UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Otimização de controladores nebulosos e sistemas especialistas reativos utilizando algoritmos genéticos

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

Eder Mateus Nunes Gonçalves

Florianópolis, 23 de fevereiro de 2001.

Otimização de controladores nebulosos e sistemas especialistas reativos utilizando algoritmos genéticos

Eder Mateus Nunes Gonçalves

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Guilherme Bittencourt
Orientador

P/ Aguinaldo Silveira e Silva
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Guilherme Bittencourt, Dr.
Presidente

Marcelo Ricardo Stemmer, Dr.

Christoph Thomalla, Dr.

A meu pai e minha mãe...

a meus irmãos: Thiago, Maurício e Sam...

AGRADECIMENTOS

Dentre as várias contribuições de um trabalho deste nível na vida de uma pessoa, o fato de chegar ao fim e ter a quem agradecer significa que a obra construída teve uma abrangência muito maior que o cunho científico ao qual ela se propôs. Significa que ela contribuiu, sobretudo, com o incremento de relações e amizades de seu autor. E isto, é sem dúvida, a maior das contribuições.

Para tanto, gostaria de agradecer inicialmente, o Prof. Guilherme Bittencourt pela oportunidade dada a mim, e principalmente, por ter acreditado, naquilo que ele me fez crer ser meu potencial.

Não posso, de forma alguma, esquecer de agradecer a Augusto Loureiro da Costa, pela atenção e paciência dispendida neste ano de uma verdadeira co-orientação ao qual ele me prestou, além da oportunidade de fazer parte do fantástico projeto UFSC-Team, o qual me abracei e pretendo seguir me dedicando.

Um trabalho de Dissertação, obviamente, não é alcançado sem o devido aporte emocional. Para tanto devo agradecer a minha família e àqueles que considero como se fossem dela. À meu pai, Ires Gonçalves, à minha mãe, Cleni Gonçalves, à meu irmão, Thiago Gonçalves, e àqueles que podem não ter meu mesmo sangue, mas que considero também como irmãos, Sam Devincenzi e Maurício Castro. Não posso esquecer também de agradecer a minha companheira, Suzane Vieira, que provavelmente foi a maior vítima das minhas constantes variações de humor neste período. Agradecimento especial também aos amigos, Adriana Pereira, Letícia Pereira, André Ianzer, Lívia D'Ávila, Mateus Brancão.

A lista acima deve ser também complementada com aqueles que juntaram-se a mim durante a jornada dentro do LCMI/UFSC: Carlos Brandão, Carlos Montez, Sérgio Almeida, Jerusa Marchi, Fred Paes, Carlos Ogawa, Frank Siqueira, Frederico de Freitas, Luciano Rottava da Silva, Rafael Obelheiro, Renato de Oliveira, Karina Barbosa, Michele Wanghan, Marcos Luiz de Assis, Wilson Costa e os bolsistas Alexandre Lorencato e Tiago Vilaça.

Agradecimento especial também dedico a Capes que forneceu o devido suporte financeiro através de minha bolsa.

Com certeza muitos outros deveriam fazer parte desta seção, e infelizmente o espaço é reduzido, mas mesmo assim, queria agradecer, do fundo do coração, àqueles que de alguma forma sentiram-se responsáveis por meu sucesso neste trabalho.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Otimização de controladores nebulosos e sistemas especialistas reativos utilizando algoritmos genéticos

Eder Mateus Nunes Gonçalves

Fevereiro/2001

Orientador: Guilherme Bittencourt
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Inteligência Artificial, Algoritmos Genéticos, Controladores Nebulosos
Número de Páginas: 91

A RoboCup, "Robot World Cup" , é uma iniciativa de um grupo internacional de pesquisadores em Inteligência Artificial e Robótica que propõe um problema padrão a ser solucionado: uma partida de futebol de robôs. Esta iniciativa permite que diversas técnicas destas áreas sejam testadas e, principalmente, comparadas em termos de eficiência. Atualmente encontra-se em andamento a implementação de um time de futebol de robôs para a categoria simuladores da RoboCup - *UFSC-Team* - na qual cada sistema que implementa um dos jogadores do time é baseado em um modelo de agente autônomo concorrente. De acordo com este modelo cada um dos agentes apresenta um sistema decisório dividido em três níveis: *reativo, instintivo e cognitivo.* Estes níveis decisórios são implementados em três diferentes processos: *Interface, Coordinator* e *Expert* de acordo com a arquitetura do ambiente Expert-Coop. Este trabalho traz a implementação de uma ferramenta para a otimização de parâmetros de controladores nebulosos e de sistemas especialistas reativos utilizando algoritmos genéticos. Esta ferramenta permite a utilização de algoritmos genéticos para determinar os valores ótimos para os parâmetros dos controladores nebulosos e dos sistemas especialistas reativos, responsáveis respectivamente pelos *comportamentos reativos*, e pelos *comportamentos instintivos* dos agentes do UFSC-Team. Sendo assim, os controladores podem ser implementados intuitivamente e em seguida submetidos a um processo evolutivo utilizando algoritmos genéticos. Isso representa um ganho significativo, tanto no tempo de desenvolvimento quanto na eficiência, do processo de implementação destes controladores e sistemas especialistas.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

Controlers fuzzy and reactive expert systems optimization using genetic algorithms

Eder Mateus Nunes Gonçalves

February/2001

Advisor: Guilherme Bittencourt
Area of Concentration: Control, Automation and Industrial Computing
Keywords: Artificial Intelligence, Genetic Algorithms, Fuzzy Control
Number of Pages: 91

Robocup (The Robot World Cup) is an initiative of an international group of researchers in Artificial Intelligence and Robotic that proposes a standard problem to be solved: a robot soccer match. This initiative allows the different techniques used in the solution of the standard problem to be tested and, mainly, to be compared in term of efficiency. Presently, a robot soccer team for the RoboCup simulation league - *UFSC-Team* - is being implemented at UFSC. In this team, the systems that implement the team players are based on a concurrent autonomous agent model. According to this model, each agent presents a three layer decision system: *reactive*, *instintive* and *cognitive*. This decision layers are implemented by three different processes: *Interface*, *Coordinator* and *Expert*, as proposed in the architecture of the Expert-Coop enviroment. This work presents the development of a tool for fuzzy control and reactive expert system parameters' otimization using genetic algorithms. This tool allows the use of genetic algorithms to determine the optimal values to the fuzzy control and to the expert system parameters, responsible, respectively, by the *reactive behaviour* and by the *instintive behaviour* of the UFSC-Team agents. Therefore, the controlers can be implemented intuitively and than optimized through an evolutive process using a genetic algorithm. This represents a significant gain in the development time and efficiency, with respect to the implementation process of these controllers and expert systems.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A *Inteligência Artificial* (IA) como um ramo da ciência da computação é uma área de pesquisa bastante recente. Oficialmente, a IA nasceu em uma conferência de verão em Dartmouth College, NH, USA. No entanto, sob uma análise mais profunda sobre o tema, logo conclui-se que a origem da IA está muito mais distante do que isso. Os conceitos chaves que deram uma primeira sustentação as teorias desenvolvidas datam de no mínimo 23 séculos. A filosofia, a lógica, a matemática emprestam alguns dos seus conceitos a IA, dando a esta uma abrangência muito maior, ainda mais se incorporarmos os avanços científicos que possibilitaram a implementação dos sistemas desenvolvidos.

Desde o início a IA já gerou polêmica. A primeira, dizia respeito ao seu nome. Segundo alguns, era presunçoso demais. Outra dúvida era com relação a seus objetivos e metodologias.

O objetivo central da IA é simultaneamente teórico - a criação de teorias e modelos para a capacidade cognitiva - e prático - a implementação de sistemas computacionais baseados nestes modelos. Nesse sentido, a IA tem uma relação com seu objeto de estudo semelhante à da psicologia, mas com uma importante diferença: os modelos e teorias da IA são *implementados* em um computador, o que os torna de certa forma autônomos. Assim, a validade de um modelo ou de uma teoria de IA não precisa ser provada através de comparação de seus resultados previstos com o comportamento psíquico humano, como no caso da psicologia, mas pode ser implementada em um computador e demonstrada diretamente através da *ação inteligente* do programa no mundo.

Uma forma encontrada de impulsionar a pesquisa em IA é a proposta de problemas padrão, quando normalmente grandes avanços em áreas de pesquisas correlatas ocorrem. O exemplo, talvez mais conhecido, são os programas de computador desenvolvidos para jogar xadrez. Através deles, poderosos algoritmos de busca foram desenvolvidos, resultando no super computador Deep Blue, que em 1997 derrotou o campeão mundial de xadrez, Gary Kasparov.

Como o objetivo de vencer o campeão mundial humano de xadrez foi alcançado, surgiu espaço para um novo desafio. Foi quando um grupo de pesquisadores uniu-se propondo o problema de uma partida de futebol entre robôs. A idéia tomou forma e em 1997 realizou-se a primeira RoboCup, em Nagoya, Japão. De lá para cá competições anuais são realizadas sempre em locais distintos. A meta final se compara com os programas para xadrez: em 2050 ter uma equipe de robôs humanóides em condições de vencer a equipe campeã mundial de futebol. Como a própria criação da IA, é uma proposta bastante presunçosa. Um aspecto interessante da proposta da RoboCup é a possibilidade de utilização e comparação de várias tecnologias e áreas de pesquisa distintas. Em particular, diferentes técnicas de IA, como sistemas especialistas, algoritmos genéticos e lógica nebulosa.

Os sistemas especialistas são programas de computador concebidos para reproduzir o comportamento de especialistas humanos na resolução de problemas do mundo real, mas o domínio destes problemas é altamente restrito. Esta tecnologia tem sido aplicada com bastante sucesso a um grande número de domínios, incluindo química orgânica, exploração mineral, e medicina.

Os algoritmos genéticos (AG) são métodos adaptativos que podem ser usados para resolver problemas de busca e otimização, e fazem parte de uma família de modelos computacionais inspirados pela evolução natural. Através de muitas gerações, populações naturais evoluem de acordo com os princípios de seleção natural, que foram primeiramente propostos por Charles Darwin no seu livro *A Origem das espécies*. Analogamente ao processo natural, os AG's são capazes de "evoluir" soluções para problemas do mundo real, desde que estes sejam codificados convenientemente.

A lógica nebulosa, principalmente no que se refere às técnicas de controle nebuloso, conquistou espaço como área de estudo em diversas instituições de ensino, pesquisa e desenvolvimento do mundo, sendo até hoje uma importante aplicação da teoria de conjuntos nebulosos. Um controlador nebuloso é um sistema nebuloso a base de regras, composto de um conjunto de regras de produção do tipo **Se** ⟨ *premissa* ⟩ **Então** ⟨ conclusão ⟩, que definem ações de controle em função das diversas faixas de valores que as variáveis de estado do problema podem assumir.

A maior dificuldade na criação de sistemas nebulosos em geral, encontra-se na definição dos termos lingüísticos e das regras. Uma maneira de automatizar este processo é por meio de AG's, e uma implementação neste sentido é apresentada neste trabalho dentro do contexto de um agente construído para jogar futebol.

Atualmente encontra-se em andamento a implementação de um time de futebol de robôs para a categoria simuladores - *UFSC-Team* - na qual cada sistema que implementa um dos jogadores do time é baseado em um modelo de agente autônomo concorrente proposto em [7]. De acordo com este

modelo cada um dos agentes apresenta um sistema decisório dividido em três níveis: reativo, instintivo e cognitivo. Estes níveis decisórios são implementados em três processos diferentes: *Interface*, *Coordinator* e *Expert* de acordo com a arquitetura do ambiente Expert-Coop [15].

O nível reativo, implementado no processo Interface, consiste em um conjunto de controladores nebulosos, os quais são responsáveis pelos *comportamentos reativos* do jogador, por exemplo, *drible_oponente*, *conduza_bola*. O nível instintivo, implementado no processo Coordinator, é formado por um sistema especialista, com apenas um ciclo de inferência, responsável por identificar em que estado a partida se encontra e por selecionar o *comportamento reativo* mais adequado ao estado corrente. Esta seleção é determinada pelas metas locais, que definem um comportamento estratégico de mais longo prazo. Finalmente, o nível cognitivo, implementado no processo Expert, consiste em um sistema especialista simbólico, responsável pelo planejamento, pela cooperação com outros agentes e pela geração das metas globais e locais.

As contribuições desta dissertação podem ser divididas em três grupos: (i) contribuições ao projeto da arquitetura do agente do *UFSC-Team* e (ii) desenvolvimento de software e (iii) experimentos com algoritmos genéticos.

Em termos do projeto do agente do *UFSC-Team*, foram pesquisadas ferramentas adequadas para a implementação da arquitetura. Devido às características de tempo real do agente, não foi encontrado um arcabouço de sistemas especialistas adequado aos níveis instintivo e cognitivo que satisfizesse os requisitos levantados, sendo assim foi decidido que um arcabouço específico seria implementado. Devido à dificuldade de implementar manualmente os diversos controladores nebulosos necessários ao nível reativo, bem como à dificuldade de determinar os limiares que determinam às mudanças de estado no nível instintivo, foi decidido que algoritmos genéticos seriam usados para agilizar estas tarefas, o que permitirá a obtenção de um significativo ganho, tanto no tempo de desenvolvimento quanto na eficiência, do processo de implementação destes controladores e sistemas especialistas. Foi encontrada na literatura uma ferramenta de domínio público adequada para este fim [44]. Uma vez determinadas as ferramentas foi especificado o processo de desenvolvimento de software do agente.

Em termos de desenvolvimento de software, foi desenvolvido o arcabouço para sistemas especialistas que servirá de base para os níveis instintivo e cognitivo do agente do *UFSC-Team*. Foi ainda implementado um sistema que permite a utilização da biblioteca de algoritmos genéticos escolhida para determinar os valores ótimos para os parâmetros dos controladores nebulosos responsáveis pelos *comportamentos reativos* dos agentes do UFSC-Team e para os sistemas especialistas reativos responsáveis pelos *comportamentos instintivos* dos agentes. Este sistema difere das aplicações

tradicionais de algoritmos genéticos devido ao fato da função de avaliação não poder ser calculada diretamente a partir dos valores do cromossomo. Para avaliar um indivíduo da população é necessário implementar o fenótipo do cromossomo, isto é, obter, a partir dos valores do cromossomo, o código fonte do programa que implementa o agente, compilá-lo e, utilizando o simulador SoccerServer, executá-lo em uma situação controlada de jogo.

Finalmente, em termos de experimentos com algoritmos genéticos, foi validada a integração entre a biblioteca GAlib e o simulador SoccerServer através da otimização do controlador nebuloso *kick_to_gol*. A otimização de outros controladores nebulosos é bastante semelhante, bastando modificar o processo de avaliação da simulação do fenótipo em ação. Infelizmente, não foi possível testar a otimização dos parâmetros do sistema especialista do nível instintivo, pois esta otimização necessita a simulação de partidas completas, e o restante do agente ainda não está em condições de realizá-las.

Esta dissertação está assim dividida:

- o capítulo 2 descreve a Robocup, apresentando um histórico sobre o seu surgimento, a descrição do problema, as áreas de pesquisa envolvidas e as metas para os próximos anos de pesquisa;
- o capítulo 3 descreve o simulador Soccerserver que foi a base para o desenvolvimento deste trabalho. O capítulo apresenta o funcionamento do simulador, as regras julgadas durante a partida, o formato das informações que são manipuladas, o modo de implementação do agente especial *Coach* e questões referentes a temporização do Soccerserver;
- o capítulo 4 descreve os sistemas especialistas, sua arquitetura, além de questões relativas a aquisição de conhecimento e representação do conhecimento;
- o capítulo 5 apresenta os algoritmos genéticos, seus princípios básicos, explicações que estão por trás de seu funcionamento e temas relativos a sua aplicabilidade;
- o capítulo 6 apresenta primeiramente os conjuntos nebulosos, que são a base de desenvolvimento dos controladores nebulosos, que são apresentados logo a seguir;
- o capítulo 7 descreve o agente UFSC-Team, seus processos, a forma como foi implementado, bem como a apresentação de um exemplo de uma situação de inferência possível pelos componentes do agente;
- o capítulo 8 apresenta a implementação do trabalho, descrevendo um dos controladores nebulosos presentes no nível reativo, a forma como foi construído o AG, uma análise dos resultados obtidos e também uma descrição das extensões destes resultados dentro do agente UFSC-Team;

- finalmente o capítulo 9 traz as conclusões do trabalho e as perspectivas para trabalhos futuros.

Capítulo 2

A RoboCup

2.1 Introdução

Como anunciado no artigo que lançou a proposta [31], assim como no seu site na internet (ver: `www.robocup.org`), a RoboCup é um projeto de união internacional que visa a promoção do estudo em inteligência artificial, robótica e campos relacionados. É uma tentativa de promover pesquisa em inteligência artificial e em robótica inteligente a partir da proposta de um problema padrão onde um grande espectro de tecnologias podem ser integrados e examinados. O problema: uma partida de futebol entre robôs. A meta final da RoboCup é por volta do ano 2050 ter um time de futebol de robôs em condições de enfrentar e vencer a equipe campeã mundial.

Neste capítulo trataremos de assuntos que descrevem este excitante desafio, os objetivos, as metas para os próximos anos, as áreas de pesquisas abrangidas pelo problema, além de um breve histórico a respeito do seu surgimento.

2.2 A Robocup

A iniciativa *Robot World Cup* (RoboCup) é uma proposta de educação e pesquisa em nível internacional. Ela busca a promoção de pesquisa em inteligência artificial e em robótica inteligente a partir de um problema padrão onde uma grande variedade de tecnologias podem ser integradas e estudadas, bem como serem usadas para um projeto integrado de educação.

Para este propósito, a RoboCup usa uma partida de futebol como um domínio primário, e organiza a **RoboCup: The World Cup Soccer Games and Conferences**. Para uma equipe de robôs realmente jogar uma partida, várias tecnologias devem ser incorporadas. Não limitadas a estas, podemos citar:

- princípios de projeto de agentes autônomos;
- colaboração multiagente;
- aquisição estratégica de conhecimento;
- raciocínio em tempo real;
- robótica;
- comportamento reativo;
- reconhecimento de contexto;
- visão;
- controle de motores.

A RoboCup é uma tarefa para um time de robôs em um ambiente dinâmico. Ela também oferece uma plataforma computacional para pesquisa nos aspectos de software relacionados ao problema proposto, apresentada no próximo capítulo.

Enquanto projeto, a competição entre robôs não é a única atividade da proposta RoboCup. As atividades atuais do projeto envolvem:

- Conferências técnicas - em paralelo à competição anual, acontecem congressos internacionais de grande importância para a comunidade de inteligência artificial, como por exemplo IJCAI e ICMAS.
- Robot World Cup - competições anuais onde os pesquisadores podem testar seus protótipos. Dentre as já ocorridas temos a RoboCup-97 em Nagoya (Japão), RoboCup-98 em Paris (França), RoboCup-99 em Stockholm (Suécia) e a RoboCup-2000 em Melbourne (Austrália). Já estão programadas a RoboCup-2001 em Seatle (EUA) e a RoboCup-2002 novamente no Japão, mas agora em Fukuoka.
- Programas de desafio - a RoboCup também oferece um conjunto de desafios de curto e médio prazo a fim de promover um avanço nas pesquisas dos seus participantes. Embora saiba-se que o conjunto de soluções necessárias para a resolução da tarefa proposta provavelmente está a algumas décadas de pesquisa, é interessante traçar metas intermediárias de médio e curto prazo de cumprimento a fim de promover progressos nas atividades em desenvolvimento. A fim de

medir os progressos rumo à solução dos desafios, bem como a construção de novos, a RoboCup criou o **RoboCup Challenge Committe** na tentativa de organizar os desafios.

- Programas de educação - visam o estabelecimento de vários programas de educação usando a RoboCup. Exemplos de cursos, tanto de graduação como de pós-graduação incluem, mas não são limitados a: Inteligência Artificial usando a RoboCup, Introdução a Robótica usando a RoboCup, Programação em Inteligência Artificial usando a RoboCup, Sistemas Multiagentes, etc...

Visando atingir os mais variados tipos de arquiteturas de robôs desenvolvidos, a RoboCup dividiu a competição em algumas categorias, baseadas no tamanho dos robôs e em seus métodos de movimento. Estas são:

- Liga do simulador
- Liga de robôs pequenos com cinco robôs por equipe
- Liga de robôs pequenos com onze robôs por equipe
- Liga de robôs de tamanho médio
- Liga de robôs Sony com pernas
- Liga de humanóides (a partir de 2002)

Além das ligas citadas acima, ainda existe uma exibição de comentários artificiais gerados durante a execução de uma partida.

