (2004) **Multi-Objective Genetic Manipulator Trajectory Planer**, *EvoWorkshops 2004*, Coimbra, Portugal, pp. 219-229.

Pires, E. J. S.; Machado, J. A. T.; Oliveira, P. B. M. (2004) **Robot Trajectory Planning Using Multi-Objective Genetic Algorithm Optimization**, *Genetic and Evolutionary Computation Conference – GECCO 2004*, Seattle, Estados Unidos, pp. 615-626.

Pirolli, P.; Card, S. K. (1999) **Information Foraging**, *UIR Technical Report*, pp. 84.

Podgorelec, V.; Kokol, P. (2001) **Towards More Optimal Medical Diagnosing with Evolutionary Algorithms**, *Journal of Medical Systems*, V. 25, no. 3, pp. 195-219.

Podlena, J. R.; Hendtlass, T. (1998) **An Accelerated Genetic Algorithm**, *Applied Intelligence*, V. 8, pp. 103-111.

Prabhaharan, G.; Asokan, P.; Rajendran, S. (2004) **Sensitivity-Based Conceptual Design and Tolerance Allocation Using the Continuous Ant Colony Algorithm (CACO)**, *International Journal of Advanced Manufacturing Technology* (in press).

Preuss, O. P. M. (2004) **Cooperating Robots: Developments in Multi-Controller Program Cooperation**, *VI Conferência Internacional de Aplicações Industriais – VI Induscon*, Joinville, SC, pp. 1371-1376.

Pugh, J.; Martinoli, A.; Zhang, Y. (2005) **Particle Swarm Optimization for Unsupervised Robotic Learning**, *IEEE Swarm Intelligence Symposium*, Pasadena, Estados Unidos, pp. 92-99.

Qin, Y-Q.; Sun, D-B.; Li, N.; Cen, Y-G. (2004) **Path Planning for Mobile Robot Using the Particle Swarm Optimization with Mutation Operator**, *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, Shangai, China, pp. 2473-2478.

Randall, M. J.; Pipe, A. G. (2000) **A Novel Soft Computing Architecture for the Control of Autonomous Walking Robots**, *Soft Computing*, V. 4, 165-185.

Rekleitis, I.; Dudek, G.; Milios, E. (2001) **Multi-Robot Collaboration for Robust Exploration**, *Annals of Mathematics and Artificial Intelligence*, V. 31, pp. 7-40.

Rivera, W. (2001) **Scalable Parallel Genetic Algorithms**, *Artificial Intelligence Review*, V. 16, pp. 153-168.

Salomon, R. (2005) **Evolving and Optimizing Braitenberg Vehicles by Means of Evolution Strategies**, *International Journal of Smart Engineering Systems Design*, in press.

Schenker, P. S.; Huntsberger, T. L.; Pirjanian, P.; Baumgartner, E. T.; Tunsel, E. (2003) **Planetary Rover Developments Supporting Mars Exploration, Sample Return and Future Human-Robotic Colonization**, *Autonomous Robots*, V. 14, pp. 103-126.

Sedighi, K. H.; Ashenayi, K.; Manikas, T. W.; Wainwright, R. L.; Tai, H. M. (2004) **Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm**, *Proceedings of the IEEE Congress on Evolutionary Computation 2004 – CEC 2004*, San Diego, Estados Unidos, pp. 1338-1344.

Seng, T. L.; Khalid, M.; Yusof, R.; Omatu, S. (1998) **Adaptative Neuro-fuzzy Control Systems by RBF and GRNN Neural Networks**, *Journal of Intelligent and Robotic System*, V. 23, pp. 267-289.

Sgouros, N. M. (2002) **Qualitative Navigation for Autonomous Wheelchair Robots in Indoor Environments**, *Autonomous Robots*, V. 12, pp. 257-266.

Shi, Y.; Eberhart, R. C. (2002) **Fuzzy Adaptive Particle Swarm Optimization**, *Proceedings of the Congress on Evolutionary Computation – CEC 2002*, Honolulu, Estados Unidos, V. 1, pp. 101-106.

Shi, Y.; Eberhart, R. C. (1998) **Parameter Selection in Particle Swarm Optimizer**, *Proceedings Seventh Annual Conference on Evolutionary Programming*, San Diego, Estados Unidos, pp. 591-601.

Sierakowski, C. A.; Araújo, H. X.; Oliveira, G. H. C.; Mendes, N.; Coelho, L. S.; Mendonça, K. C.; Santos, G. H. (2004) **ASTECCA: Programa para Análise Térmica e de Estratégias de Controle para Condicionamento de Ambientes**, *IV Feira e Congresso de Ar Condicionado, Refrigeração e Ventilação do Mercosul – Mercofrio*, Curitiba, PR.

Sierakowski, C. A.; Coelho, L. S. (2004) **Otimização do Planejamento de Trajetórias de Robôs Móveis Baseada em Colônia de Bactérias**, *SBRN – Brazilian Symposium on Neural Networks*, São Luís, MA.