2.3 RoboCup como problema padrão

A proposta de uma partida de futebol como problema não surgiu por acaso. A necessidade era de uma idéia que representasse um grande apelo à comunidade científica e ao mesmo tempo que também se mostrasse como um extraordinário desafio. Uma partida de futebol entre robôs atende todos estes requisitos. É claro que as soluções finais, ou seja, a realização do objetivo, é uma meta de longo prazo. Entretanto, vários estágios intermediários se apresentarão durante este caminho, e as soluções para estes subproblemas são as grandes contribuições de um projeto deste nível. Para a realização de uma meta deste porte, muita evolução nas áreas de pesquisa abrangidas pelo projeto RoboCup acontecerá,

através das metas de curto e médio prazo. Por estes motivos, os idealizadores da proposta RoboCup a encaram como um um marco na história da inteligência artificial.

Analisando a RoboCup como um problema padrão várias teorias, algoritmos, e arquiteturas podem ser avaliadas. Programas de computador que jogam xadrez são um exemplo típico de problema padrão. Vários algoritmos de busca foram avaliados e desenvolvidos usando este domínio. Com o recente desenvolvimento do super computador Deep Blue, que bateu Gary Kasparov, um grande mestre humano, usando as regras oficiais, o desafio de um programa de computador jogar xadrez está próximo do final.[1] Uma das razões que levaram estes programas ao sucesso como problema padrão é que a avaliação do progresso era claramente definida. Entretanto, os programas para xadrez estão próximos de cruzar sua meta, surgindo a necessidade de um novo desafio. A RoboCup pretende preencher esta lacuna.

Abaixo apresenta-se um tabela comparativa entre os dois domínios, xadrez e Robocup.

Domínio	Xadrez	RoboCup
Ambiente	Estático	Dinâmico
Mudança de Estado	A cada jogada	Tempo Real
Acesso a Informação	Completa	Incompleta
Sensores	Simbólico	Não Simbólico
Controle	Central	Distribuído

Tabela 2.1: Comparação entre problemas padrões

2.4 Um breve histórico da RoboCup

Esta seção é o relato da história da RoboCup baseado em [22].

Na história da inteligência artificial e da robótica, o ano de 1997 será lembrado como um divisor de águas. Em maio de 1997, o IBM Deep Blue derrotou o humano campeão mundial de xadrez. Quarenta anos após o desafio ser lançado à comunidade de inteligência artificial a meta foi alcançada com sucesso. Em 4 de julho de 1997, a Missão Pathfinder da Nasa obteve um pouso satisfatório e o primeiro sistema autônomo robótico, Sojouner, aterrissou na superfície de Marte. Junto com estas realizações, a RoboCup dava seus primeiros passos em direção ao desenvolvimento de jogadores de futebol robóticos que poderão bater, algum dia, a equipe campeã da Copa do Mundo de humanos.

A idéia de robôs jogarem futebol não é nova [35]. Independentemente deste trabalho, um grupo de pesquisadores japoneses organizou o **Workshop on Grand Challenges in Artificial Intelligen-**

[1] Para maiores informações sobre a partida ente Gary Kasparov e o super computador Deep Blue acesse `http://www.research.ibm.com/deepblue/home/html/b.html`

ce, em outubro de 1992, em Tóquio, onde a discussão era a proposta de novos problemas padrões. Este workshop levou a sérias discussões sobre o uso de partidas de futebol para promover ciência e tecnologia. Uma série de investigações foram efetuadas, incluindo um estudo sobre a viabilidade tecnológica, uma avaliação sobre o impacto social e um estudo sobre a viabilidade financeira. Além disso, regras foram esboçadas, bem como o desenvolvimento de protótipos de robôs reais e virtuais foram iniciados. Como resultado dos estudos, concluiu-se que o projeto era viável e desejável. Em junho de 1993, um grupo de pesquisadores, incluindo Minoru Asada, Yasuo Kuniuoshi, e Hiroaki Kitano, decidiram lançar uma competição robótica, primariamente denominada **J-League Robot** (J-League é o mesmo nome da liga japonesa de futebol profissional). Dentro de um mês, surgiram manifestações de pesquisadores de fora do Japão, requisitando que a iniciativa se estendesse como um projeto de participação internacional. Desta forma, renomeou-se o projeto como **Robot World Cup Initiative**, ou RoboCup em resumo.

Paralelo a esta discussão, vários pesquisadores já estavam utilizando o futebol como o domínio de sua pesquisa. Por exemplo, Itsuki Noda, no **Electro Technical Laboratory** (ETL), um centro de pesquisa do governo japonês, já conduzia pesquisas em sistemas multiagentes atacando o problema, e para isso começou o desenvolvimento de um simulador para partidas de futebol. Este, tornou-se mais tarde o simulador oficial da RoboCup. Também independentemente, o laboratório do Professor Minoru Asada, na Universidade de Osaka, a Professora Manuela Veloso e seu aluno Peter Stone da Universidade Carnegie Mellon, trabalhavam em robôs para jogar futebol. Sem a participação destes pesquisadores pioneiros no campo, a RoboCup poderia não ter surgido.

Em setembro de 1993, o primeiro anúncio público da iniciativa foi feito, e regras específicas foram traçadas. Conseqüentemente, discussões e assuntos técnicos foram surgindo em torno do tema em várias conferências e workshops, incluindo entre estes o **American Association of Artificial Inteligence** (AAAI-94) e ainda em vários encontros da sociedade de robótica.

Enquanto isso, a equipe de Noda da ETL anunciou a versão 0 do Soccer Server (versão em LISP), o primeiro simulador, habilitando a pesquisa em sistemas multiagentes. Em seguida veio a versão 1.0 do Soccer Server (versão em C++) que foi distribuída via web. A primeira demonstração pública do simulador aconteceu na IJCAI-95.

Durante a **International Joint Conference on Artificial Intelligence** (IJCAI-95), que aconteceu em Montreal, Canadá, em agosto de 1995, promoveu-se o anúncio do primeiro **Robot World Cup Soccer Games and Conferences** que aconteceria junto com o IJCAI-97 em Nagoya, Japão. Ao mesmo tempo, também foi formalizado o anúncio da Pre-RoboCup-96, a fim de identificar problemas

em potencial associados à organização da RoboCup em escala mundial. A decisão foi tomada no sentido de fornecer dois anos de preparação e de tempo de desenvolvimento, de forma que o grupo inicial de pesquisadores pudesse começar o desenvolvimento de suas equipes.

A Pre-RoboCup-96 aconteceu durante o **International Conference on Intelligence Robotics and Systems** (IROS-96), em Osaka, de 4 a 8 de novembro de 1996, com oito times competindo na liga de simulação além da demonstração de robôs reais da liga de robôs médios. Mesmo que limitada, esta competição foi a primeira usando partidas de futebol para pesquisa e educação.

A primeira RoboCup aconteceu em 1997 com grande sucesso. Contou com a participação de mais de 40 equipes (combinando robôs reais e virtuais), além da participação de mais de 5000 espectadores. A última RoboCup, aconteceu em Melbourne, Austrália, de 23 de agosto a 4 de setembro de 2000. Esta edição, contou com a demonstração de uma partida entre robôs humanóides, como uma prova do progresso da iniciativa.

2.5 Problemas gerais de ordem primária

A RoboCup foi concebida para tratar problemas de manipulação complexa, problemas do mundo real, de maneira integrada. Através destes problemas cobrem-se muitas áreas de robótica e inteligência artificial, dentre elas [31]:

- arquiteturas de robôs,
- capacidade de ações reativas coordenadas com ações de planejamento,
- reconhecimento, planejamento e raciocínio em tempo real,
- sistemas multiagentes,
- capacidade de aprendizado para tarefas complexas.

Para alcançar todas as metas pré-estabelecidas pela RoboCup, todas as áreas de pesquisas abordadas acima devem ser tratadas [6].

Arquitetura de Robôs

Uma tarefa fundamental no desenvolvimento de uma equipe robótica é o projeto do hardware do robô. Os robôs devem ser capazes de empreender múltiplas tarefas tais como encontrar e manipular a bola e, não obstante, evitar colisões com as paredes e com os demais robôs. O robôs necessitam

também ser móveis, ter um fornecimento de energia on-board, ter um sistema de comunicação sem fio (do inglês, "wireless"), além da habilidade de perceber a bola, os gols, o campo, sua própria posição no campo e a posição dos demais jogadores. Adicionalmente, os robôs devem também atender às restrições de tamanho de sua respectiva liga, bem como às restrições orçamentárias para por uma equipe em campo. Um problema adicional é criado pela própria natureza da competição. O time de robôs, juntamente com todo o hardware utilizado, necessitam ser transportados internacionalmente, e adaptar-se a diferentes condições de jogo.

Na liga de simuladores este problema, especificamente, não é totalmente verificado, uma vez que o agente projetado neste caso é virtual. Entretanto, no sentido de manipulação de múltiplas tarefas, continua sendo um problema a ser solucionado.

Capacidade de ações reativas coordenadas com ações de planejamento

Em uma partida de futebol, existe efetivamente um número infinito de situações diferentes. Para serem capazes de manipular todas estas possibilidades, os robôs devem ter mais que apenas ações reflexivas, devem possuir também a habilidade de planejar ações futuras.

Reconhecimento, planejamento e raciocínio em tempo real

Uma partida de futebol é uma situação tempo real. Isto significa que os robôs devem ser capazes de reagir quase instantaneamente a mudanças no seu ambiente. Esta exigência implica que os robôs não podem despender um excessivo período de tempo analisando e processando a informação antes de agir.

Enquanto o problema parece trivial para um humano, uma partida de futebol mostra-se como um problema elaborado para uma inteligência artificial. Considerando que a tarefa apresenta um espaço de estados muito grande, e que os estados mudam constantemente, determinar o comportamento ótimo em um dado instante de tempo é uma tarefa complexa. Além disso, a necessidade de um grupo de robôs agir coletivamente cria complicações adicionais. Táticas, trabalho de equipe e comunicação necessitam ser consideradas para a simples tarefa, por exemplo, de localizar e chutar a bola.

Sistemas multiagentes

Tal como no futebol entre humanos, onde existem dois times, o futebol de robôs envolve a interação de muitos robôs, metade dos quais têm objetivo oposto a outra metade. Este ambiente mostra-se muito rico para pesquisa em sistemas multiagentes. Dentre os assuntos envolvidos constam

interação entre agentes, comunicação e coordenação de planos global e local.

Capacidade de aprendizado para tarefas complexas

Dado que o espaço do problema é ilimitado, os robôs necessitam ter comportamentos que possam ser adaptados às mais variadas situações. Uma solução em potencial para este problema é criar um método para os robôs aprenderem. O uso de redes neurais tem sido um dos métodos sugeridos para realização desta tarefa.

2.6 Metas para o agente virtual

Como é da própria natureza da RoboCup, ela oferece uma meta que, com absoluta certeza, deve consumir alguns anos de pesquisa. Graças a clareza desta meta, é interessante a definição de algumas submetas, com o caráter de curto e médio prazo, com o propósito de avaliar o desenvolvimento do trabalho realizado pela comunidade científica. O cumprimento destas submetas levarão por conseqüência à conclusão da meta final.

Tratando objetivamente do desenvolvimento do agente virtual, existem três desafios básicos, de cumprimento a curto prazo, propostos em [32]:

Desafio de aprendizado - (do inglês "Learning challenge") que trata de máquinas de aprendizado em ambientes multiagente, colaborativo, na presença de um adversário.

Desafio para um trabalho de equipe - (do inglês "Teamwork challenge") que trata de arquiteturas multiagente, habilitando planejamento multiagente em tempo real e da execução dos planos a serviço de toda a equipe.

Desafio da modelagem do adversário - (do inglês "Opponent modeling challenge") que trata especificamente de técnicas para a modelagem de características encontradas em algum adversário específico.

Desafio de aprendizado

Os propósitos deste desafio podem ser declarados através de seus objetivos [32]:

> "Os objetivos do desafio de aprendizado da RoboCup é solicitar esquemas de aprendizado amplos, aplicáveis ao aprendizado de sistemas multiagentes que necessitam adaptar-se a

uma situação, e avaliar os méritos e deméritos da abordagem proposta usando tarefas padrão."

Em síntese, pode-se afirmar que a proposta deste desafio é a criação de métodos de treinamento e aprendizado para um grupo de agentes. Adequados a proposta do desafio, os métodos de aprendizado podem ser divididos dentro de alguns subitens:

1. Aprendizado de habilidades por agentes individuais em modo off-line;
2. Aprendizado colaborativo por times de agentes em modo off-line;
3. Aprendizado colaborativo e de habilidades em modo on-line;
4. Aprendizado de adversário em modo on-line.

É importante que as técnicas de aprendizado on-line apresentem um tempo de otimização rápido em relação ao desfecho de uma partida. Por exemplo, se existe um conjunto de rotinas específicas para o modelamento das características específicas do atual adversário, estas devem apresentar resultados antes do fim do jogo, antes que um novo adversário surja.

As prerrogativas deste desafio descrevem-se como a seguir:

Aprendizado de habilidades por agentes individuais em modo off-line: isto implica habilidades do tipo interceptar a bola, chutar a bola com a potência apropriada para um passe. Este tipo de aprendizado é trabalhado off-line uma vez que estas habilidades específicas são invariantes de jogo para jogo e não há a necessidade de reaprendê-las a cada início de uma nova partida.

Aprendizado colaborativo por times de agentes em modo off-line: significa habilidade para passar e receber a bola. Este tipo de habilidade é qualitativamente diferente das habilidades individuais de modo que deve haver a coordenação de múltiplos agentes.

Aprendizado colaborativo e de habilidades em modo on-line: embora os métodos de aprendizado off-line sejam úteis neste caso, podem existir vantagens no aprendizado incremental de tais habilidades. Por exemplo, aspectos particulares do comportamento de uma dada equipe adversária, podem render métodos de passe e chute mais apurados. Outra característica importante é capacidade de uma equipe de trocar um comportamento de acordo com as mudanças impostas pelo jogo. Os melhores times devem ter a capacidade de trocar configurações em resposta a eventos que ocorrem durante o transcorrer de um jogo.

Aprendizado de adversário on-line: significa capacidade para reagir a situações prognosticadas no oponente. Se um jogador pode identificar padrões de comportamento do adversário, ele deve ser capaz de explorar estes comportamentos previsíveis. Por exemplo, se o jogador número 4 do adversário sempre passa a bola para o número 6, então o número 6 deve estar sempre marcado, a partir do momento que o número 4 tem a posse da bola.

Como método de avaliação deste desafio, propõe-se que os agentes desenvolvidos devam enfrentar os times disponíveis publicamente e contra pelo menos um que não foi visto previamente. É proposto que estes confrontos dêem-se a partir de algumas situações pré-definidas, como por exemplo, o início de jogo com posições definidas em campo, e com alguns movimentos do adversário planejado previamente.

Desafio para um trabalho de equipe

Um trabalho de equipe pode ser descrito a partir de seu objetivo [32]:

> "Este desafio trata assuntos de planejamento em tempo real, replanejamento, e a execução de um trabalho de equipe multiagente em um ambiente dinâmico na presença de um adversário. Assuntos de interesse específico para este desafio são arquiteturas para planejamento em tempo real e a execução de planos em contexto de equipe. Além disso, generalidades da arquitetura para aplicações não-RoboCup serão um fator importante."

Dado um domínio complexo, dinâmico, multiagente, este exige comunicação e coordenação altamente flexíveis para superar as incertezas do ambiente, por exemplo, mudanças dinâmicas nas metas do time, incapacidade não esperada de um membro do time para cumprir suas responsabilidades, ou ainda, a descoberta, não aguardada, de oportunidades.

Um ambiente, como uma partida de futebol, caracteriza-se por apresentar um espaço de estados muito grande, prejudicando qualquer trabalho no sentido de algum planejamento específico. As incertezas envolvidas são muito grandes, e a equipe deve estar sempre preparada para uma mudança súbita na estratégia pré-planejada, uma tarefa complicada. As dinâmicas do domínio causadas pela não previsibilidade das ações do oponente criam situações consideravelmente mais difíceis.

Buscando uma arquitetura que tenha a capacidade de traçar um planejamento e posteriormente um plano de execução, apresentam-se as tarefas chaves de pesquisa envolvidas neste desafio:

Planejamento de contingência: Antes de uma partida começar, espera-se que a equipe trace um planejamento estratégico para o jogo que inclua uma contingência capaz de reconhecer mudanças

e eventualmente readaptar-se em tempo real. Dois desafios podem ser identificados para esta tarefa:

- Definição de ações estratégicas com condições e efeitos de aplicabilidade probabilística. A incerteza na especificação da ação está diretamente relacionada à identificação de possíveis divisões probabilísticas ou a eventos externos favoráveis.
- Definição de objetivos a alcançar. Neste domínio, as metas de vencer e marcar gols devem ser decompostas em uma variedade de metas mais concretas que servem para atingir a meta final.

Decomposição e união de planos: A correspondência entre metas e ações do time e metas e ações individuais deve ser fixada. A decomposição do plano do time pode criar metas individuais que não são necessariamente conhecidas pelo restante da equipe. Além disso, dentro do planejamento traçado, espera-se que exista uma variedade de metas dependentes do adversário e metas independentes do adversário. A decomposição, coordenação e união apropriada de planos individuais a serviço do planejamento principal permanece como uma área de pesquisa em aberto. A RoboCup oferece uma excelente estrutura para o estudo deste assuntos.

Execução dos planos da equipe: Este é o fator determinante na performance de uma equipe. A execução dos planos trata da coordenação de contingências que surgem durante a execução dos planos de coordenação específicos do domínio. A execução também monitora as condições de contingência que são parte do planejamento global da equipe. A seleção do curso apropriado de ações é dirigida pela informação de estado disparada pela execução.

Em termos de avaliação, um bom trabalho de equipe mede-se pela sua robustez. Robustez, neste caso, significa que uma equipe pode continuar a execução de sua missão, mesmo que mudanças não aguardadas aconteçam, tais como a remoção acidental de um jogador, mudança súbita na composição da equipe, ou até mesmo mudanças no ambiente de operação. Assim, para avaliação deste desafio, a equipe deve reagir positivamente com relação às seguintes alterações:

- Alguns jogadores serão desabilitados, ou sua capacidade será significativamente minada. Eventualmente, alguns jogadores podem ser reabilitados durante a partida.
- Troca de estratégia pelo oponente, onde o time deve responder com uma nova estratégia em tempo real.

- A situação assumida no primeiro ítem, só que aplicada ao oponente.
- Mudanças na formação da equipe durante a partida.
- Mudanças no fator climático.

Seria também interessante o confronto entre a equipe projetada contra uma outra que sabidamente atende às condições deste desafio com intuito de avaliar seu desempenho.

Finalmente, dada as premissas acima, espera-se que o sistema de planejamento multiagente em tempo real tenha a capacidade de integrar-se com a abordagem de aprendizado, isto é, que ele dinamicamente adapte seu comportamento completo baseado na sua experiência passada.

Desafio da modelagem do adversário

Modelagem e raciocínio a respeito das metas, planos, conhecimento ou habilidades do agente adversário, são assuntos chaves na interação multiagente. Este desafio trata da modelagem de um time adversário em um domínio dinâmico e multiagente. Neste sentido podemos dividi-lo em três partes:

Rastreamento on-line: envolve o rastreamento dinâmico das metas e intenções dos jogadores adversários baseado em observações de ações. Um jogador pode fazer uso deste rastreamento para predizer as jogadas e reações do oponente. Assim, se um jogar prever que o jogador 5 passará a bola para o jogador 4, ele pode, antecipadamente, marcar este último. Outro aspecto deste ponto é o uso de rastreamento do oponente no sentido de indução ao erro.

Reconhecimento de estratégia on-line: o agente "Coach" pode observar o jogo do lado de fora, e compreender as estratégias de alto-nível empregadas pela equipe adversária. Isto contrasta com o rastreamento on-line uma vez que o "Coach" pode efetuar uma análise abstrata, de alto-nível, com a ausência de pressões tempo real, elaborando uma análise mais detalhada.

Revisão off-line: agentes especialistas (do inglês, "Expert") podem observar alguns jogos envolvendo adversários, reconhecendo virtudes e defeitos dos times, e assim prover uma base de dados de jogos.

A forma encontrada para avaliação, em um primeiro momento, é a execução de partidas com e sem o rastreamento no intuito de comparar os resultados obtidos. Posteriormente, na presença do rastreamento, seriam incrementados novos comportamentos na equipe adversária no aguardo do reconhecimento dos mesmos pela estratégia imposta.

Capítulo 3

Soccerserver

3.1 Introdução

A RoboCup, no intuito de promover pesquisas na área de software, dentro de sua proposta [31] oferece uma plataforma capaz de simular uma partida de futebol entre robôs. O simulador, conhecido como *Soccerserver* [12], tem a propriedade de simular, a partir de um modelo numérico do ambiente, os movimentos dos jogadores (agentes) e da bola, ainda oferecendo a possibilidade de visualização do campo por intermédio de um display gráfico X windows. O Soccerserver também controla a partida, através de um módulo chamado *referee*, encarregado de mediar a partida segundo as regras do futebol estabelecidas pela FIFA[1]. É ainda de responsabilidade do simulador o envio de informações relativas aos sensores dos agentes a cada um dos seus respectivos clientes.

A proposta deste capítulo é a apresentação do Soccerserver, fornecendo as informações necessárias ao projeto de agentes capazes de jogar futebol em um ambiente dinâmico virtual. Este simulador foi a base dos trabalhos desenvolvidos e apresentados nesta dissertação.

3.2 Soccerserver

Soccerserver é um sistema que possibilita a agentes autônomos, consistindo de programas escritos em várias linguagens, jogarem uma partida de futebol uns contra os outros.

Uma partida é realiza no estilo cliente/servidor. Um servidor, o soccerserver, fornece um campo virtual e simula todos os movimentos da bola e dos jogadores. Cada cliente controla o movimento de um jogador. A comunicação entre o servidor e cada um dos clientes é feita via socket UDP/IP [43].

[1] Federation Internationale de Football Association: órgão regulador das regras e leis do futebol no mundo

Então, os usuários podem trabalhar com qualquer tipo de sistema de programação que apresente as facilidades UDP/IP.

O simulador consiste de dois programas, **soccerserver** e o **soccermonitor**. O **soccerserver** é um programa servidor que simula os movimentos de uma bola e dos jogadores, comunica-se com os clientes, e controla uma partida de acordo com as regras. Já o **soccermonitor** é um programa que apresenta um campo virtual no monitor, a partir do **soccerserver**, usando o sistema X windows. Diversos programas **soccermonitor** podem comunicar-se com um único **soccerserver**, de modo que o jogo pode ser acompanhado em vários monitores. Uma tela de amostra do campo é apresentada na figura 3.1.

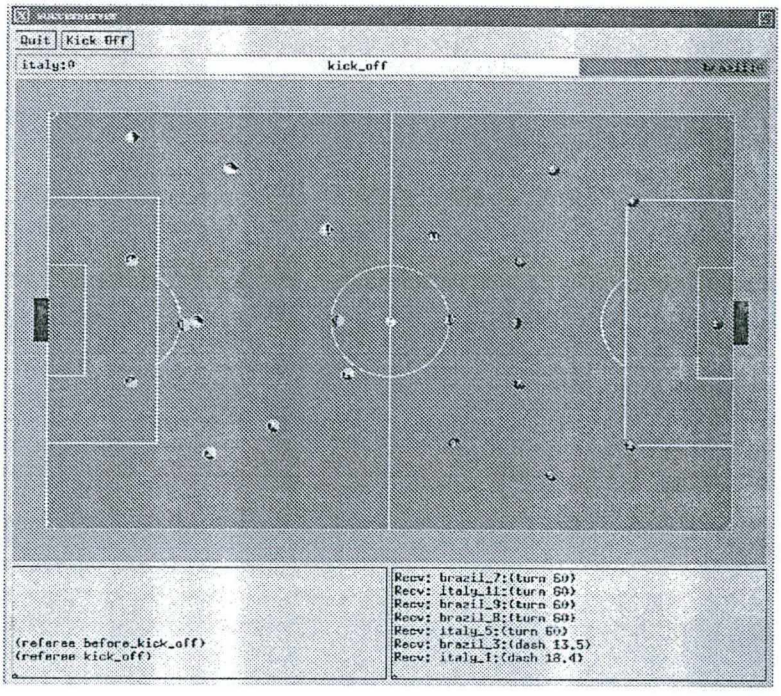

Figura 3.1: Tela do Soccer Server

Um cliente conecta-se com o **soccerserver** por um socket UDP. Através do socket, o cliente envia comandos para controlar o jogador do respectivo cliente e recebe informações captadas dos sensores do mesmo, como mostrado na figura 3.2. Ou seja, um programa cliente funciona como o cérebro do jogador: o cliente recebe informações dos sensores auditivos e visuais do servidor, e envia comandos de controle após o processamento destes.

Cada cliente pode controlar somente um jogador. Assim, um time consiste do mesmo número de clientes como de jogadores. Uma exigência da comissão que regulamenta a RoboCup é que toda

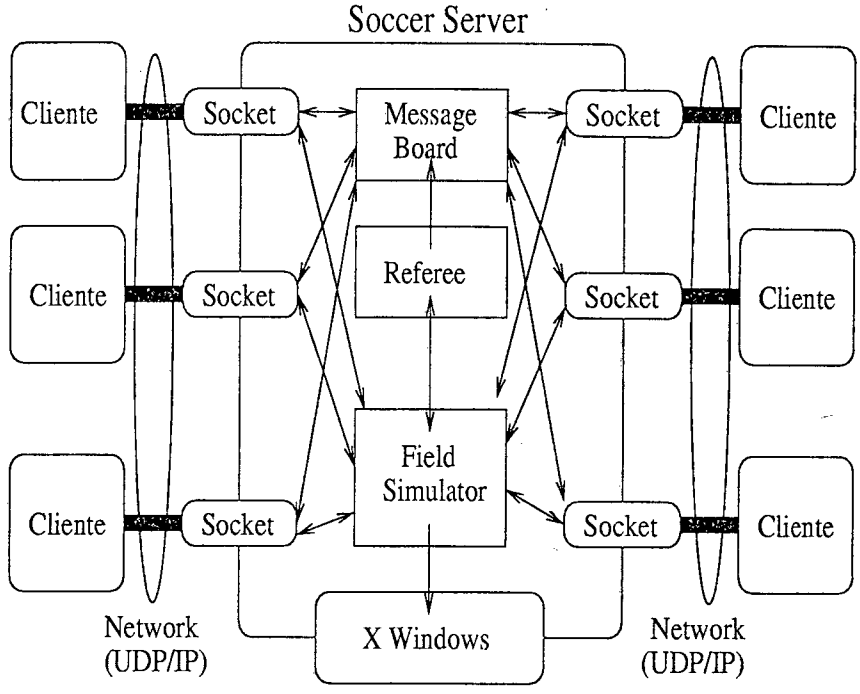

Figura 3.2: Arquitetura do Soccerserver

a comunicação entre clientes deve ser feita via **soccerserver**. Este compromisso dá-se no intuito de avaliar sistemas multiagentes, em que comunicação é um dos critérios.

Uma partida organizada pelo Soccerserver é realizada obedecendo aos seguintes passos:

1. Cada cliente de cada time conecta-se com o servidor por meio do comando `init`.
2. Quando todos os clientes estão prontos para jogar, o juiz da partida (a pessoa que executa o servidor) começa o jogo pressionando o botão kick-off na janela do **soccermonitor**. Assim, começa o primeiro tempo.
3. O primeiro tempo é de 5 minutos. Quando este termina, o servidor suspende a partida.
4. O intervalo é também de 5 minutos. Durante este tempo, os competidores podem trocar seus programas clientes.
5. Antes do reinício do jogo, cada cliente conecta-se com o servidor por um comando `reconnect`.
6. Quando todos os clientes estiverem prontos, o juiz reinicia a partida pressionando o botão kick-off.

7. O segundo tempo, como o primeiro, é de mais 5 minutos. Ao final, o servidor para o jogo.

8. No caso de empate, a prorrogação tem início. Esta termina no instante que o primeiro gol é marcado (morte súbita ou *golden goal*).

O sistema Soccerserver apresenta um conjunto de parâmetros que visam o modelamento do ambiente de simulação, tanto do **soccerserver** como do **soccermonitor**. O conjunto de parâmetros está descrito em [12], e estes podem ser modificados através dos comandos `soccerserver` e `soccermonitor`.

3.3 Regras

Como apresentado na figura 3.2, o soccerserver, através de um módulo denominado *referee*, arbitra a partida segundo as regras estabelecidas pela FIFA. No entanto existem algumas regras que são passíveis de interpretação, não podendo ser totalmente codificadas, devendo ser analisadas por um árbitro humano. Esta seção descreve as regras julgadas pelo servidor e pelo juiz humano.

3.3.1 Regras julgadas pelo servidor

O soccerserver controla a partida de acordo com as regras a seguir relacionadas.

Gol

Quando uma equipe marca um gol, o árbitro anuncia o gol difundindo uma mensagem para todos os clientes (*broadcast*), atualiza o placar, suspende a partida por 5 segundos, move a bola para o centro do campo, e modifica o *modo de partida* (do inglês "play mode") para kick_off. Durante este rápido período de suspensão, os jogadores devem voltar e posicionar-se no seu campo de defesa. Durante este intervalo, os clientes podem usar o comando `move`. Se um jogador permanece no campo de ataque depois do período de suspensão, o árbitro move o jogador para uma posição aleatória do seu campo de defesa (veja a regra "Início de jogo" abaixo).

Início de Jogo

Apenas antes do início de uma partida (do inglês "kick-off"), isto implica tanto o período que antecede o começo do jogo como a reposição de bola depois de um gol, todos os jogadores devem estar no seu próprio campo de defesa. Se os jogadores encontram-se no campo adversário durante

este período, o juiz move-os de volta a seu campo para uma posição escolhida arbitrariamente. Durante estes 5 segundos, os clientes podem usar o comando `move` para mover cada jogador para sua respectiva posição.

Fora do campo

Quando a bola sai de campo, o juiz move a bola para o respectivo lugar de reposição e modifica o *modo da partida* de acordo com a forma de reposição, que pode ser kick_in no caso de arremesso lateral, corner_kick para a cobrança de escanteio, ou ainda goal_kick no caso de um tiro de meta.

Distâncias

Quando o *modo de partida* é kick_off, kick_in, ou corner_kick, o árbitro remove todos os jogadores de defesa localizado dentro de um círculo centrado na bola com raio de 9.15m. Os jogadores removidos são postos no perímetro deste círculo.

Quando o *modo de partida* vai para offside, todos os jogadores de ataque são movidos de volta para uma posição fora do impedimento. Os jogadores em posição de impedimento são postos a 9.15m da bola.

Quando o modo de partida é goal_kick, todos os jogadores de ataque são movidos para fora da grande área. Estes não podem entrar na grande área, enquanto não acontecer a reposição de bola. O *modo de partida* muda para play_on imediatamente depois que a bola sai da grande área.

Controle do modo de partida

Quando o *modo de partida* atual é kick_off, kick_in, ou corner_kick o árbitro muda-o para play_on imediatamente depois que a bola começa seu movimento por intermédio de um comando `kick`.

Intervalo e final de jogo

O árbitro suspende a partida quando o primeiro ou o segundo tempo terminam. O tempo default de cada etapa é de 3000 ciclos de simulação (aproximadamente 5 minutos). Em caso de empate o jogo é extendido. A prorrogação extende-se até o momento que alguma das equipes marca um gol. O time autor deste gol vence a partida (morte súbita).

3.3.2 Regras julgadas pelo árbitro humano

Faltas como "obstrução" são difíceis para serem julgadas automaticamente uma vez que tratam das intenções do jogador. Para resolver tais situações, o servidor fornece uma interface para a intervenção humana. Deste modo, um árbitro humano pode suspender a partida e dar uma falta para um dos times. A seguir apresentam-se algumas das regras que devem ser julgadas por um árbitro humano:

1. Bloquear a bola com muitos jogadores
2. Não colocar a bola em jogo
3. Intencionalmente, bloquear o movimento de outros jogadores
4. Abuso do comando `catch`. O goleiro não pode soltar a bola e voltar a agarrá-la.

Em resumo, o árbitro humano deve tratar das violações não compreendidas pelo modo *referee* do soccerserver.

3.4 O cliente Treinador

Como em uma partida de futebol entre humanos, esta não pode ser interrompida por fatores externos ao jogo. Assim, nada nem ninguém, exceto os jogadores e o árbitro, podem influenciar e/ou controlar a partida. Entretanto, seria bastante útil se houvesse a possibilidade de existir um certo controle sobre a partida durante o desenvolvimento dos clientes. Por exemplo, a possibilidade de executar sessões de treinamento em que uma certa ação, tal como driblar, é testada de uma forma automatizada dando ao "treinador" a oportunidade de aplicar métodos de aprendizado de máquina.

Assim, um cliente privilegiado, denominado cliente *Treinador* (do inglês, "coach"), foi introduzido. Este cliente tem as seguintes atribuições:

- Pode controlar o *modo de partida.*
- Pode difundir mensagens de áudio. Tais mensagens podem consistir de um comando ou de alguma informação dirigida a um ou mais clientes. Seu significado e interpretação é definido pelo usuário.
- Pode mover um objeto (qualquer jogador ou até mesmo a bola) para qualquer posição do campo independente do atual *modo de partida.*

- Pode obter informações sobre as posições de todos os objetos móveis do campo.

O Treinador como juiz

O modo *referee* do soccerserver pode ser desabilitado, dando ao treinador toda a responsabilidade pelo monitoramento do *modo de partida*. Neste caso, o treinador fica com incumbência de arbitrar o jogo, aplicando as regras que julgar cabíveis. Este modo é interessante para o desenvolvimento de clientes.

O Treinador em um jogo

Neste modo, o objetivo é utilizar o cliente treinador para a observação do jogo construindo uma análise mais abstrata da partida buscando um planejamento estratégico a ser empregado pela equipe. Para este propósito, este cliente é capaz de:

1. obter informações a respeito da posição da bola e dos jogadores.
2. receber e difundir mensagens auditivas.

Não é permito ao treinador conduzir as ações de um jogador, mantendo o caráter autônomo deste. Assim, cabe ao treinador apenas a observação da partida e a partir desta análise aplicar as alterações táticas cabíveis.

3.5 Informação Sensorial

O simulador tem a responsabilidade de enviar a cada um dos clientes, via socket, informações a respeito de seus respectivos sensores. Em intervalos de tempo regulares, por default 150ms, o simulador envia ao cliente informação referente a sua câmera localizada no topo do respectivo jogador, ou seja, sua informação visual, constituindo-se em uma informação de caráter síncrono. Além da informação visual, o cliente recebe assincronamente mensagens auditivas, que são mensagens enviadas pelo árbitro bem como mensagens enviadas por outros clientes. A seguir são apresentados os modos como estas mensagens são enviadas, além da descrição da mensagem do tipo *Sense Body* que lista o status "físico" atual do cliente.

3.5.1 Informação Visual

Cada um dos clientes, conectados via socket ao servidor, recebe por meio desta conecção informações visuais relativas à câmera posicionada no topo do jogador. Essa informação visual chega do servidor no seguinte formato:

(ObjName Distance Direction DistChng DirChng BodyDir HeadDir)

ObjName ::= *(player Teamname UniformNumber)*
| *(goal [l—r])*
| *(ball)*
| *(flag c)*
| *(flag [l—c—r] [t—b])*
| *(flag p [l—r] [t—c—b])*
| *(flag g [l—r] [t—b])*
| *(flag [l—r—t—b] 0)*
| *(flag [t—b] [l—r] [10—20—30—40—50])*
| *(flag [l—r] [t—b] [10—20—30])*
| *(line [l—r—t—b])*

Distance, Direction, DistChng e DirChng são calculados a partir das seguintes expressões:

$$p_{rx} = p_{xt} - p_{xo} \tag{3.1}$$

$$p_{ry} = p_{yt} - p_{yo} \tag{3.2}$$

$$v_{rx} = v_{xt} - v_{xo} \tag{3.3}$$

$$v_{ry} = v_{yt} - v_{yo} \tag{3.4}$$

$$Distance = \sqrt{p_{rx}^2 + p_{ry}^2} \tag{3.5}$$

$$Direction = \arctan(p_{ry}/p_{rx}) - a_o \tag{3.6}$$

$$e_{rx} = p_{rx}/Distance \tag{3.7}$$

$$e_{ry} = p_{ry}/Distance \tag{3.8}$$

$$DistChng = (v_{rx} * e_{rx}) + (v_{ry} * e_{ry}) \tag{3.9}$$

$$DirChng = [(-(v_{rx} * e_{ry}) + (v_{ry} * e_{rx}))/Distance] * (180/\pi) \tag{3.10}$$

onde (p_{xt}, p_{yt}) é a posição do objeto observado, (p_{xo}, p_{yo}) é a posição do observador, (v_{xt}, v_{yt}) é a velocidade do objeto observado, (v_{xo}, v_{yo}) é a velocidade do observador, a_o é a direção absoluta para a qual o observador está voltado. Adicionalmente, (p_{rx}, p_{ry}) e (v_{rx}, v_{ry}) são respectivamente a posição relativa e a velocidade relativa do objeto observado, e (e_{rx}, e_{ry}) o vetor unitário paralelo ao vetor posição relativa. Os campos *BodyDir* e *HeadDir* são somente inclusos na informação caso o objeto observado seja um jogador, e são as direções do corpo e cabeça do jogador observado em relação as direções do corpo e cabeça do jogador que está observando. Assim, se os jogadores têm seus corpos apontados na mesma direção, então *BodyDir* seria 0. O mesmo acontece para *HeadDir*.

O objeto (*goal r*) é interpretado como o ponto central sob o gol posicionado do lado direito do campo. (*flag c*) é um flag virtual que situa-se no centro do campo. (*flag l b*) é o flag que se localiza no canto inferior esquerdo do campo. (*flag p l b*) é um flag virtual localizado no canto inferior direito da grande área posicionada do lado esquerdo do campo. (*flag g l b*) é um flag virtual que marca o poste direito localizado no gol do lado esquerdo do campo. Os tipos restantes de flags estão todos localizados a 5 metros fora do campo de jogo.

No caso de (*line ...*), *Distance* é a distância do ponto onde a linha de centro da visão do jogador cruza a dada linha, e *Direction* é a direção da linha.

Atualmente existem 55 flags (os gols contam como flags) e 4 linhas possíveis de serem visualizadas. Todos os flags e linhas são mostradas na figura 3.3.

Faixa de visão

O setor visível de um jogador é dependente de vários fatores. Primeiramente temos os parâmetros do servidor *sense_step* e *visible_angle* que determinam o passo de tempo básico entre a informação visual e quantos graus possui o cone de visão do jogador. Os valores default são 150ms e 90 graus. O jogador pode também influenciar na freqüência e qualidade da informação modificando os parâmetros *ViewWidth* e *ViewQuality*. Estes serão melhores descritos na seção 3.6. Para calcular os atuais valores de *view_frequency* e *view_angle* do agentes usa-se as seguintes equações:

$$\textit{view\_frequency=sense\_step*view\_quality\_factor*view\_width\_factor} \tag{3.11}$$

onde *view_quality_factor* é igual a 1 quando *ViewQuality* é igual a *high* e 0.5 para *ViewQuality* igual a *low*, *view_width_factor* é igual a 2 quando *ViewWidth* é igual a *narrow*, 1 para *ViewWidth* igual a *normal*, e 0.5 para *ViewWidth* igual a *wide*.

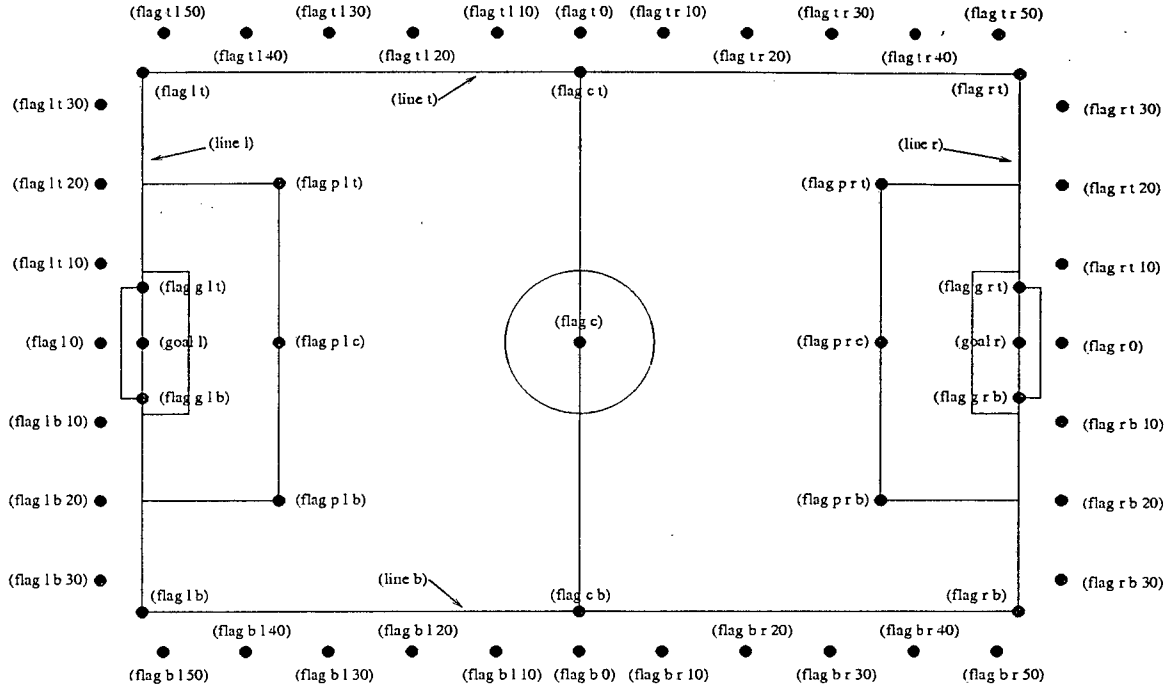

Figura 3.3: Localização dos flags e linhas de simulação

$$view\_angle = visible\_angle * view\_width\_factor \tag{3.12}$$

onde *view_width_factor* é igual a 0.5 para *ViewWidth* igual a *narrow*, 1 para *ViewWidth* igual a *normal*, e 2 para *ViewWidth* igual a *wide*.