Sierakowski, C. A.; Coelho, L. S. (2004) **Particle Swarm Optimization Applied to Path Planning of a Mobile Robot: New Approaches and Comparative Study**, *VI Conferência Internacional de Aplicações Industriais – VI Induscon*, Joinville, SC, pp. 1-6.

Sierakowski, C. A.; Costa, A. C. P. L.; Coelho, L. S. (2004) **Uma Comparação de Algoritmos Genéticos, Programação Evolucionária e Evolução Diferencial para Otimização de Trajetórias de Robôs Móveis**, *II ENRI – Encontro de Robótica Inteligente*, Salvador, BA.

Sierakowski, C. A.; Coelho, L. S. (2005) **Ant Colony Search Model Based on Continuous Optimization for Path Planning in Mobile Robotics**, *World Conference on Soft Computing in Industrial Applications - WSC10*, http://www.cranfield.ac.uk/wsc10/.

Sierakowski, C. A.; Coelho, L. S. (2005) **Study of Two Swarm Intelligence Techniques for Path Planning of Mobile Robots**, *IFAC World Congress 2005*, Prague, República Checa.

Sierakowski, C. A.; Guerra, F. A.; Coelho, L. S. (2005) **Particle Swarm Optimization Approach for Multi-Step-Ahead Prediction Using Radial Basis Function Neural Network**, *IFAC World Congress 2005*, Prague, República Checa.

Silva, L. N. C. (2001) **Engenharia Imunológica: Desenvolvimento e Aplicação de Ferramentas Computacionais Inspiradas em Sistemas Imunológicos Artificiais**, *Tese de Doutorado – DCA/UNICAMP*, Campinas, SP.

Silva, L. N. C.; Timmis, J. (2002) **An Artificial Immune Network for Multimodal Function Optimization**, *Proceedings of the IEEE Congress on Evolutionary Computation*, V. 1, pp. 669-674.

Silva, L. N. C. (2002) **Immune, Swarm, and Evolutionary Algorithms Part I: Basic Models**, *Proceedings of the ICONIP Workshop on Artificial Immune Systems*, V. 3, pp. 1464-1468.

Silva, L. R. (2003) **Análise e Programação de Robôs Móveis Autônomos da Plataforma Eyebot**, *Dissertação de Mestrado em Engenharia Elétrica – Universidade Federal de Santa Catarina*, Florianópolis, SC.

Silva, L. N. C.; Zuben, F. J. V. (2004) **Sistemas Imunológicos Artificiais**, *Ciência Hoje*, V. 35, no. 205, pp. 20-25.

Spaulding, K. A. (1998) **Natural Metaphoric Optimization Algorithms**, *Master of Science in Engineering – The University of Texas at Austin*, Austin, Estados Unidos.

Stützle, T.; Hoos, H. H. (2000) **MAX-MIN Ant System**, *Future generation Computer Systems*, V. 16, pp. 889-914.

Syarif, A.; Gen, M. (2003) **Solving Exclusionary Side Constrained Transportation Problem by Using a Hybrid Spanning Tree-based Genetic Algorithm**, *Journal of Intelligent Manufacturing*, V. 14, pp. 389-399.

Takagi, T.; Sugeno, M. (1985) **Fuzzy Identification of Systems and its Applications to Modeling and Control**, *IEEE Transactions on Systems, Man and Cybernetics*, V. 15, pp. 116-132.

Tan, K. C.; Lee, T. H.; Khor, E. F. (2002) **Evolutionary Algorithms for Multi-Objective Optimization: Performance Assessments and Comparisons**, *Artificial Intelligence Review*, V. 17, pp. 253-290.

Thongchai, S.; Suksakulchai, S.; Wilkes, D. M.; Sarkar, N. (2000) **Sonar Behavior-Based Fuzzy Control for a Mobile Robot**, *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Nashville, Estados Unidos, pp. 1-6.

Thomaz, C. E.; Pacheco, M. A. C.; Vellasco, M. M. B. R. (1999) **Mobile Robot Path Planning Using Genetic Algorithms**, *International Conference on Evolutionary Computation in Engineering – ICECE'99*, Rio de Janeiro, RJ.

Tian, Y.; Song, J.; Yao, D.; Hu, J. (2003) **Dynamic Vehicle Routing Problem Using Hybrid Ant System**, *Intelligent Transportation Systems*, V. 2, pp. 970-974.

Trianni, V.; Grob, R.; Labella, T. H.; Sahin, E.; Dorigo, M. (2003) **Evolving Aggregation Behaviors in a Swarm of Robots**, *7th European Conference on Artificial Life – ECAL 2003*, Dortmund, Alemanha, pp. 865-874.

Tunstel, E.; Oliveira, M. A. A.; Bergman, S. (2002) **Fuzzy Behavior Hierarchies for Multi-Robot Control**, *International Journal of Intelligent Systems*, V. 17, pp. 449-470.