O jogador pode também "ver" um objeto se este está dentro da distância estabelecida em *visible_distance*. Se o objeto está dentro desta distância mas não no cone de visão então o jogador pode saber somente o tipo do objeto (bola, jogador, gol, flag), mas não o nome exato do objeto.

O significado do parâmetro *view_angle* é ilustrado na figura 3.4.

3.5.2 Informação Auditiva

Uma informação auditiva é transmitida pelo servidor no seguinte formato:

(*hear Time Direction Message*)

onde *Direction* é a direção do remetente. No caso em que o enviador é o próprio cliente, *Direction* é igual a *self*. *Time* indica o tempo atual de simulação. Esta mensagem é enviada imediatamente

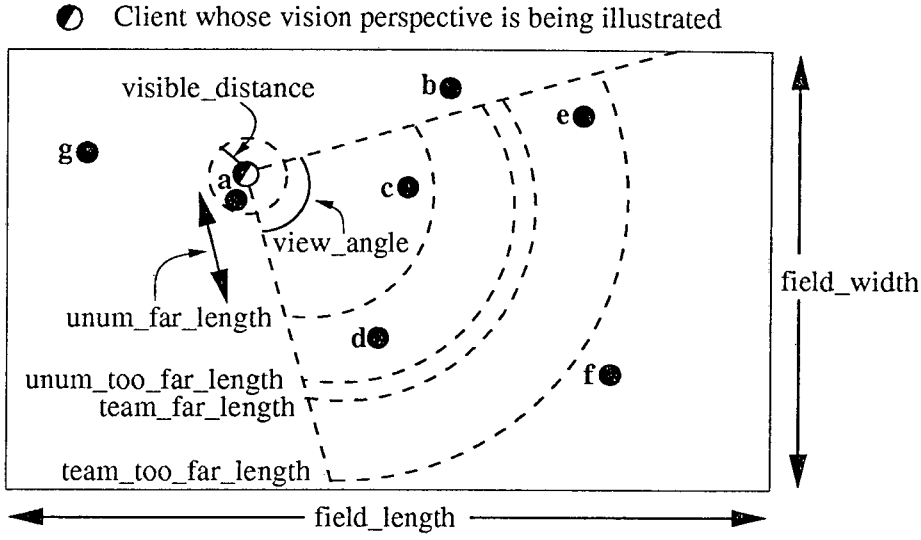

Figura 3.4: Campo visual do jogador

quando um cliente envia o comando (*say Message*). Julgamentos do árbitro são transmitidos também desta forma. Neste caso *Direction* é *referee*.

Uma mensagem enviada por um jogador é transmitida somente para os jogadores dentro de um raio de 50 metros. Mensagens do árbitro podem ser ouvidas por todos os jogadores. Um jogador pode ouvir somente uma mensagem de cada time durante cada 2 ciclos de simulação. Se várias mensagens chegam do mesmo time no mesmo momento elas são escolhidas de acordo com a ordem de chegada. De acordo com a regra um jogador pode ainda ouvir e enviar uma mensagem no mesmo ciclo de simulação.

3.5.3 Informação de Status

A partir de sua versão 5.x, o servidor envia uma mensagem do tipo (*sense_body*) em intervalos periódicos de tempo definido no parâmetro *sense_step*, por default 100ms, que retorna informações sobre o status "físico" atual do agente. O formato deste tipo de mensagem é:

(sense_body Time

(view_mode ViewQuality ViewWidth)

(stamina Stamina Effort)

(speed AmountOfSpeed)

(head_angle HeadDirection)

(kick KickCount)

(dash DashCount)

(turn TurnCount)

(say SayCount)

(turn_neck TurnNeckCount))

A variável *ViewQuality* pode assumir os valores *high* ou *low* e a variável *ViewWidth* pode assumir os valores *narrow*, *normal*, e *wide*. Estes são importantes fatores na qualidade e na freqüência da informação visual. *AmountOfSpeed* é uma aproximação do valor da velocidade do jogador. Se a direção da velocidade é necessária esta deve ser estimada de uma outra forma. *HeadDirection* é a direção relativa da cabeça do jogador. As variáveis *Count* representam o número total de vezes que o dado comando foi executado pelo servidor. Por exemplo, *DashCount* = 134 significa que o comando `dash` foi executado 134 vezes.

3.6 Comandos do Soccerserver

Após o processamento das informações visuais e auditivas recebidas pelo agente, este deve ser capaz de responder ao soccerserver com os comandos apropriados naquele instante de simulação. O soccerserver aceita um novo comando de simulação a cada 20ms, entretanto existem limitações para utilização destes comandos. Os comandos disponíveis e suas respectivas limitações são:

- (`turn` *mi*) - Permite que o jogador execute um giro de [-180, 180] graus em torno do seu próprio eixo. A este comando é associado um argumento *mi*, ou seja, o valor do ângulo em graus.
- (`turn neck` *mi*) - Permite que o jogador gire [-180, 180] graus em torno de seu próprio eixo, a parte superior do robô onde se encontra a câmera.
- (`dash` *p*) - Incrementa a velocidade do jogador na direção atual com a potência *p* especificada no argumento. A potência pode assumir um valor inteiro no intervalo [-30, 100].

- (`kick` *p d*) - Chuta a bola com a potência *p* [-30, 100] na direção *d* [-180, 180]. Este comando só será executado se a bola estiver dentro da distância estabelecida pelos parâmetros do simulador (a distância deve ser menor que *kickable_margin* + *ball_size* + *player_size*).

- (`catch` *d*) - Tenta agarrar a bola na direção *d*. A possibilidade de sucesso é definida no parâmetro *catch_possibility*. A bola deve encontrar-se em uma área definida pelo parâmetro *goalie_catchable_area*, um retângulo que por default é de 2m X 1m. Este comando deve ser utilizado apenas pelo goleiro.

- (`say` *msg*) - Difunde uma mensagem para todos os jogadores em um raio de 50m, com o centro no jogador que a enviou.

- (`change_view` *a q*) - Modifica o ângulo do setor visível para *a* ([-45, 45], [-22.5, 22.5], [-90, 90]) e a qualidade visual para *q* (*low*, *high*).

Os valores limites dos intervalos apresentados acima são todos parâmetros ajustáveis do simulador. O soccerserver aceita um novo comando a cada 20ms, entretanto apenas um comando `turn`, `kick` ou `dash`, é executado a cada ciclo de simulação (100ms).

3.7 Temporização

A comunicação entre o soccerserver e os agentes envolve mensagens síncronas e assíncronas [14], como demonstrado na figura 3.5.

- **Tempo de simulação** - Cada ciclo de simulação tem a duração de 100ms.

- **Aceitação de comandos** - O soccerserver aceita um novo comando a cada 20ms.

- **Informação visual** - Depende do modo de visão utilizado (*normal*, *narrow*, *wide*) e da qualidade da informação visual (*high*, *low*) utilizada. Para o modo *normal* com qualidade da informação *high*, uma nova mensagem contendo a informação visual é recebida a cada 150ms.

- **Informações Auditivas** - São trocadas assincronamente pelo árbitro e pelos demais agentes.

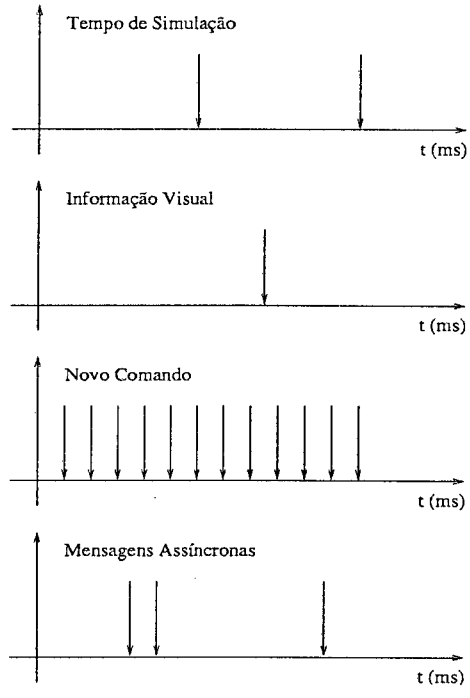

Figura 3.5: Temporização do soccerserver

Capítulo 4

Sistemas Especialistas

4.1 Introdução

Neste e nos próximos dois capítulos serão apresentados as áreas de pesquisa integradas pelo projeto UFSC-Team, e que são de vital importância para a implementação deste trabalho de dissertação, onde serão abordados tópicos conceituais bem como suas aplicações em outros domínios. Neste capítulo especificamente, são apresentados os Sistemas Especialistas (SE), que em uma primeira definição podemos descrever como programas de computador que têm como objetivo simular a perícia de especialistas humanos solucionando problemas do mundo real, em um domínio restrito. Dentro do agente proposto para o UFSC-Team, os SE's representam um papel fundamental, implementando os motores de inferência contidos dentro dos níveis instintivo e cognitivo do agente. O capítulo inicia com dados relativos a história dos SE's. A seguir, é apresentada a arquitetura de um SE, onde os elementos que o compõe são descritos. Na seção seguinte assuntos relativos a aquisição de conhecimento são discutidos finalizando com a descrição dos formalismos de representação de conhecimento mais conhecidos.

4.2 Sistemas Especialistas

A tecnologia de SE's deriva de disciplinas de pesquisa da *Inteligência Artificial* (IA): um ramo da Ciência da Computação interessada no projeto e implementação de programas que são capazes de emular as habilidades cognitivas humanas tais como resolução de problemas, percepção visual e compreensão da linguagem [28]. Esta tecnologia tem sido aplicada com bastante sucesso a uma vasta diversividade de domínios incluindo química orgânica, exploração mineral, e medicina. Tarefas

típicas de sistemas especialistas envolvem:

- interpretação de dados (tal como em sinais de sonar),
- diagnóstico de disfunções (tal como em falhas de equipamentos),
- análise estrutural de objetos complexos (tal como em componentes químicos),
- configuração de objetos complexos (tal como sistemas computacionais), e
- planejamento de seqüência de ações. (tal como as necessárias para controlar um robô)

A área de SE é uma solução viabilizadora do clássico problema de programação inteligente. O Professor Edward Feigenbaun da Universidade de Stanford, um pioneiro da tecnologia de SE's, definiu um SE como [2]:

> "...um programa de computador que usa conhecimento e procedimentos de inferência para resolver problemas que são difíceis o suficiente a ponto de exigirem um especialista humano para a sua solução."

Repare que o termo *emular* faz-se literalmente presente na primeira definição descrita no primeiro parágrafo desta seção bem como em várias outras. Este termo implica que um SE tem como objetivo agir como um especialista humano em todos os aspectos do domínio. Uma emulação é mais forte que uma simulação no sentido de que esta última busca agir como a coisa real somente em alguns aspectos [26].

Segundo [28], existem algumas prerrogativas que permitem que se chame um programa de computador de especialista. Dentre estas estão:

- Uma exigência básica é a de que o programa deve *possuir conhecimento*. O simples fato de possuir um algoritmo, como por exemplo algum tipo de lista de coisas para testar, não é realmente suficiente.
- Este conhecimento deve estar focado em um *domínio específico*. Uma coleção aleatória de datas, nomes, lugares e velhos provérbios não é o tipo de conhecimento que provê uma base de perícia. Conhecimento implica organização e integração, de forma que diferentes pedaços de conhecimento relacionando-se com outros pedaços formarão novos conhecimentos.
- Finalmente, este conhecimento deve prover a capacidade de *resolução de problemas*. Ou seja, o sistema deve ser capaz de inferir soluções a partir do conhecimento armazenado.

Unindo estas considerações podemos formular, segundo [28], que:

> "Um sistema especialista é um programa de computador que representa e raciocina com o conhecimento de algum especialista com o objetivo de resolver problemas do seu domínio de abrangência."

A arquitetura dos SE's que será apresentada na próxima seção deriva dos chamados *Sistemas de Produção*, que é um nome genérico para todos os sistemas baseados em *regras de produção*. Estas regras de produção são pares de expressões consistindo em uma condição e uma ação [8]. A idéia inicial dos sistemas de produção foi introduzida por Post, em 1936, quando ele propôs os hoje chamados *sistemas de Post.* [38]. Um sistema de Post consiste em um conjunto de regras para a especificação sintática de transformações sobre cadeias de caracteres, e representa, como demonstrou Post, um método geral para o processamento de dados. Os sistemas de produção foram redescobertos durante os anos setenta como uma ferramenta para a modelagem da psicologia humana. O formato condição-ação se adapta à modelagem de todos os comportamentos baseados em pares estímulo-resposta.

Os SE's também utilizam o formato de regras de produção como método de representação de conhecimento. O objetivo dos SE's é, ao mesmo tempo, mais restrito e mais ambicioso do que o objetivo dos modelos psicológicos: os SE's são concebidos para reproduzir o comportamento de especialistas humanos na resolução de problemas do mundo real, mas o domínio destes problemas é altamente restrito. Os primeiros SE's que obtiveram sucesso em seu objetivo foram os sistemas DENDRAL [23] e MYCIN [42]. O sistema DENDRAL é capaz de inferir a estrutura molecular de compostos desconhecidos a partir de dados espectrais de massa e de resposta magnética nuclear. O sistema MYCIN auxilia médicos na escolha de uma terapia de antibióticos para pacientes com bacteremia, meningite, e cistite infecciosa, em ambiente hospitalar. Desde então, muitos SE's foram desenvolvidos para resolver problemas em muitos domínios diferentes, incluindo: agricultura, química, sistemas de computadores, eletrônica, engenharia, geologia, gerenciamento de informações, direito, matemática, medicina, aplicações militares, física, controle de processos e tecnologia espacial.

Uma forma de abordar as características de um SE é comparando-o com outras abordagens utilizadas para resolver problemas que possam ser resolvidos por ele [28]. Um SE pode ser distinguido de programas convencionais uma vez que:

- Ele *emula o raciocínio humano* em um dado problema, ao invés de emular o domínio do problema. Isto distingue os SE's de estilos de programação convencional que envolvem modelamento

matemático ou animação computacional. Isto não significa dizer que o programa é um modelo psicológico fiel do especialista, apenas seu foco está em emular a habilidade do especialista em resolver problemas, isto é, executar uma porção de tarefas relevantes da mesma maneira, ou melhor, que o especialista.

- Ele raciocina a partir de *representações do conhecimento humano*, além de fazer cálculos numéricos ou recuperação de dados. O conhecimento em um programa é normalmente expresso em uma linguagem de propósito especial e mantém-se separado do restante do código que executa o raciocínio. Estes módulos distintos de programa são referenciados como *base de conhecimento* e *motor de inferência*, respectivamente.

- Ele resolve os problemas por *métodos heurísticos* ou *métodos de aproximação* que, diferentemente de soluções algorítmicas, não apresentam garantia de sucesso. Uma heurística é essencialmente uma regra de manuseio que codifica um pedaço do conhecimento sobre como resolver problemas em um dado domínio. Tais métodos são aproximativos no sentido de que (i) eles não requerem dados perfeitos (ii) as soluções derivadas do sistema podem ser propostas com graus de certeza variável.

Um SE difere de outros tipos de programas de IA uma vez que:

- Ele trata de assuntos de *complexidade prática* que normalmente exige uma quantidade razoável de perícia humana. Muitos programas de IA são realmente veículos de pesquisa, e podem então focar-se em problemas matemáticos abstratos ou versões simplificadas de problemas reais a fim de adquirir perspicácia ou refinar técnicas.

- Ele deve exibir *alta performance* (do inglês "high performance") em termos de velocidade e confiabilidade a fim de ser uma ferramenta útil. Veículos de pesquisa em IA podem não rodar muito rápido, bem como podem conter bugs. Mas um SE deve propor soluções em um tempo razoável e deve ser direto na maioria das vezes, isto é, como um especialista humano.

- Ele deve ser capaz de *explicar e justificar as soluções* ou recomendações a fim de convencer o usuário que está raciocinando de maneira correta. Programas de pesquisa são tipicamente executados somente por seus criadores, ou por outras pessoas em laboratórios similares. Um SE será executado por uma grande faixa de usuários, e deve ser então projetado de tal modo que seus trabalhos sejam mais transparentes.

O termo *sistema baseado em conhecimento* (do inglês "knowledge-based system") é em alguns momentos usado como um sinônimo para SE, embora, estritamente falando, o primeiro é mais geral. Um sistema baseado em conhecimento é qualquer sistema que executa uma tarefa aplicando regras de manuseio para um representação simbólica de conhecimento, ao invés de empregar principalmente métodos algoritmicos ou estatísticos. Desta forma, um programa capaz de conversar sobre o tempo seria um sistema baseado em conhecimento, mesmo se o programa não incorporar qualquer tipo de perícia em metereologia, mas um SE no domínio de metereologia deve ser capaz de fornecer-nos a previsão do tempo.

4.3 Arquitetura dos Sistemas Especialistas

Os sistemas de produção de Post evoluíram para o que hoje é conhecido como SE's. Buscando uma maior eficiência e expressividade a arquitetura dos sistemas de produção de Post foi generalizada, apresentando uma aparência como demonstrado na figura 4.1. Esta nova arquitetura apresenta três módulos: uma *base de regras*, uma *memória de trabalho* e um *motor de inferência*. A base de regras mais a memória de trabalho formam a chamada *base de conhecimento* do SE. O motor de inferência é o mecanismo responsável por buscar na base de regras os padrões encontrados na memória de trabalho e executar a ação apropriada.

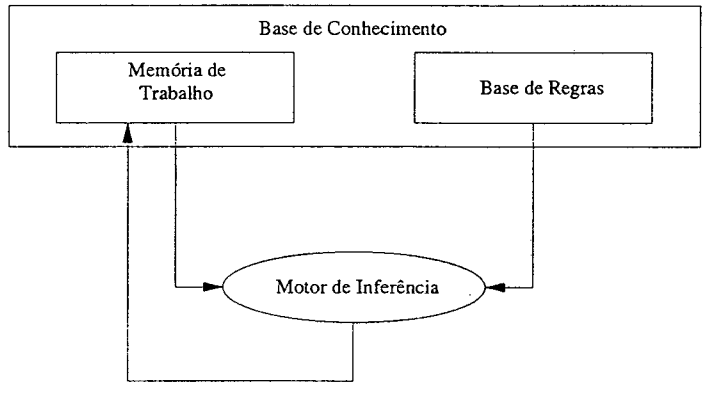

Figura 4.1: Arquitetura de um SE

A memória de trabalho, que limitava-se a apenas uma seqüência de caracteres no modelo de Post, no modelo generalizado pode conter qualquer tipo de estrutura de dados. Entretanto, esta estrutura deve respeitar um *método de representação de conhecimento* (ver seção 4.5), isto é, uma linguagem formal e uma descrição matemática de seu significado. A lógica de primeira ordem enquadra-se como

um exemplo de representação de conhecimento.

A base de regras passa a conter condições que simbolizam "perguntas" à representação de conhecimento armazenada na memória de trabalho, que geralmente envolvem variáveis a serem instanciadas e eventualmente algum tipo de inferência. No modelo de Post estas perguntas eram apenas comparação de caracteres. Já hoje a sintaxe das regras varia de acordo com o sistema e pode ser bastante flexível e próxima da linguagem natural, como nos sistemas ROSIE [21] e G2 [25], ou bastante formais, como na família de sistemas OPS [24]. Atualmente, já existem inclusive, mecanismos de raciocínio incerto que permitem representar a incerteza a respeito do conhecimento do domínio.

O motor de inferência controla a atividade do sistema. Esta atividade ocorre em ciclos, cada ciclo consistindo em três fases:

1. Correspondência de dados, onde as regras que satisfazem a descrição da situação atual são selecionadas.

2. Resolução de conflitos, onde as regras que serão realmente executadas são escolhidas dentre as regras que foram selecionadas na primeira fase, e ordenadas.

3. Ação, a execução propriamente dita das regras.

As principais vantagens dos sistemas de produção como método de representação de conhecimento são [45]: modularidade, uniformidade e naturalidade. As principais desvantagens são: ineficiência em tempo de execução e complexidade do fluxo de controle que leva a solução dos problemas. Estas vantagens e desvantagens acabam determinando quais domínios são adequados para o desenvolvimento de SE. Estes domínios devem:

- ser descritos por um conhecimento consistindo em um conjunto muito grande de fatos parcialmente independentes,

- dispor de métodos de solução consistindo de ações independentes, e

- apresentar uma nítida separação entre conhecimento e ação.

O conhecimento de um SE deve ser adquirido junto a um especialista humano do domínio e representado de acordo com o método de representação de conhecimento escolhido. Esta fase do desenvolvimento é crucial uma vez que a chave para o desempenho de um SE está no conhecimento armazenado nas suas regras e em sua memória de trabalho.

4.4 Aquisição de Conhecimento

O processo de construção de uma base de conhecimento é chamado de *engenharia de conhecimento* [40]. Um engenheiro de conhecimento é alguém que investiga um domínio particular, determina quais conceitos são importantes neste domínio, e cria uma representação formal dos objetos e relações do domínio. Na maioria das vezes, o engenheiro de conhecimento é treinado na representação mas não é um especialista no domínio que manipula. O engenheiro, então, normalmente entrevista os especialistas reais de forma a educar-se sobre o domínio que está tentando representar a fim de deduzir o conhecimento necessário, em um processo denominado *aquisição de conhecimento*. Este processo ocorre antes, ou de forma intercalada, da criação das representações formais.

Segundo [10], aquisição de conhecimento define-se como:

> "... a transferência e transformação de perícia potencial para resolução de problemas de alguma fonte de conhecimento para um programa."

No entanto, o método de entrevistas como forma de absorção do conhecimento necessário apresenta uma produtividade muito pobre. É estimado que esta forma de trabalho produza entre duas a cinco unidades de conhecimento por dia (por exemplo, regras de manuseio) [28]. As razões para isto devem-se a:

- Cada domínio tem seus próprios jargões, e é muito difícil muitas vezes para um especialista expressar seu conhecimento em uma linguagem cotidiana. A análise dos conceitos que estão por trás dos jargões dificilmente dá-se de forma direta, desde que este conceitos não necessitam de matemática precisa ou uma definição lógica.

- Os fatos e princípios subjacentes de muitos domínios de interesse não podem ser caracterizados precisamente em termos de uma teoria matemática ou um modelo determinístico cujas propriedades são bem entendidas.

- Especialistas necessitam saber mais que meros fatos ou princípios para resolver problemas. Por exemplo, eles normalmente sabem que tipo de informação é relevante para um dado julgamento, o quão confiável é uma fonte de informação, e como fazer problemas difíceis tornarem-se mais fáceis pela divisão destes em subproblemas. Elucidar este tipo de conhecimento, que é normalmente baseado em experiência pessoal ao invés de treinamento formal, é muito mais difícil que elucidar outros fatos particulares ou princípios gerais.

- É difícil delinear a quantidade e a natureza de conhecimento geral necessário para tratar com um caso arbitrário.

Fica evidente que a parte mais sensível no desenvolvimento de um SE é, certamente, a aquisição de conhecimento. Além dos problemas já evidenciados acima, esta aquisição não pode limitar-se à adição de novos elementos de conhecimento à base de conhecimentos; é necessário integrar o novo conhecimento ao conhecimento já disponível, através da definição de relações entre os elementos que constituem o novo conhecimento e os elementos já armazenados na base [8]. Dois tipos de mecanismos para a definição de tais relações foram propostos: ligar os elementos de conhecimento diretamente através de ponteiros, ou reunir diversos elementos relacionados em grupos (do inglês "clustering").

Outro ponto importante na aquisição de conhecimento é o tratamento de incoerências. Dependendo da forma como o novo conhecimento é adquirido, podem haver erros na aquisição. Técnicas foram desenvolvidas para evitar erros de aquisição, como, por exemplo, a especificação de regras de aquisição em que o tipo de conhecimento esperado é definido. Por outro lado, uma base de conhecimento pode ser examinada periodicamente com a finalidade de detectar incoerências eventualmente introduzidas no processo de aquisição.

A insatisfação com o método de entrevista tem levado alguns pesquisadores a tentar automatizar o processo de aquisição de conhecimento. Trata-se da área de pesquisa conhecida como *extração automática de conhecimento* (do inglês "automated knowledge elicitation"), em que o conhecimento de um especialista é transferido para um programa de computador como efeito colateral do diálogo entre o especialista e o computador. Outra forma encontrada com o intuito de contornar este problema é através de outro campo de pesquisa conhecido como *aprendizado de máquinas* (do inglês "machine learning") [28].

4.5 Métodos para Representação de Conhecimento

Representação de conhecimento é uma subárea substancial da IA, que compartilha muitos interesses tanto com a filosofia formal como com a psicologia cognitiva. Ela está interessada no modo como a informação pode ser armazenada e associada no cérebro humano, normalmente de uma perspectiva lógica, ao invés de biológica. Em outras palavras, ela não está interessada nos detalhes físicos de como o conhecimento é codificado, mas sim com seu esquema conceitual global.

Em termos de SE's, a representação de conhecimento está mais interessada no modo com

que grandes corpos de informações podem ser formalmente descritos para os propósitos de uma computação simbólica. *Descrita formalmente* significa a utilização de alguma linguagem ou notação não ambígua que possui uma *sintaxe* bem definida orientando a forma das expressões na linguagem, e uma *semântica* bem definida que revela o significado de tais expressões em virtude de sua forma [28].

A escolha do método de representação de conhecimento é de maior importância em um SE por duas razões. Primeiro, a estrutura do SE é projetada para um certo tipo de representação de conhecimento tais como lógica ou quadros. Segundo, o modo com que um SE representa seu conhecimento afeta o desenvolvimento, eficiência, velocidade e manutenção do sistema [26]. A linguagem associada ao método escolhido deve ser suficientemente expressiva (mas não *mais* do que o suficiente) para permitir a representação do conhecimento a respeito do domínio escolhido de maneira completa e eficiente. Em tese, uma representação geral como a lógica seria suficientemente expressiva para representar qualquer tipo de conhecimento. No entanto, problemas de eficiência, facilidade de uso e a necessidade de expressar conhecimento incerto e incompleto levaram ao desenvolvimento de diversos tipos de formalismos de representação de conhecimento. A seguir apresentam-se alguns dos formalismos de representação de conhecimento mais utilizados.

Lógica

A *lógica* é um formalismo largamente utilizado em SE. As duas principais vantagens deste tipo de formalismo são: a expressividade inerente da linguagem e uma semântica bem definida e bastante estudada. Uma base de conhecimento construída utilizando-se a lógica como formalismo, consiste em um conjunto de fórmulas lógicas, que podem ser representadas, por exemplo, em expressões LISP. Este formalismo adequa-se a domínios onde o conhecimento é largamente desestruturado e consiste em uma coleção de fatos independentes. O método de inferência pode ser uma simples unificação ou dedução automática.

Quadros

Os *quadros* (do inglês "frames") foram propostos por Marvin Minsky em 1975 [36], como um formalismo de representação de conhecimento. Uma base de conhecimento construída utilizando-se deste formalismo consiste de uma hierarquia de estruturas de dados chamada de quadros. Cada quadro possui um conjunto de atributos onde um valor pode ser associado a cada atributo. Este valor pode ser qualquer informação primitiva sobre o conceito representado pelo quadro, ou um ponteiro

para um outro quadro. Existem três mecanismos de inferência integrados no formalismo: herança de valores de atributos através da hierarquia, valores previamente escolhidos ("default") podem ser utilizados quando não há informação disponível e ligação procedural para permitir a execução de uma função externa ao formalismo. Este tipo de formalismo é adequado para domínios estruturados taxonomicamente, onde o mecanismo de herança pode ser explorado eficientemente.

A linguagem de quadros permite a definição de estruturas complexas de dados com procedimentos complexos de herança através de hierarquia e facilidade de ligações procedurais.

Redes Semânticas

As *redes semânticas* foram introduzidas por Quillian em 1968 [39], como um modelo para a memória associativa do ser humano. Este formalismo consiste em um grafo orientado. Os nodos podem ser utilizados para representar conceitos primitivos, predicados, objetos, etc. Os arcos são utilizados para associar os nodos com relações binárias. Os arcos podem ser positivos ou negativos permitindo assim a representação explícita de exceções.

O mecanismo de inferência básico é a herança através dos arcos do grafo. Na ausência de arcos negativos o mecanismo é simples e eficiente. A introdução de arcos negativos, representando exceções, exige um maior cuidado para que os resultados do mecanismo de inferência correspondam à interpretação intuitiva desejada.

Capítulo 5

Algoritmos Genéticos

5.1 Introdução

Os *Algoritmos Genéticos* (AG) são métodos adaptativos que podem ser usados para resolver problemas de busca e otimização, uma família de modelos computacionais inspirados pela teoria da evolução natural. Eles são baseados nos processos genéticos de organismos biológicos. Através de muitas gerações, populações naturais evoluem de acordo com os princípios de seleção natural, que foram primeiramente propostos por Charles Darwin no livro *A Origem das espécies*. Analogamente ao processo natural, os AG's são capazes de "evoluir" soluções para problemas do mundo real, desde que sejam codificados convenientemente. Por exemplo, AG's podem ser usados para projetar estruturas de pontes, pela taxa força/peso máxima, ou ainda para determinar o "layout" mais econômico para o corte de formas em tecidos[3]. Eles também podem ser usados para controle de processos "online", tais como uma planta química, ou balanceamento de carga em um sistema computacional multiprocessador. Dentro do agente proposto para o UFSC-Team, os AG's são utilizados basicamente de duas formas: (i) no primeiro caso eles visam determinar os valores ótimos dos controladores nebulosos presentes no nível reativo do agente e futuramente na determinação das regras destes controladores, (ii) e no segundo caso na otimização dos valores limiares que definem o estado de uma partida, valores estes estabelecidos no nível instintivo do agente.

Os princípios básicos dos AG's foram inicialmente estabelecidos de forma rigorosa em [27]. Eles simulam aqueles processos em populações naturais que são essenciais à evolução. Exatamente quais processos biológicos são *essenciais* para a evolução, e quais processos têm pouco ou nenhum papel é ainda um problema a pesquisar, mas as fundações são claras.

Na natureza, indivíduos em uma população competem entre si por recursos tais como comida,

água e abrigo. Além disso, membros de uma mesma espécie sexuada competem entre si em busca de um companheiro para reprodução. Aqueles indivíduos que obtêm um melhor resultado em termos de sobrevivência e na busca por um companheiro terão um número relativamente maior de descendentes. Performances individuais pobres renderão poucos, ou até mesmo nenhum descendente. Isto significa que os genes dos mais adaptados, se difundirão a um número maior de indivíduos nas próximas gerações. A combinação de boas características de diferentes antepassados podem algumas vezes produzir um "superindivíduo", cuja aptidão é maior que de qualquer pai. Deste modo, espécies evoluem para se tornar mais e mais adaptadas a seu ambiente [3].

Os AG's usam uma analogia direta ao comportamento natural. Eles trabalham com uma *população* de "indivíduos", onde cada um representa uma possível solução para um dado problema. A cada indivíduo é designado um "valor de aptidão" (do inglês "fitness score") de acordo com quão boa é a solução para o problema. Por exemplo, um valor de aptidão pode ser a taxa força/peso para um dado projeto de ponte. Na natureza isto é equivalente a designar o quão efetivo um organismo é na competição por recursos. Aos indivíduos com maior aptidão são dadas maiores oportunidades de reproduzir-se. Isto produz novos indivíduos como "descendentes", que compartilham algumas características tomadas de cada um dos "pais".

Uma nova população inteira de possíveis soluções é então produzida pela seleção dos melhores indivíduos da atual geração, e acasalando-os para produzir um novo conjunto de indivíduos. Esta nova geração contém uma maior proporção das características possuídas pelos bons membros da geração anterior. Deste modo, através de várias gerações, boas características são difundidas através da população. Devido ao favorecimento da seleção de indivíduos mais aptos, uma área mais promissora do espaço de busca é explorada. Se o AG for bem projetado, a população *convergirá* para uma solução ótima do problema.

Em um sentido mais amplo do termo, um algoritmo genético é qualquer modelo baseado em população que usa operadores de seleção e recombinação para gerar novos pontos de amostra em um espaço de busca. Muito modelos de algoritmos genéticos têm sido introduzidos por pesquisadores trabalhando sob uma perspectiva experimental. Muitas destas pesquisas são aplicações orientadas e estão interessadas em algoritmos genéticos tipicamente como método de otimização.

O poder dos AG's vem do fato que a técnica é robusta, e pode tratar satisfatoriamente com uma grande faixa de problemas, incluindo aqueles que são difíceis de serem resolvidos por outros métodos. Os AG's não garantem o encontro da solução ótima global, mas eles são muito bons em encontrar uma resposta "aceitavelmente boa" em um tempo "aceitável". Onde técnicas especializadas existem para

resolução de problemas específicos, estas são preferencialmente usadas por sua precisão e velocidade no resultado final. O principal campo, então, para os AG's, está em áreas difíceis onde nenhuma técnica específica existe. Mesmo onde técnicas existentes trabalham bem, melhorias são alcançadas usando um sistema híbrido com AG's.

Como exemplo de outros métodos de busca, a título de comparação, vamos citar três exemplos. O primeiro é o método de *busca aleatória* onde é utilizado uma abordagem que faz uso da força bruta a fim de encontrar uma solução. Este método varre todo o espaço de busca do problema, de forma sistemática, também atribuindo uma avaliação a cada ponto examinado. Não é uma estratégia muito inteligente, e por isso mesmo, é pouco utilizada. Uma segunda alternativa é o *método do gradiente*, que por meio do gradiente da função a ser examinada, orienta a direção da busca. No entanto este método deve ser utilizado apenas com funções *unimodais* [1], visto que se a derivada da função não pode ser computada, porque é descontínua, este método pode falhar. Um terceiro método combina os dois anteriores e é conhecido como *busca reiterada* (do inglês "iterated search"). Uma vez que um pico foi localizado na função, uma nova busca, a partir de um novo ponto escolhido aleatoriamente, é iniciada. Esta técnica tem como vantagem a simplicidade e funciona bem para funções com poucos máximos ou mínimos locais.

5.2 Questões relativas ao problema

Normalmente existem somente dois componentes principais, na maioria dos AG's, que são dependentes do problema: a codificação e a função de avaliação.

Considere um problema de otimização de parâmetros onde devemos otimizar um conjunto de variáveis qualquer para maximizar algum objetivo tal como lucro, ou minimizar o custo de alguma medida de erro. Podemos visualizar tal problema como uma caixa preta com uma série de controles de ajuste representando diferentes parâmetros; a única saída da caixa preta é o valor retornado por uma função de avaliação indicando o quão bem uma combinação particular de parâmetros resolve o problema de otimização. A meta é fixar os vários parâmetros de forma a otimizar a saída. Em termos mais tradicionais, deseja-se minimizar (ou maximizar) alguma função $F(X_1, X_2, ..., X_M)$ [46].

A primeira suposição normalmente feita é que as variáveis representando parâmetros podem ser representadas por uma seqüência de bits. Isto significa dizer que as variáveis são discretizadas em um momento anterior, e que a faixa de discretização corresponde a alguma potência de 2. Por exemplo,

[1] Funções com apenas um pico

com 10 bits por parâmetro, obtém-se uma faixa com 1024 valores discretos. Se os parâmetros são realmente contínuos então esta discretização não é um problema particular. Isto assume, é claro, que a discretização fornece resolução suficiente para se fazer o ajuste da saída possível com o nível de precisão desejado.

Se algum parâmetro pode somente ser tomado por um conjunto exato de valores finitos então a questão da codificação torna-se mais difícil. Por exemplo, o caso de existirem exatamente 1200 valores discretos que podem ser designados para alguma variável X. Necessita-se de pelo menos 11 bits para cobrir esta faixa de valores, porém isto codifica para um total de 2048 valores discretos. Os 848 valores não necessários podem resultar em nenhuma avaliação, ou alguns parâmetros podem ser representados duas vezes de forma que todas as cadeias de caracteres binárias resultem em um conjunto legal de valores de parâmetros. A resolução de tais problema de codificação é normalmente considerada ser parte do projeto da função de avaliação.

Deixado o assunto de codificação a parte, a função de avaliação é normalmente dada como parte da descrição do problema. Por outro lado, o desenvolvimento de uma função de avaliação pode, algumas vezes, envolver o desenvolvimento de uma simulação. Em outros casos, a avaliação pode ser baseada em performance e pode representar somente uma avaliação aproximada ou parcial. Por exemplo, considere uma aplicação de controle onde o sistema pode ser representado por um número exponencialmente grande de estados possíveis. Assume-se que uma AG é usado para otimizar alguma forma de estratégia de controle. Em tais casos, o espaço de estado deve ser amostrado de um modo limitado e a avaliação resultante das estratégias de controle é aproximada e ruidosa.

A função de avaliação deve ser também relativamente rápida. Isto é tipicamente verdade para qualquer método de otimização, inclusive AG's. Uma vez que os AG's trabalham com uma população de soluções potenciais, isto incorre o custo da avaliação desta população. Além disso, a população é substituída (toda ou em parte) a cada ciclo de geração. Os membros da população reproduzem-se, e seus descendentes devem ser avaliados. Se cada avaliação toma uma hora, então é necessário um ano para realizar-se 10.000 avaliações. Isto corresponderia a aproximadamente 50 gerações para uma população de somente 200 cadeias de caracteres.

5.3 Princípios Básicos

Um AG padrão pode ser representado como na Figura 5.1.

Antes que um AG possa ser executado, uma *codificação* adequada (ou *representação*) para o

problema deve ser formulada. Também exige-se uma *função de avaliação* [2], que designa uma figura de mérito para cada solução codificada. Durante a execução, pais devem ser selecionados para *reprodução*, e recombinados para gerar descendentes. Estes aspectos serão descritos a seguir.

```
BEGIN /* algoritmo genético */
gerar população inicial
computar avaliação de cada indivíduo

WHILE NOT final DO
BEGIN /* produzir nova geração */

FOR I := 0 TO tamanho_população / 2 DO
BEGIN /* ciclo reprodutivo */
selecionar dois indivíduos da geração anterior para reprodução
recombinar os dois indivíduos para gerar dois descendentes
computar avaliação dos dois descendentes
inserir descendentes na nova geração
END

IF população convergiu THEN
final := TRUE

END
END
```

Figura 5.1: Um AG tradicional

5.3.1 Codificação

Assume-se que uma solução potencial para um problema possa ser representada como um conjunto de parâmetros (por exemplo, as dimensões das vigas em um projeto de uma ponte). Estes parâmetros, conhecidos como *genes*, são unidos para formar um cadeia de caracteres, conhecidos como *cromossomos*. Por exemplo, se o nosso problema é maximizar uma função de três variáveis, $F(x, y, z)$, podemos representar cada variável por um número binário de 10 bits. Nosso cromossomo possuiria, então, três genes, consistindo de 30 dígitos binários.

[2]Na literatura, existe um certo desencontro em termos de nomenclatura com relação a função que retorna a avaliação numérica de cada solução, e o seu valor relativo. Alguns autores chamam aquela função de **fitness function** [3]. Estes mesmos autores definem que o número de tentativas de reprodução de um indivíduo em uma geração é baseado no seu **fitness** relativo, que é calculado a partir da função de fitness do indivíduo e da média de fitness da população. Outros autores já definem como função de avaliação numérica de uma solução uma **evaluation function** deixando para o valor relativo de um indivíduo a designação de **fitness function** [46]. Aqui usaremos para o primeiro caso, **função de avaliação** e no segundo caso **avaliação relativa**.

Em termos genéticos, o conjunto de parâmetros representados por um cromossomo particular é denominado *genótipo*. O genótipo contém a informação exigida para construir um organismo – que é denominado *fenótipo*. Os mesmos termos são usados em AG. Por exemplo, na tarefa de construção de uma ponte, o conjunto de parâmetros especificando um projeto particular é o *genótipo*, enquanto a construção finalizada é o *fenótipo*. A avaliação de um indivíduo depende da performance do fenótipo. Isto, em geral, pode ser deduzido a partir do genótipo – isto é, pode ser computado a partir do cromossomo, usando a função de avaliação. Em alguns casos é necessária uma simulação onde o fenótipo pode ser avaliado em relação ao seu comportamento.

5.3.2 Função de Avaliação

Uma função de avaliação deve ser formulada para cada problema a ser resolvido. Dado um cromossomo particular, a função de avaliação retorna um valor numérico simples, que é proporcional a "utilidade" ou "habilidade" do indivíduo que aquele cromossomo representa.

5.3.3 Reprodução

Durante a fase reprodutiva do AG, indivíduos são selecionados da população e recombinados, produzindo descendentes que compreenderão a próxima geração. Pais são selecionados aleatoriamente a partir da população atual usando um esquema que favorece os indivíduos mais aptos. Bons indivíduos provavelmente serão selecionados várias vezes em uma geração, enquanto os indivíduos menos aptos podem nem ser selecionados.

Uma vez selecionados dois pais, seus cromossomos são *recombinados*, usando os mecanismos de *crossover* e *mutação*. As formas mais básicas se apresentam assim:

Crossover – toma dois indivíduos, e corta seus cromossomos em alguma posição aleatória, produzindo dois segmentos em cada um dos cromossomos. O primeiro "pedaço" de cada cromossomo mantém-se, enquanto o segundo pedaço é trocado, gerando dois novos indivíduos, como mostrado na figura 5.2. Os dois descendentes herdam cada um alguns genes dos pais. Este é conhecido como crossover de *ponto simples*. Algumas variações são permitidas.