Vellasco, M. M. B. R.; Pacheco, M. A. C.; Brasil, I. L. (1999) **Mobile Robot Control Using Fuzzy Logic**, *1st International Khepera Workshop*, Paderborn, Alemanha.

Xiao, J. Michalewicz, Z.; Zhang, L.; Trojanowski, K. (1997) **Adaptative Evolutionary Planner/Navigator for Mobile Robots**, *IEEE Transactions on Evolutionary Computation*, V. 1, no. 1, pp. 18-28.

Xie, X.; Zhang, W.; Yang, Z. (2002) **A Dissipative Particle Swarm Optimization**, *Proceedings of the Congress on Evolutionary Computation – CEC 2002*, Honolulu, Estados Unidos, V. 2, pp. 1456-1461.

Yager, R. R.; Filev, D. P. (1994) **Essentials of Fuzzy Modeling and Control**, New York – Estados Unidos: *Wiley-Interscience*.

Yan, T.; Ota, J.; Nakamura, A.; Arai, T. Kuwahara, N. (2002) **Development of a Remote Fault Diagnosis System Applicable to Autonomous Mobile Robots**, *Advanced Robotics*, V. 16, no. 7, pp. 573-594.

Yan, Z.; Yuan, C. (2004) **Ant Colony Optimization for Feature Selection in Face Recognition**, *International Conference on Biometric Authentication – ICBA 2004*, Hong Kong, China, pp. 221-226.

Yang, S.; Wang, M.; Jiao, L. (2004) **A Quantum Particle Swarm Optimization**, *Proceedings of the Congress on Evolutionary Computation – CEC 2004*, Portland, Estados Unidos, pp. 320-324.

Yang, S. X.; Wang, X.; Wang, G.; Meng, M. Q.-H. (2005) **An Embedded Genetic Fuzzy Motion Controller for a Mobile Robot**, *IFAC World Congress 2005*, Prague, República Checa.

Yang; S. X.; Hu, Y. (2005) **Robot Path Planning in Unstructured Environments Using a Knowledge-Based Genetic Algorithm**, *IFAC World Congress 2005*, Prague, República Checa.

Yao, F.; Shao, G.; Takaue, R.; Tamaki, A. (2003) **Automatic Concrete Tunnel Inspection Robot System**, *Advanced Robotics*, V. 17, no. 4, pp. 319-337.

Yen, J. (1999) **Fuzzy Logic – A Modern Perspective**, *IEEE Transactions on Knowledge and Data Engineering*, V. 11, no. 1, pp. 153-165.

Yen, J.; Langari, R. (1999) **Fuzzy Logic: Intelligence, Control and Information**, New Jersey: Prentice-Hall.

Yun, Y.; Gen, M. (2003) **Performance Analysis of Adaptive Genetic Algorithms with Fuzzy Logic and Heuristics**, *Fuzzy Optimization and Decision Making*, V. 2, pp. 161-175.

Yun, Y.; Gen, M.; Seo, S. (2003) **Various hybrid methods based on genetic algorithm with fuzzy logic controller**, *Journal of Intelligence Manufacturing*, V. 14, pp. 401-419.

Wang, P. K. C. (2003) **Optimal Path Planning Based on Visibility**, *Journal of Optimization Theory and Applications*, V. 117, no. 1, pp. 157-181.

Watanabe, K.; Hashem, M. M. A.; Izumi, K. (1999) **Global Path Planning of Mobile Robots as an Evolutionary Control Problem**, *Proceedings of European Control Conference*, Karlsruhe, Alemanha.

White, J. A.; Garrett, S. M. (2003) **Improved Pattern Recognition with Artificial Clonal Selection?**, *2nd International Conference on Artificial Immune Systems – ICARIS 2003*, Edinburgh, Inglaterra, pp. 181-193.

White, T. (2004) **Swarm Intelligence: a Gentle Introduction with Applications**, http://www.sce.carleton.ca/netmanage/tony/swarm-presentation/sld003.htm, data de acesso: 15/04/2004.

Wu, C-J.; Liu, G-Y. (2000) **A Genetic Approach for Simultaneous Design of Membership Functions and Fuzzy Control Rules**, *Journal of Intelligent and Robotic Systems*, V. 28, pp. 195-211.

Wu, W.; QiSen, Z.; Mbede, J. B.; Xinhan, H. (2001) **Research on Path Planning for Mobile Robots Among Dynamic Obstacles**, *IFSA World Conference and 20th NAFIPS International Conference*, Vancouver, Canadá, V. 2, pp. 763-767.

Zadeh, L. A. (1965) **Fuzzy Sets**, *Information and Control*, V. 8, pp. 338-353.

Zadeh, L. A. (1994) **Fuzzy logic and soft computing: issues, contentions and perspectives**, *3rd International Conference Fuzzy Logic, Neural Nets and Soft Computing*, Iizuka, Japão, pp. 1-2.