O crossover não é necessariamente aplicado a todos os pares que foram selecionados para reprodução. Uma escolha aleatória é feita, onde a probabilidade do crossover ocorrer varia normalmente entre 0.6 e 1.0. Se o crossover não ocorre , os descendentes são produzidos apenas pela duplicação dos pais.

Mutação – é aplicada a cada descendente individualmente logo após ao crossover. Aleatoriamente altera-se cada gene com uma pequena probabilidade (normalmente de 0.001). A figura 5.2 apresenta o quinto gene de um cromossomo sofrendo mutação.

A visão tradicional diz que o crossover é a mais importante das duas técnicas uma vez que rapidamente explora um espaço de busca. Já a mutação abrange uma pequena área no espaço de busca, e ajuda a garantir que nenhum ponto tem probabilidade zero de ser examinado, além de evitar convergência para máximos (ou mínimos) locais.

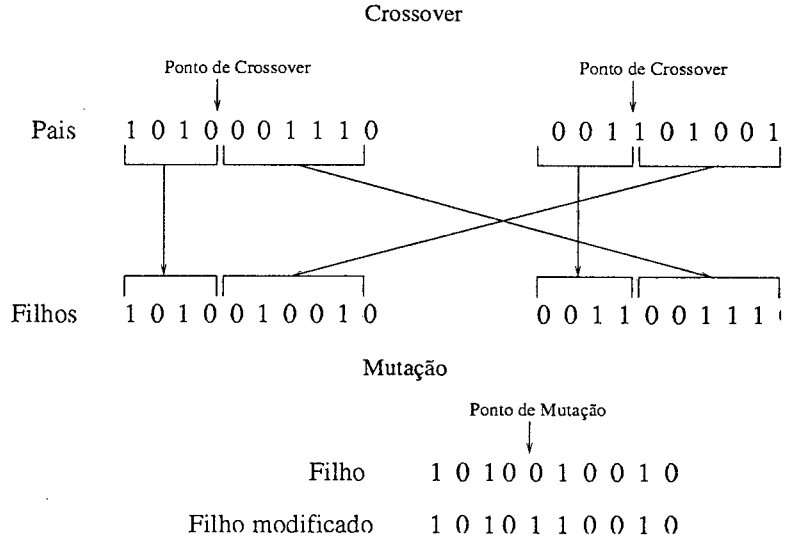

Figura 5.2: Formas de reprodução

5.3.4 Convergência

Se o AG foi corretamente implementado, a população evoluirá através de sucessivas gerações de forma que a avaliação do melhor indivíduo e a média da população crescerá em direção ao ótimo global. *Convergência* é a progressão em direção crescente uniforme. Um *gene* é dito ter convergido quando tipicamente 95% da população compartilha o mesmo valor. A *população* é dita ter convergido quando todos os genes convergiram.

5.4 A teoria por trás dos AG's

A questão que a maioria das pessoas novas no campo dos AG's faria neste ponto é porque eles funcionam. Por que deve-se acreditar que eles nos levarão a um resultado de forma efetiva em problemas

de busca ou otimização?

A resposta mais aceita para explicar o comportamento computacional dos AG's vem de [27]. Neste clássico livro de 1975, Holland desenvolveu vários argumentos projetados para explicar como um "plano genético" ou "algoritmo genético" pode resultar em uma busca complexa e robusta pela amostragem implícita das partições de um hiperplano de um espaço de busca.

Talvez o melhor modo de compreender como um AG amostra partições de um hiperplano é considerar um simples espaço 3-dimensional, como na figura 5.3. Assume-se que temos um problema codificado com apenas 3 bits; isto pode ser representado com um simples cubo com a cadeia de bits 000 na origem. Os cantos deste cubo são numerados por cadeias de bits e todos os cantos adjacentes são etiquetados por cadeias que diferem por exatamente 1 bit. Um exemplo é dado na parte superior esquerda da figura 5.3. O plano frontal do cubo contém todos os pontos que começam com 0. Se o símbolo "*" é usado com um "tanto faz", então este plano pode ser representado pela cadeia de caracteres especial 0**. As cadeias que contêm * são referenciadas como um *conjunto de blocos* (do inglês "schemata"); cada *bloco* (do inglês "schema") corresponde a um hiperplano no espaço de busca. A *ordem* de um hiperplano refere-se ao número de bits de valores reais que aparecem no bloco. Assim, 1** é de ordem 1, enquanto 1**1******0** seria de ordem 3.

A parte inferior da figura 5.3 ilustra um espaço 4-dimensional representado por um cubo suspenso dentro de outro. Os pontos são etiquetados como apresentado a seguir. Etiqueta-se os pontos no cubo interior e no cubo exterior exatamente como feito no espaço 3-dimensional da figura superior. A seguir, prefixa-se cada cadeia do cubo interior com o bit 1 e cada cadeia do cubo exterior com o bit 0. Isto cria uma designação para os pontos no hiperespaço que dá aos pontos adjacentes a diferença de apenas 1 bit entre as cadeias. O cubo interior agora corresponde ao hiperplano 1*** enquanto o cubo exterior corresponde ao hiperplano 0***. É fácil também visualizar que *0** corresponde ao subconjunto de pontos que correspondem aos planos frontais dos cubos. O hiperplano de ordem 2 10** corresponde ao plano frontal do cubo interior.

Uma cadeia de bits iguala um certo conjunto de blocos se esta cadeia pode ser construída a partir deste conjunto de blocos recolocando no local onde encontram-se os símbolos "*" os devidos valores de bits. Em geral, todas as cadeias que igualam um conjunto de blocos em particular estão contidas na partição de hiperplano representado por aquele conjunto particular. Cada codificação é um cromossomo que corresponde a um canto no hipercubo e é membro de $2^L - 1$ hiperplanos diferentes, onde L é o comprimento da codificação binária. (A cadeia onde todos os bits são * corresponde ao próprio espaço e assim não pode ser considerado como partição do próprio espaço [27]). Em outras palavras,

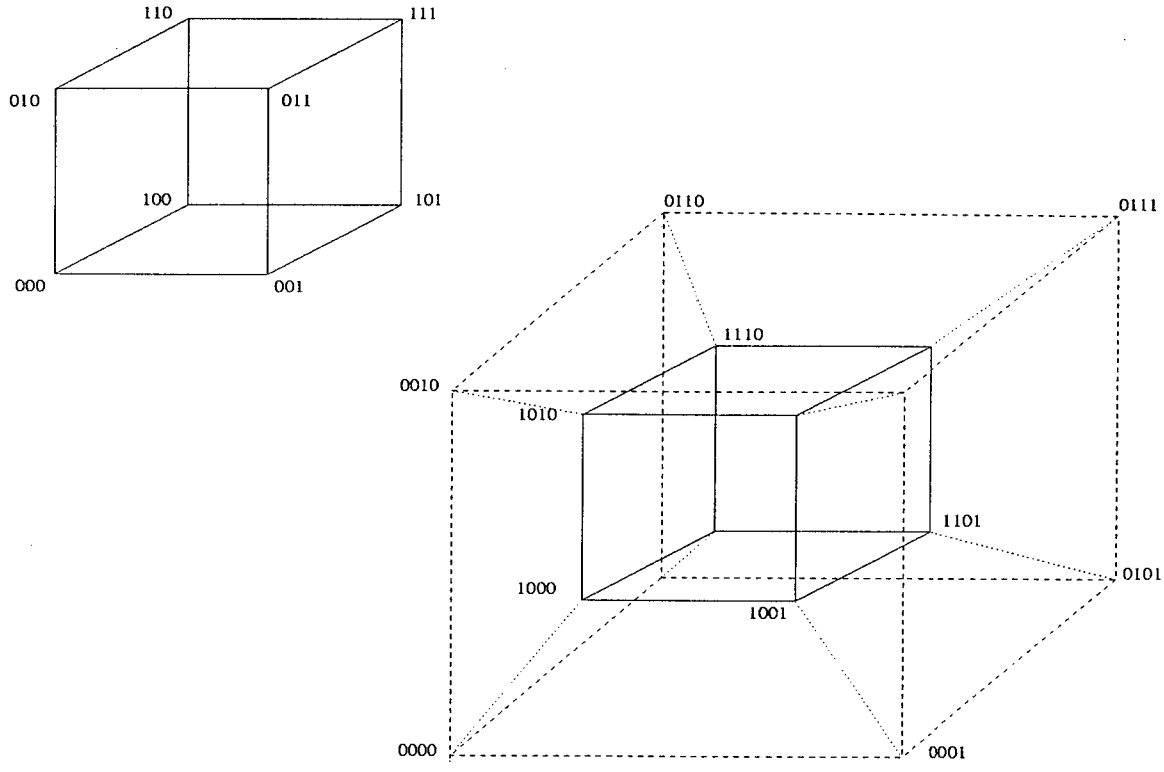

Figura 5.3: Um cubo 3-dimensional e um hipercubo 4-dimensional. Nem todos os pontos são etiquetados no hipercubo 4-D.

existem L posições na cadeias de bits e cada posição pode ser ou um valor ou o símbolo "*".

É também relativamente fácil ver que $3^L - 1$ partições de hiperplano podem ser definidas através do espaço de busca inteiro. Para cada uma das L posições na cadeia podemos ter os valores *, 1, ou 0 que resultam em 3^L combinações.

Afirmar que cada cadeia é membro de $2^L - 1$ partições de hiperplano não fornece muita informação se cada ponto no espaço de busca é examinado de forma isolada. Por isto é que a noção de uma busca baseada em população é crítica para os AG's. Um população de pontos de amostra fornece informação sobre numerosos hiperplanos; além disso, hiperplanos de baixa ordem devem ser amostrados por numerosos pontos na população. Um ponto chave do *paralelismo implícito* ou *intrínseco* dos AG's é derivado do fato que muitos hiperplanos são amostrados quando uma população de cadeia binárias é avaliada; de fato, pode ser argumentado que mais hiperplanos são amostrados que o número de cadeias contidas na população [27]. Vários hiperplanos diferentes são avaliados de forma *implicitamente paralela* a cada vez que uma simples cadeia é avaliada; mas é o efeito acumulativo da avaliação de uma população de pontos que fornece informação estatística sobre qualquer subconjunto

de hiperplanos.

Paralelismo implícito implica que muitas competições de hiperplanos são simultaneamente resolvidas em paralelo. A teoria sugere que através do processo de reprodução e recombinação, o conjunto de blocos de hiperplanos em competição aumenta ou decrementa sua representação na população de acordo com a avaliação relativa das cadeias posicionadas naquelas partições de hiperplano. Uma vez que os AG's operam em populações de soluções, pode-se descobrir a representação proporcional de um simples bloco representando um hiperplano específico em uma população e indicar se aquele hiperplano aumentará ou decrementará sua representação na população através do tempo quando a seleção baseada em avaliação é combinada com crossover para produzir descendentes a partir das cadeias existentes na população.

5.5 Aplicabilidade dos AG's

Assumindo que a interação entre parâmetros é não linear, o tamanho do espaço de busca está relacionado ao número de bits usados na codificação do problema. Para uma cadeia de caracteres binária codificada com comprimento L, o tamanho do espaço de busca é 2^L e forma um hipercubo. O AG amostra os cantos deste hipercubo L-dimensional.

Considere inicialmente uma função qualquer codificada com 30 bits. Qualquer coisa menor que isso representa um espaço que pode ser enumerado. É claro, a expressão 2^L cresce exponencialmente com relação a L. Assim, considere um problema com uma codificação de 400 bits. Qual seria o tamanho deste espaço de busca ? Um livro clássico introdutório em IA dá uma caracterização de um espaço de busca deste tamanho [47]. Winston afirma que 2^{400} é uma boa aproximação para o tamanho efetivo do espaço de busca das possíveis configurações de um tabuleiro de xadrez. Ele chegou a este resultado assumindo um fator de enraizamento (do inglês "branching factor") de 16 para cada movimento possível e assumindo também que um jogo é feito de 100 movimentos, ou seja, $16^{100} = (2^4)^{100} = 2^{400}$. Winston declara que este é um número "ridiculamente" grande. Para se ter uma noção do tamanho deste número, se todos os átomos no universo computassem movimentos de xadrez a taxa de picosegundos desde o big bang (se este realmente ocorreu), hoje a análise estaria apenas começando.

A questão é que enquanto as "boas soluções" para um problema estiverem esparsas com relação ao espaço de busca, uma busca aleatória ou uma busca por enumeração de um grande espaço de busca não será uma forma prática de resolver o problema. Por outro lado, qualquer outra busca diferente de um busca aleatória impõe algum viés em termos de como procura-se por melhores soluções e

onde procura-se no espaço de busca. Os AG's realmente introduzem um viés particular em termos de quais novos pontos serão amostrados no espaço de busca. Não obstante, um AG pertence a classe de métodos conhecidos como *métodos fracos* na comunidade de IA uma vez que um AG faz poucas considerações sobre o problema que ele está resolvendo.

É claro existem muitos métodos de otimização já desenvolvidos pela matemática. Qual o papel que os AG's executam como uma ferramenta de otimização? Eles são muitas vezes descritos como um método de busca global que não usa informação de gradiente. Assim, funções não diferenciáveis bem como funções com múltiplos ótimos locais representam classes de problemas aos quais os AG's podem ser aplicados. Os AG's, como um método fraco, são muito robustos. Se existe um bom método de otimização especializado para um problema específico, então AG pode não ser a melhor ferramenta para esta aplicação. Por outro lado, alguns pesquisadores, trabalham com algoritmos híbridos que combinam os métodos existentes com AG's.

Capítulo 6

Controladores Nebulosos

6.1 Introdução

Convencionou-se chamar de *incerteza* na literatura de sistemas baseados em conhecimento, os problemas ligados a imperfeição da informação. No entanto, o termo incerteza é um tanto restritivo uma vez que uma informação incerta, pode estar tratando de alguma outra imperfeição da informação, como imprecisão, conflito, ignorância parcial, etc ...

Para exemplificar algumas variações em torno de um termo incerto, usaremos como exemplo a informação sobre o horário de início de um filme. Dentre as respostas podemos citar [8]:

- Informação perfeita: O filme começa às 8h 15min.
- Informação imprecisa: O filme começa entre 8h e 9h.
- Informação incerta: Eu acho que o filme começa às 8h (mas não tenho certeza).
- Informação vaga: O filme começa lá pelas 8h.
- Informação probabilista: É provável que o filme comece às 8h.
- Informação possibilista: É possível que o filme comece às 8h.
- Informação inconsistente: Maria disse que o filme começa às 8h, mas João disse que ele começa às 10h.
- Informação incompleta: Eu não sei a que horas começa o filme, mas usualmente os filmes neste cinema começam às 8h.

- Ignorância total: Eu não faço a menor idéia do horário do filme.

Como podemos observar, somos levados a tratar com todo o tipo de informação, desde absolutamente precisa, a completamente imperfeita e vaga. E mesmo assim, o ser humano é capaz de tomar decisões baseado nestes tipos de dados. O desafio aqui, é codificar esta informação em termos computacionais a partir de um modelo de representação de incerteza, para que um sistema baseado em conhecimento tome uma decisão sensata.

Assim, para cada um dos tipos de informação descrito acima, existe um modelo formal conhecido de tratamento. Além disso, encontramos na literatura implementações, formais ou "ad hoc", para cada um desses modelos. A informação de conotação probabilista pode ser tratada pela teoria das probabilidades ou pela teoria da evidência (também conhecida como Dempster-Shafer); a informação possibilista pode ser tratada pela teorias das probabilidades, possibilidades ou evidência, ou por modelos "ad hoc". As informações inconsistentes e aquelas que chamamos, um pouco impropriamente, incompletas, podem ser tratadas por lógicas não clássicas. Para um breve descrição de cada um desse modelos, consulte [8].

Neste capítulo estamos interessados no modelo nebuloso, que lida com a informação do tipo vaga e/ou imprecisa, o qual tem como alicerce a teoria dos conjuntos nebulosos, que pode ser vista como uma generalização da teoria dos conjuntos clássicos. A teoria dos conjuntos nebulosos, quando utilizada em um contexto lógico, como o de sistemas baseados em conhecimento, é conhecida como lógica nebulosa ou lógica difusa (do inglês "fuzzy logic").

A lógica nebulosa é uma das tecnologias atuais mais bem sucedidas para o desenvolvimento de sistemas para controlar processos sofisticados. Com sua utilização, requerimentos complexos podem ser implementados em controladores simples, de fácil manutenção e baixo custo. O uso de sistemas construídos desta maneira, chamados de *controladores nebulosos*, é especialmente interessante quando o modelo matemático está sujeito a incertezas.

Um controlador nebuloso é um sistema nebuloso a base de regras, composto de um conjunto de regras de produção do tipo **Se** < *premissa* > **Então** < *conclusão* >, que definem ações de controle em função de diversas faixas de valores que as variáveis de estado do problema podem assumir. Estas faixas (usualmente mal definidas) de valores são modeladas por conjuntos nebulosos e denominados de termos lingüísticos. No UFSC-Team, os controladores nebulosos são utilizados para implementar os comportamentos reativos do agente, ou seja, os fundamentos necessários para jogar futebol. Para citar alguns dos controladores, temos: *kick_to_goal*, *drive_to_goal*, *catch*, entre outros.

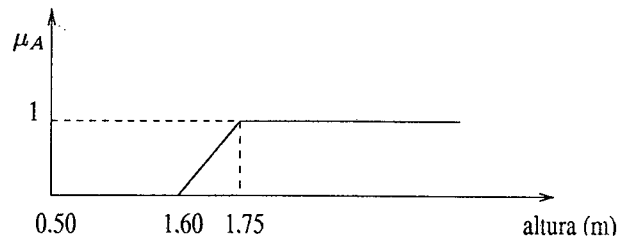

Figura 6.1: Conjunto nebuloso A ≡ "alto"

6.2 Conjuntos Nebulosos

A *teoria dos conjuntos nebulosos* é o modelo mais tradicional para o tratamento da informação imprecisa e vaga. Este modelo, introduzido em [48], tem por objetivo permitir graduações na pertinência de um elemento a uma dada classe, ou seja, de possibilitar a um elemento de pertencer com maior ou menor intensidade àquela classe. Basicamente, isso se faz quando o grau de pertinência de um elemento ao conjunto, que na teoria dos conjuntos "clássica" assume apenas os valores 0 ou 1, passa a ser dado por um valor no intervalo dos números reais $[0, 1]$. Para uma descrição detalhada desta teoria, ver [19] e [20], e para uma descrição mais sucinta, ver [9].

Dado um universo de discurso X, um subconjunto nebuloso A de X é definido por uma função de pertinência que associa a cada elemento x de X o grau $\mu_A(x)$, compreendido entre 0 e 1, com o qual x pertence a A [48]:

$$\mu_A : X \rightarrow [0, 1]$$

Exemplo: Suponhamos que queiramos modelar o conceito "alto". Usualmente, podemos dizer com certeza que uma pessoa que mede mais de $1,75m$ é alta, e que ela não é alta se tiver menos de 1,60m. Já uma pessoa que mede entre $1,60m$ e $1,75m$ será considerada tanto mais alta quanto mais sua altura esteja próxima de $1,75m$. Então podemos modelar o conceito "alto" pelo conjunto nebuloso A, definido no intervalo de $0,5m$ a $2,5m$, dado por:

$$\mu_A(x) = \begin{cases} 1 & x > 1,75m \\ 0 & x < 1,60m \\ \frac{x-1,6}{0,15} & 1,60m \leq x \leq 1,75m \end{cases}$$

A figura 6.1 ilustra o conjunto nebuloso "alto".

Na teoria dos conjuntos nebulosos, a negação $n : [0, 1] \rightarrow [0, 1]$ é implementada por uma família

Tabela 6.1: Principais T-normas e T-conormas duais

T-norma	T-conorma	Nome
$min(x,y)$	$max(x,y)$	Zadeh
$x \cdot y$	$x+y-xy$	Probabilista
$max(x+y-1,0)$	$min(x+y,1)$	Lukasiewicz
$\frac{xy}{\gamma+(1-\gamma)(x+y-xy)}$	$\frac{x+y-xy-(1-\gamma)xy}{1-(1-\gamma)xy}$	Hamacher ($\gamma > 0$)
x, se $y=1$ y, se $x=1$ 0 se não	x, se $y=0$ y, se $x=0$ 1 se não	Weber

de operadores, sendo que o mais comumente utilizado é dado por $n(x) = 1 - x$. Ou seja, se $\overline{A}$ representa a negação de A no universo Y, então utilizando a negação n temos $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$.

Nesta teoria, a intersecção é implementada por uma família de operações – chamadas *T-normas* – e a união é implementada por outra família de operações – chamadas *T-conormas* [20]. O conjunto das T-normas e T-conormas formam as *normas triangulares*. Cada norma triangular $\nu : [0,1] \times [0,1] \to [0,1]$ verifica, para todos os x, y, z em $[0,1]$, as propriedades abaixo:

(i) Comutatividade: $\nu(x,y) = \nu(y,x)$

(ii) Associatividade: $\nu(x,\nu(y,z)) = \nu(\nu(x,y),z)$

(iii) Monotonicidade: se $x \leq z$ e $y \leq t$, então $\nu(x,y) \leq \nu(z,t)$

Além disso, cada T-norma $\top$ verifica a propriedade:

(iv) $\top(x,1) = x$ (elemento neutro 1),

e cada T-conorma $\perp$ verifica a propriedade:

(v) $\perp(x,0) = x$ (elemento neutro 0).

Uma T-norma $\top$ e uma T-conorma $\perp$ são duais em relação a uma operação de negação n se elas satisfazem as relações de De Morgan, isto é, se $n(\top(A,B)) = \perp(n(A),n(B))$ e $n(\perp(A,B)) = \top(n(A),n(B))$. As T-normas e T-conormas duais, em relação à operação de negação $1 - x$, mais utilizadas são apresentadas no quadro 6.1.

Os *operadores de implicação* $I : [0,1] \times [0,1] \to [0,1]$ são usados para modelar regras de inferência do tipo "**Se** <*premissa*> **Então** <*conclusão*>". Seja α o grau de compatibilidade entre

Tabela 6.2: Principais operadores de implicação

Implicação	Nome
$max(1-x,y)$	Kleene
$min(1-x+y,1)$	Lukasiewicz
1, se $x \leq y$ y, se não	Gödel
$min(x,y)$	Mandani
$x \cdot y$	Larsen

as condições estabelecidas na premissa e os valores encontrados na realidade e C, definido em Z, o conjunto nebuloso presente na conclusão da regra. Então, para verificarmos o grau com que a premissa implica a conclusão, dados os valores encontrados na realidade, verificaremos o quanto α implica $\mu_C(z)$, verificando $I(\alpha, \mu_C(z))$ para todo $z \in Z$. O quadro 6.2 traz as principais operações de implicação encontradas na literatura.

6.3 Controladores Nebulosos

A utilização mais significativa da teoria dos conjuntos nebulosos em sistemas baseados em conhecimento são os controladores nebulosos [34], [18]. Um controlador nebuloso pode ser visto como um sistema especialista simplificado, onde a conseqüência de uma regra não é aplicada como antecedente de outra. Isto porque as ações de controle são baseadas em um único nível de inferência. As regras de controle nebuloso são usualmente do tipo

R_j : Se x_1 é A_{1j} e $\cdots$ e x_n é A_{nj} então y é B_j.

onde os A_{ij} e B_j são conjuntos nebulosos, e a implicação é implementada como uma função de implicação nebulosa. Em cada ciclo do processo, os valores das variáveis x_i são medidos e então comparados aos conjuntos nebulosos A_{ij} nas regras, gerando uma medida da adequação dos valores medidos à premissa da regra (utilizando uma T-norma para implementar o conectivo "e" na premissa das regras). A implicação então utiliza esta medida de adequação e o conjunto nebuloso B_j na conclusão para obter um valor B'_j para a variável de controle, em relação àquela regra. Os valores B'_j são então agregados em uma única ação de controle C, utilizando uma T-norma ou T-conorma. Em alguns controladores os B'_j e C são nebulosos, e então é necessário se determinar um valor preciso para a variável de controle, a partir de C.

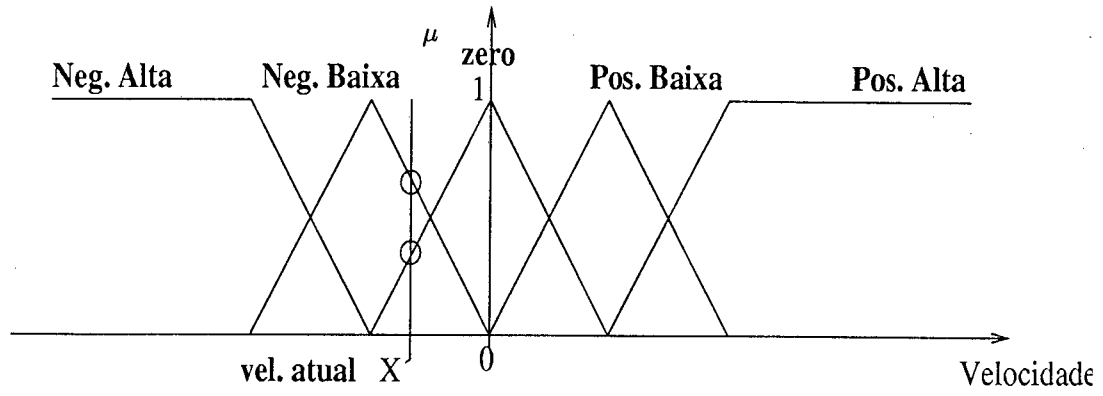

Figura 6.2: Representação da variável lingüística **velocidade**

Ao contrário dos controladores convencionais em que o algoritmo de controle é descrito analiticamente por equações algébricas ou diferenciais, através de um modelo matemático, em controle nebuloso utilizam-se de regras lógicas no algoritmo de controle, com a intenção de descrever numa rotina a experiência humana, intuição e heurística para controlar um processo [48].

Uma *variável lingüística* pode ser definida por uma quádrupla $(X, \Omega, T(X), M)$, onde X é o nome da variável, Ω é o universo de discurso de X, $T(X)$ é um conjunto de nomes para valores de X, e M é uma função que associa uma função de pertinência a cada elemento de $T(X)$. Chamamos de *termos lingüísticos*, indistintamente, tanto os elementos de $T(X)$ quanto suas funções de pertinência [41].

A figura 6.2 exemplifica a variável lingüística velocidade em termos nebulosos dados por $\{$*Negativa Alta, Negativa Baixa, Zero, Positiva Baixa, Positiva Alta*$\}$.

O grau com que um valor x^* em Ω satisfaz o termo lingüístico A é a pertinência de x^* em A, dada por $\mu_A(x^*)$.

Os controladores nebulosos são robustos e de grande adaptabilidade, incorporando conhecimento que outros sistemas nem sempre conseguem acomodar. Também são versáteis, principalmente quando o modelo físico é complexo e de difícil representação matemática.

Na sua maioria, os controladores nebulosos encontram aplicações em sistemas não-lineares, sendo até mesmo capaz de adaptar-se a plantas com elevado nível de ruído e sujeitas a perturbações. Conseguem, mesmo em sistemas onde existe algum grau de incerteza, agregar uma robustez característica. Porém, existem dificuldades em provar-se algumas características de robustez neste tipo de abordagem.

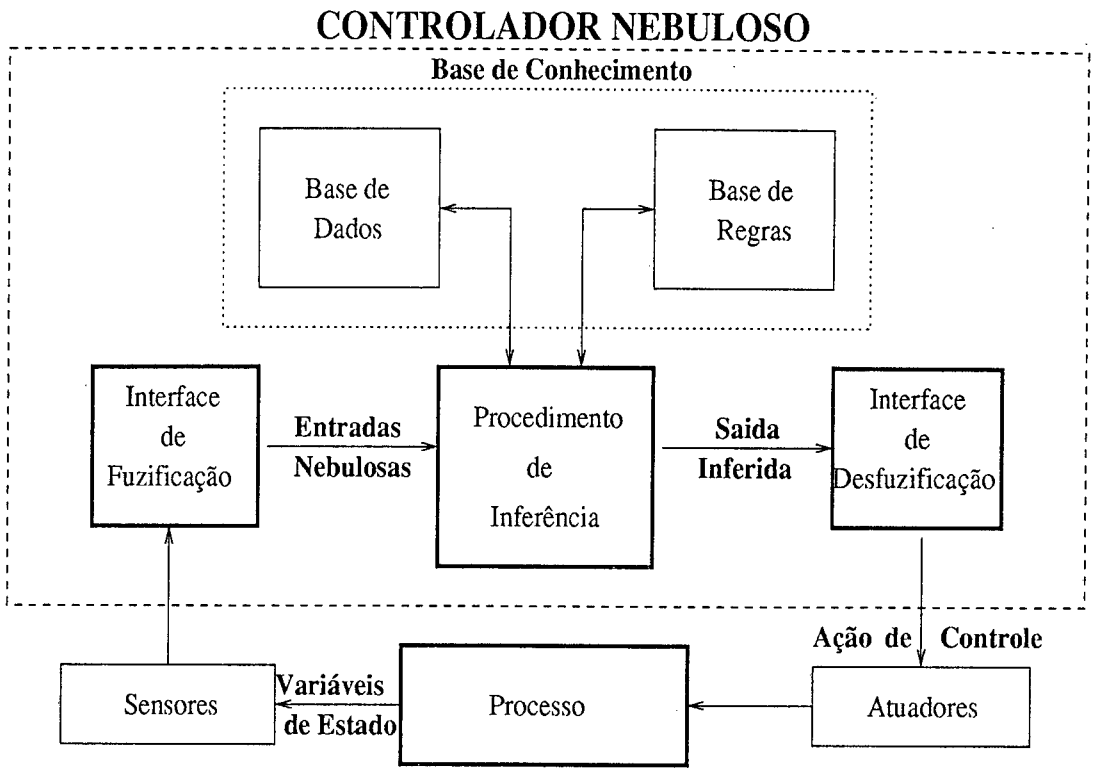

Figura 6.3: Estrutura de um Controlador Nebuloso

A figura 6.3 apresenta a estrutura básica de um controlador nebuloso apresentando seus módulos principais. Algumas variações são encontradas na literatura que adaptam-se segundo as necessidades do projeto, no entanto o modelo apresentado é suficiente para fornecer a idéia do fluxo de informação normal dentro do controlador.

6.3.1 Interface de "Fuzificação

A *interface de "fuzificação"* faz a identificação dos valores das variáveis de entrada, as quais caracterizam o estado do sistema (variáveis de estado), e as normaliza em um universo de discurso padronizado. Estes valores são então "fuzificados", de forma que possam ser instanciados em variáveis lingüísticas.

6.3.2 Base de Conhecimento

A *base de conhecimento*, assim com em qualquer sistema baseado em conhecimento, é formado por uma base de regras e uma base de dados, que caracteriza a estratégia de controle aplicada bem como seus objetivos.

Na *base de dados* ficam armazenadas as definições sobre discretização e normalização dos universos de discurso, e as definições das funções de pertinência dos termos nebulosos.

A *base de regras* é formada por estruturas do tipo

Se <*premissa*> **Então** <*conclusão*>

como por exemplo:

Se *Erro é Grande* **e** Δ*Erro é Positivo* **Então** *Velocidade é Positiva Pequena*

Um *procedimento de inferência* avalia os dados de entrada baseado nas regras da base de conhecimento inferindo ações de controle segundo o estado do sistema, aplicando o operador de implicação, de acordo com um dos procedimentos de inferência existentes na literatura [11].

É importante que exista um total mapeamento das combinações possíveis para os valores das variáveis de entrada construindo tantas regras quanto forem necessárias garantindo uma base completa, ou seja, garantir que sempre exista ao menos uma regra a ser disparada para qualquer entrada. É vital também garantir a consistência da base de regras, evitando contradições e interações entre elas.

6.3.3 Procedimento de Inferência

Um controlador nebuloso pode ser visto como um sistema especialista onde a conseqüência de uma regra não é aplicada como antecedente de outra [18]. Assim, o processo de inferência consiste em [41]:

1. Verificação do grau de compatibilidade entre os fatos e as cláusulas nas premissas das regras;
2. Determinação do grau de compatibilidade global da premissa de cada regra;
3. Determinação do valor da conclusão, em função do grau de compatibilidade da regra com os dados e a ação de controle constante na conclusão (precisa ou não);
4. Agregação dos valores obtidos como conclusão nas várias regras, obtendo-se uma ação de controle global.

Pode-se dividir os modelos de controladores nebulosos encontrados na literaturas em modelos clássicos, compreendendo o modelo de Mandani e o de Larsen, e os modelos de interpolação, compreendendo o modelo de Takagi-Sugeno e o de Tsukamoto [18] [34].

Os modelos diferem quanto à representação dos termos da premissa, quanto a representação das ações de controle e quanto aos operadores utilizados para implementação do controlador.

Nos modelos clássicos, a conclusão de cada regra especifica um termo nebuloso dentre um conjunto fixo de termos (geralmente em número menor que o número de regras). Estes termos são usualmente conjuntos nebulosos convexos como triângulos, funções em forma de sino e trapézios.

Dado um conjunto de valores para as variáveis de estado, o sistema obtém um conjunto nebuloso (muitas vezes sub-normalizado), como o valor da variável de controle. Este conjunto nebuloso representa uma ordenação no conjunto de ações de controle aceitáveis naquele momento. Finalmente, uma ação de controle global é selecionada dentre aquelas aceitáveis em um processo conhecido como defuzificação.

Nos modelos de interpolação, cada conclusão é dada através de uma função estritamente monotônica, normalmente diferente para cada regra. No modelo Takagi-Sugeno, a função é uma combinação linear das entradas, tendo como parâmetros um conjunto de constantes. No esquema de Tsukamoto, a função é geralmente não linear, tendo como domínio os possíveis graus de compatibilidade entre cada premissa e as entradas.

Em ambos os esquemas, obtém-se, para cada regra, um único valor para a variável de controle. Finalmente, uma ação de controle global é obtida fazendo-se uma média ponderada dos valores individuais obtidos (os pesos são dados pelo próprio grau de compatibilidade entre a premissa da regra e as entradas, normalizadas).

6.3.4 Interface de "Defuzificação"

No conjuntos nebulosos clássicos, a interface de "defuzificação"é utilizada para obter uma única ação de controle precisa. O procedimento compreende a identificação do domínio das variáveis de saída num correspondente universo de discurso e com a ação de controle nebulosa inferida evolui-se uma ação de controle não nebulosa.

Dentre os métodos de "desfuzificação" mais utilizados estão:

1. Primeiro Máximo (SOM): Encontra o valor de saída através do ponto em que o grau de pertinência da distribuição da ação de controle atinge o primeiro valor máximo;

2. Método da Média dos Máximos (MOM): Encontra o ponto médio entre os valores que têm o maior grau de pertinência inferido pelas regras;

3. Método do Centro de Área (COA): O valor da saída é o centro de gravidade da função de

distribuição de possibilidade da ação de controle.

O método deve ser escolhido seguindo critérios que devem levar em consideração as características do processo controlado e o comportamento de controle necessário. Para citar um exemplo, os métodos dos últimos máximos ou média dos máximos, cujo efeito se assemelham ao de um controlador "bang-bang", podem conduzir à ações de controle inadequadas (solavancos) ao modo de operação e assim, causar danos de ordem prática em equipamentos como atuadores.

Estes não são os únicos métodos existentes na literatura, mas a diferença com relação aos apresentados dá-se com relação a velocidade e eficiência, características que devem ser analisadas em conjunto com os requisitos de projeto.

Uma vez que os controladores nebulosos do tipo interpolação já apresentam diretamente os valores precisos para as entradas do processo controlado, estes não necessitam de um interface de "desfuzificação".

Capítulo 7

O Agente UFSC-Team

7.1 Introdução

Em sua primeira participação na liga de simuladores da RoboCup 98, o UFSC-Team apresentou uma arquitetura multiagente cognitiva concorrente [16]. A idéia era implementar percepção, ação, comunicação, cooperação, planejamento e tomada de decisão explorando a abordagem de programação concorrente [1].

Esta primeira arquitetura concorrente era baseada em três processos: *Interface*, *Coordinator* e *Expert*. O processo *Interface* foi projetado para manipular percepção e ação. Como já apresentado no Capítulo 3, a interação ambiente/agente é suportada pelo Soccerserver, consistindo de trocas de mensagens usando um canal Inet Domain Socket. A informação de percepção é recebida e os comandos de ação são enviados através deste canal. A função do processo Interface era apenas traduzir as informações de percepção e comunicação para a linguagem Parla [13] (a linguagem de comunicação de agentes usada pelos agentes do UFSC-Team) e expressões da linguagem Parla para comandos do Soccerserver.

O processo *Coordinator* era responsável pela comunicação do agente e por começar e conduzir o processo de cooperação. De acordo com a arquitetura original proposta no ambiente Expert-Coop [15], este processo seria responsável pelo gerenciamento da comunicação interprocessos, isto é, ele receberia diretamente as mensagens enviadas pelos outros agentes e as manipularia. Porém, de acordo com as regras da liga de simuladores da RoboCup, toda a comunicação interagentes deve ser feita pelo Soccerserver. Devido a isto, a comunicação interprocessos e a informação de percepção são todas recebidas através do mesmo socket. Sendo assim, nesta implementação, o processo *Interface* também recebia as mensagens interagentes e as repassava para serem manipuladas pelo processo *Coordinator*,

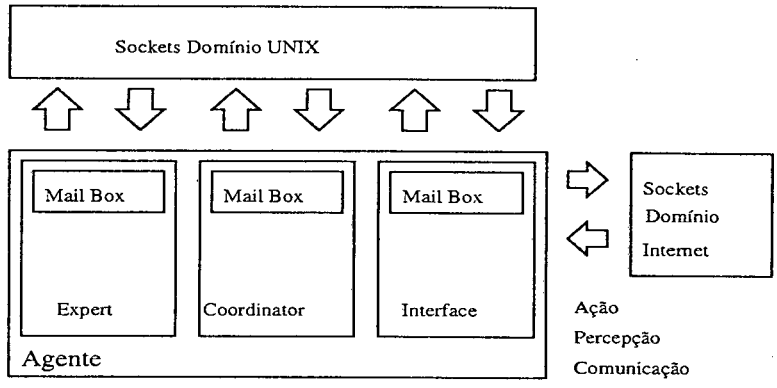

Figura 7.1: Arquitetura Concorrente

juntamente com a informação de percepção e as mensagens do juiz.

Finalmente, o processo *Expert* era responsável pelo planejamento e tomada de decisão. Este, possuia um sistema baseado em conhecimento encapsulado onde a informação de percepção, as mensagens do juiz e de outros agentes do UFSC-Team eram armazenadas e usadas para inferir decisões, de acordo com as regras do sistema baseado em conhecimento. Estes três processos comunicavam entre si por troca de mensagens usando sockets dentro do domínio Unix, como mostrado na figura 7.1.

A primeira implementação, com uma abordagem por decisão centralizada, apresentou alguns problemas com relação a sincronização agente/ambiente e o tempo de resposta era considerado muito alto. De fato, as melhores respostas tempo-real apresentadas pela arquitetura do agente estavam entre 70 e 80ms, mesmo usando raciocínio baseado em caso (do inglês "Case-Based Reasoning") para dividir o conhecimento em diferentes pacotes. Além disto, o sistema baseado em conhecimento responsável pelas decisões tornou-se muito complexo, um vez que incluia regras para tratar informação de alto nível, como que tipo de jogada coletiva deve ser escolhida em uma dada situação, até baixo nível, do tipo qual valor para potência de arranque ou que valor para potência de chute devem ser escolhidos.

Para resolver estes problemas, a arquitetura de agente usada no UFSC-Team migrou de uma abordagem concorrente com tomada de decisão centralizada, para uma arquitetura de agente autônomo, inspirada na arquitetura proposta em [7], com três níveis de decisão - reativo, instintivo e cognitivo - implementados de forma concorrente, como mostrado na figura 7.2. O modelo concorrente manteve-se com os mesmos três processos: *Interface*, *Coordinator* e *Expert*. Porém agora, cada um destes processos encapsula um motor de inferência diferente e é responsável por um dos três níveis de de-

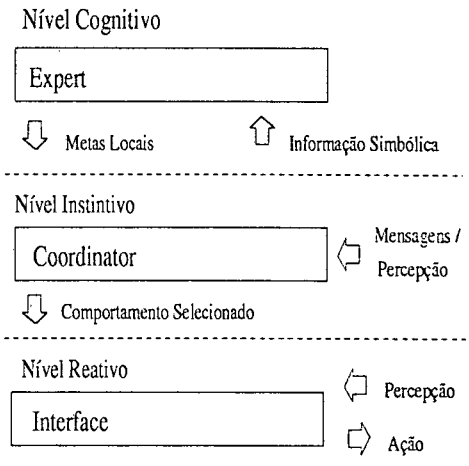

Figura 7.2: O fluxo de informação no agente

cisão. Tanto a primeira implementação como a atual foram escritas em linguagem de programação C++ e integram uma implementação parcial do ambiente para desenvolvimento de sistemas multiagentes cognitivos sob restrições tempo-real chamado Expert-Coop++.

O motor de inferência do nível reativo é implementado no processo *Interface* e é responsável pela resposta tempo real do agente, isto é, por receber a informação de percepção do Soccerserver e por enviar os comandos de ação apropriados para ele. Este processo consiste de um conjunto de controladores nebulosos. Em um dado momento, somente um controlador está ativo e ele decide que comandos devem ser enviados para o Soccerserver, juntamente com seus respectivos valores. Esta escolha é baseada na informação recebida do Soccerserver e é determinada pelas regras do controlador nebuloso ativo. Cada um dos controladores nebulosos disponíveis no agente representa um *comportamento* específico e tem algumas condições associadas que especificam as situações em que ele é efetivo.

O motor de inferência do nível instintivo é implementado no processo *Coordinator* e é responsável por atualizar as variáveis simbólicas usadas pelo nível cognitivo e por escolher os comportamento adequados, isto é, os controladores nebulosos adequados, que devem ser usados no nível reativo a fim de alcançar uma dada *meta*. Uma meta pode ser alcançada por uma seqüência de comportamentos reativos que levam o agente a uma situação intencionada. A escolha desta seqüência de comportamentos é implementada através de um sistema especialista de um ciclo de inferência que escolhe, a cada vez que o *estado* da partida muda, o comportamento reativo mais adequado. Cada estado do jogo é definido por um conjunto de condições que são monitoradas pelo nível instintivo. Estas condições referem-se a percepção e as mensagens do juiz, e são usadas na condição das regras, analogamente

ao nível reativo. Mas no nível instintivo, a conclusão das regras são simbólicas e são usadas ou para atualizar a informação simbólica usada no nível cognitivo, ou para selecionar um comportamento reativo. A cada momento, o comportamento escolhido deve executar ações no sentido de alcançar a meta definida e deve ter suas condições associadas satisfeitas pelo estado do jogo. Uma vez que um comportamento é escolhido, o nível instintivo mantém-se monitorando as condições associadas a este comportamento e, se alguma delas não é mais satisfeita, ele usa suas regras para inferir um novo comportamento. Se isto é impossível, a meta falha e uma nova meta deve ser especificada pelo nível cognitivo. O nível instintivo também manipula as mensagens enviadas pelo juiz informando uma mudança no "status" do jogo. Estas mudanças são tratadas analogamente às mudanças no estado do jogo, ou seja, elas também determinam ao nível instintivo a escolha de um novo comportamento apropriado.

Finalmente, o motor de inferência do nível cognitivo é implementado no processo Expert, e é responsável por determinar as metas locais e globais do agente. O nível cognitivo não tem uma atuação direta sobre o nível reativo, ele apenas escolhe a atual meta e a passa para o nível instintivo. Isto causa o efeito de modificar as regras do motor de inferência no nível instintivo, que indiretamente causará comportamentos diferentes a serem selecionados. Enquanto uma meta não falha ou obtém sucesso, o nível cognitivo não interfere no jogo. Este tempo ocioso é usado para planejamento estratégico. Este planejamento consiste na determinação de possíveis metas locais futuras, de acordo com o resultado da atual, e na especificação de requisições de cooperação para alcançar metas globais. Estas requisições serão manipuladas pelo processo *Coordinator* e resultará em outros agentes adotando metas locais compatíveis com a meta global intencionada. O nível cognitivo é também implementado através de um sistema especialista, mas este sistema especialista pode ser muito mais complexo que aquele do nível instintivo, uma vez que seu tempo de resposta é muito maior.

Na nova implementação, os três processos são implementados usando a abordagem de programação multi-thread [37]. Esta tecnologia permite dividir um processo em partes e rodar estas partes concorrentemente. No nosso caso, cada processo consiste de dois threads. O primeiro é responsável pela manipulação da interrupção Unix SIGIO, usada para informar que uma nova mensagem foi recebida pelo socket, e por colocar esta mensagem dentro do mailbox. O outro thread, o thread principal, é responsável pelas atividades do processo. A exclusão mútua entre os threads é alcançada pelo uso de semáforos. Esta implementação é uma abordagem concorrente para o problema clássico produtor/consumidor. Isto evita que o processo principal gaste algum tempo precioso checando se existe uma nova mensagem no socket ou não.

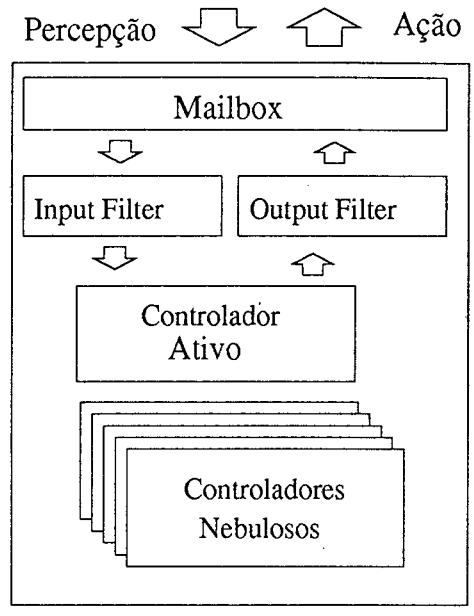

Figura 7.3: O processo Interface

Este capítulo, nas suas próximas seções, tem por objetivo uma descrição mais detalhada de cada um dos níveis citados acima, finalizando com a apresentação de um exemplo desta arquitetura onde permite-se que o agente reaja concorrentemente a um estímulo do ambiente em tempo real e execute uma tarefa mais sofisticada tal como planejar, estabelecer novas metas, abrir ou participar de um processo de cooperação.

7.2 O nível reativo

O motor de inferência do nível reativo é implementado no processo *Interface*. Este processo consiste de um mailbox, um conjunto de controladores nebulosos, um filtro de entrada e um filtro de saída, como mostrado na figura 7.3. O mailbox é responsável pelo processo de recepção de mensagens. Todas as mensagens recebidas pelo processo, incluindo a informação de percepção enviada pelo Soccerserver, será armazenada no mailbox.

Os controladores nebulosos foram implementados usando uma biblioteca C++. Esta biblioteca foi projetada para auxiliar na implementação de sistemas especialistas nebulosos ou na implementação de controladores nebulosos, chamada CNCL [29]. Cada controlador nebuloso é responsável por uma habilidade reativa do agente, chamada *comportamento*. Primeiramente, os seguintes conjuntos de comportamentos foram escolhidos para serem implementados dentro do agente UFSC-

Team: *Initialize-Player*, *Kick-Off-Position*, *Move-to-Position*, *Move-to-Ball*, *Pass-Ball*, *Kick-to-Goal*, *Dribble-Opponent*, *Drive-Ball-Fwd*, *Get-Ball-Control*, *Tackle*, *Follow-Opponent*, *Rounding-Opponent*, *Watch-Ball* and *Catch-Ball*. O conjunto de controladores nebulosos associados a cada agente depende do grupo de agentes ao qual ele pertence: goleiro, jogadores defensivos, meio-campos ou jogadores de ataque. É claro, não faz sentido para um atacante ou meia ter um controlador nebuloso responsável por defender a bola, ou para um goleiro ter um controlador nebuloso responsável por chutar no gol adversário.

O filtro de entrada é responsável por extrair os valores das variáveis lingüísticas, usadas pelo controlador nebuloso ativo, a partir da informação enviada pelo Soccerserver. O filtro de saída é responsável por averiguar as saídas do controlador nebuloso ativo e combiná-las. Os seguintes critérios são observados pelo filtro de saída:

- **Saída Nula**: se a saída do comando `dash` e/ou a saída do comando `turn` apresentar valor nulo, o respectivo comando não é enviado ao Soccerserver.
- **Comandos `turn` e `dash` simultâneos**: se o controlador nebuloso apresenta saída para os comandos `turn` e `dash` simultaneamente, então primeiro o comando `turn` é enviado ao Soccerserver e depois de 20 ms é enviado o comando *dash*.
- **Direção do chute e potência do chute**: os valores para potência e direção de chute são enviados juntos no comando `kick`.

A maioria dos controladores nebulosos apresentam quatro saídas: *kick_direction*, *kick_power*, *turn_moment* e *dash_power*. Os controladores nebulosos *Pass-Ball* e *Kick-to-Goal* têm apenas as saídas *kick_direction* e *kick_power* e o controlador *Move-to-Position* tem apenas as saídas *turn_moment* e *dash_power*. As entradas são um conjunto de variáveis lingüísticas, dependendo de qual comportamento está ativo. Cada controlador tem seu próprio conjunto de variáveis lingüísticas e o filtro de entrada (na figura 7.3 Input Filter) é responsável por extrair da informação de percepção, os respectivos valores que serão utilizados nas variáveis lingüísticas.

O uso de controladores nebulosos para implementar o nível reativo apresentou algumas vantagens. Primeiramente, é possível sincronizar o agente apenas ajustando a taxa entre entrada e saída, ou em outras palavras, ajustando o ganho do controlador. Este ajuste de ganho é feito no conjunto nebuloso que representa a entrada e a saída do controlador. É possível também um ajuste fino, ou até mesmo uma resposta mais suave destes conjuntos nebulosos. A figura 7.4 apresenta o conjunto nebuloso

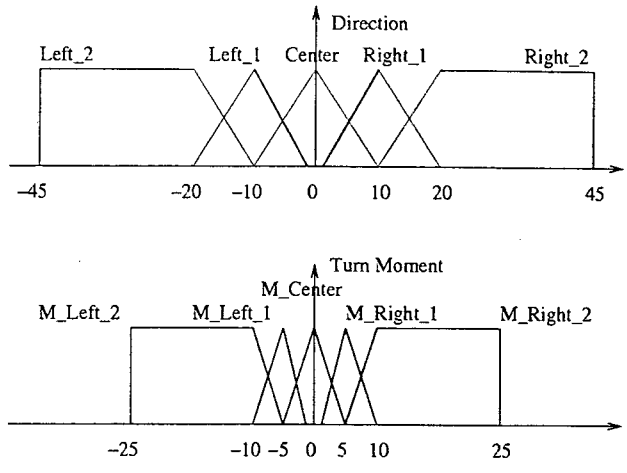

Figura 7.4: Conjuntos Nebulosos

usado pela saída *turn_moment* e a respectiva variável lingüística *ball_direction*. Note que neste caso o ganho do controlador é $0.56 = \frac{50}{90}$.

As regras usadas nos controladores nebulosos podem ser construídas de maneira intuitiva, evitando a dificuldade e o consumo de tempo na tarefa de construir um modelo de ambiente dinâmico. É também possível o uso de algoritmos genéticos para melhorar os conjuntos nebulosos usados nos controladores, sendo esta última proposta explorada neste trabalho, com os seus resultados apresentados no próximo capítulo.

Outra vantagem importante de implementar o comportamento reativo usando controladores nebulosos é que é possível garantir que um dado controlador será sempre capaz de satisfazer as exigências de tempo real, uma vez que um controlador nebuloso é um sistema determinístico. Além disso, uma vez que o controlador ativo é o comportamento mais apropriado em uma dada situação, ele permite que os níveis instintivo e cognitivo gastem mais tempo em tarefas mais sofisticadas como extrair características simbólicas interessantes de percepção, estabelecer planejamentos e metas ou de participar de processos de cooperação.

7.3 O nível instintivo

O motor de inferência do nível instintivo é implementado no processo *Coordinator* e é responsável tanto pela execução das metas locais do agente como pela geração da informação simbólica que atualiza a base de conhecimento do nível cognitivo. É implementado através de um sistema especialista de um ciclo de inferência que escolhe, a cada vez que o *estado* do jogo muda, o comportamento reativo

mais adequado dada a atual meta local. A meta local atual é estabelecida pelo nível cognitivo e determina o conjunto de regras a serem usadas no motor de inferência. Cada estado do jogo é definido por um conjunto de condições sobre a informação de percepção. Estas condições normalmente dependem de alguns valores limiares, que devem ser determinados experimentalmente.

As entradas do motor de inferência do nível instintivo são a informação de percepção, recebida do processo *Interface*, e as mensagens do juiz. A informação de percepção consiste da mesma percepção recebida pelo processo *Interface* oriunda do Soccerserver, mas, diferentemente do nível reativo, o nível instintivo apresenta uma memória. Esta memória consiste de uma lista, onde esta percepção é armazenada, e cujo tamanho inicial é um parâmetro da implementação. Isto torna possível escolher quantos quadros de informação visual pode ser usado em um ciclo de inferência do motor de inferência. Por exemplo, assumindo que o agente recebe informação visual a cada 150ms e o tamanho da lista é 3, em um dado tempo t, o ciclo de inferência do processo levará em conta a informação visual enviada nos tempos $t, t-150, t-300$.

A informação de percepção é armazenada em uma lista síncrona e as mensagens do juiz são armazenadas em uma lista assíncrona, como mostrado na figura 7.5. A cada vez que uma destas listas é atualizada ou quando uma nova meta local foi recebida a partir do nível cognitivo, o sistema especialista é executado. Dada uma entrada, as regras são capazes de reconhecer mudanças no estado do jogo. O resultado da execução das regras pode ser ou a atualização da base de conhecimento do nível cognitivo ou a seleção do controlador nebuloso do nível reativo mais adequado a fim de dirigir o agente do estado atual até a meta local.

Uma das contribuições deste trabalho foi a implementação do nível instintivo. Foram desenvolvidas as bibliotecas necessárias para a construção do sistema especialista acima descrito, usando uma representação de conhecimento baseada em quadros. Todos as classes desenvolvidas foram incorporadas a biblioteca Expert-Coop++.

7.4 O nível cognitivo

O motor de inferência do nível cognitivo é implementado no processo *Expert*. Ele consiste de um sistema baseado em conhecimento simbólico orientado a objeto que manipula tanto a informação simbólica vinda do nível instintivo como as mensagens assíncronas recebida por outros agentes do UFSC-Team. Ele gera as metas locais e globais. Este sistema baseado em conhecimento apresenta três bases de conhecimento: *Dynamic KB*, *Static KB* e *Export KB*.

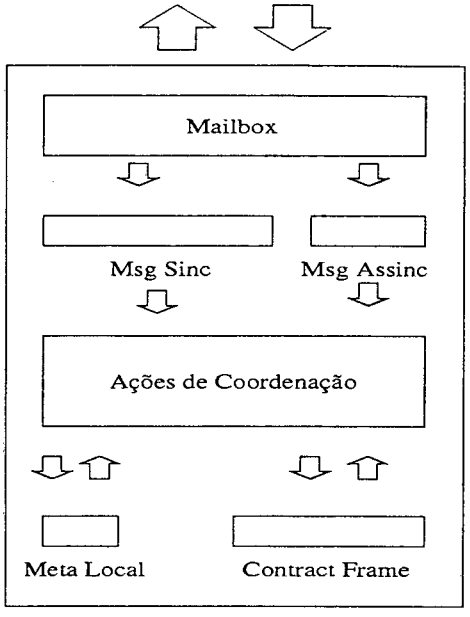

Figura 7.5: O processo Coordinator

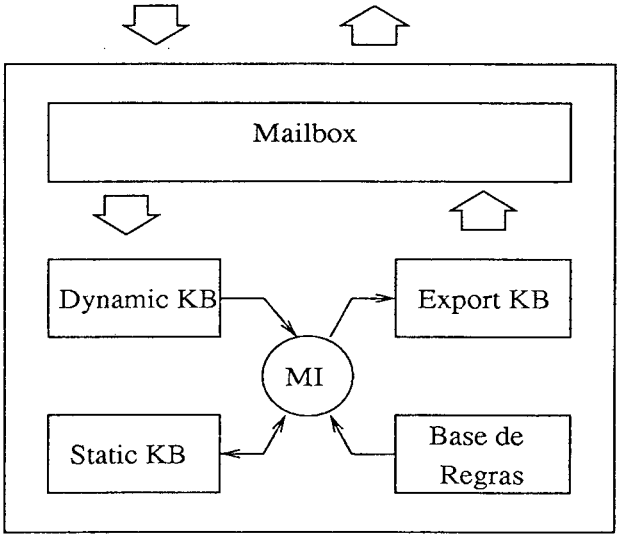

Figura 7.6: O processo Expert

Dynamic KB é usada para armazenar a informação simbólica gerada pelo nível instintivo e as mensagens assíncronas enviadas por outros agentes do UFSC-Team. A *Static KB* armazena o conhecimento que foi inferido pelo processo *Expert* sobre o jogo, o time, o oponente, os planos do agente, metas, etc. Ambas, *Dynamic KB* e *Static KB*, são usadas pelo motor de inferência para gerar as metas dos agentes. Os novos fatos sobre o jogo, os planos e metas são armazenados dentro de *Static KB*. *Export KB* é usada para armazenar a saída do processo Expert. Basicamente esta saída consiste de metas locais a serem enviadas ao nível instintivo, informação para ser usada em estratégias de cooperação, ou mensagens a serem enviadas a outros agentes do UFSC-Team.

Supomos que alguma informação simbólica e/ou mensagens tenham sido recebidas do processo *Coordinator*, seguido por uma requisição. Isto causa a seguinte seqüência de ações:

1. A informação é armazenada em *Dynamic KB*.
2. O motor de inferência avalia ambas informações armazenadas em *Dynamic KB* e os fatos armazenados em *Static KB*.
3. Os novos fatos, planos e metas gerados, são armazenados dentro de *Static KB*.
4. Se uma nova meta é escolhida e/ou existem algumas informações a serem enviadas para outro agente, são armazenadas em *Export KB*.
5. Se *Export KB* não está vazia, seu conteúdo é enviado ao processo Coordinator.
6. *Dynamic KB* e *Export KB* são esvaziadas.
7. Uma resposta é enviada ao processo *Coordinator*.

Uma importante característica nesta nova arquitetura é que o nível cognitivo dispõe de mais tempo para planejamento, estabelecimento de metas, etc, uma vez que o nível reativo e, em algumas situações, o nível instintivo, são responsáveis pela interação tempo real com o ambiente.

7.5 Um exemplo

Um exemplo onde a proposta da arquitetura de agente autônomo concorrente pode ser útil é apresentado nesta seção. Vamos assumir uma situação onde o time adversário tem a posse de bola e está executando uma jogada ofensiva tentando passar a bola. Nesta situação o jogador perto do adversário, que tem a posse de bola, terá como meta local *get-ball-control-back* (melhor dizendo,

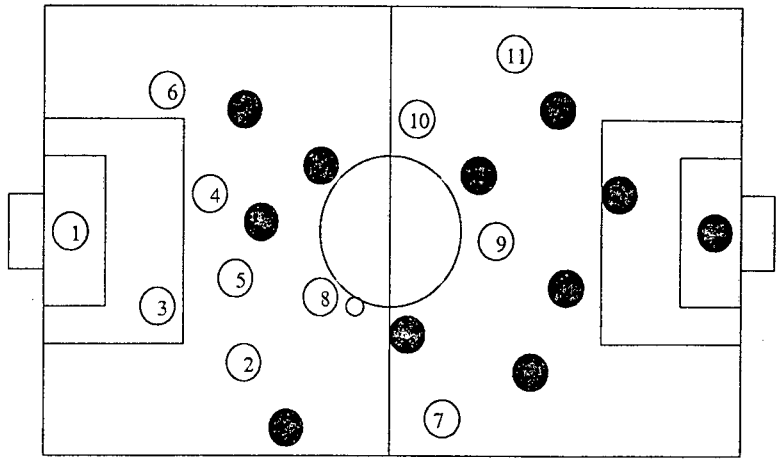

Figura 7.7: Um exemplo

"tomar de volta a bola"), e estará executando o comportamento reativo *Rounding-Opponent*. Supomos que o oponente ao tentar passar a bola cometeu um erro, e o meio campo número 8, toma o controle da bola, como mostrado na figura 7.7. Nesta situação, o nível instintivo do agente número 8 reconhecerá que a meta foi alcançada e mudará o atual comportamento para *Drive-Ball-Fwd* e informará o nível cognitivo do novo estado do jogo. Então, o processo Interface executará um novo comportamento e o motor de inferência do nível cognitivo escolherá uma das jogadas padrões armazenadas para ser executada. Isto significa selecionar uma meta global a ser alcançada, difundir esta meta aos agentes envolvidos e enviar uma meta local, relacionada com a meta global selecionada para ser executada pelo nível instintivo.

Capítulo 8

Otimização de variáveis lingüísticas de Controladores Nebulosos

8.1 Introdução

Como já apresentado no Capítulo 7, o nível reativo é composto por alguns controladores nebulosos, que implementam cada um dos fundamentos necessários para a prática de uma partida de futebol. As regras usadas nos controladores nebulosos podem ser construídas de maneira intuitiva, evitando a necessidade da construção de um modelo dinâmico do ambiente, característica esta que diferencia este tipo de controle do controle clássico, onde uma descrição matemática faz-se necessária.

Embora a idéia de um controlador nebuloso seja muito atrativa, o projeto de uma base de regras apropriada e a especificação dos conjuntos nebulosos necessários para um controle ótimo do processo é uma tarefa difícil e bastante tediosa, tomando um tempo valioso de projeto. A automatização, ao menos parcial, destas tarefas é considerada importante e um grande número de abordagens envolvendo redes neurais já foram proposta [5] [33]. Atualmente algoritmos genéticos (AG) são também considerados como uma possível solução para o problema de projeto e otimização de controladores nebulosos [30].

Neste capítulo é apresentado a descrição de um AG que tem por objetivo a otimização dos valores dos conjuntos nebulosos existentes nos controladores implementados no nível reativo do agente proposto em [17]. As próximas seções trazem uma descrição de um dos controladores presentes no nível reativo, uma pequena descrição da biblioteca C++ utilizada para a construção dos AG's, a apresentação da função de avaliação utilizada, e finalizando, uma comparação entre os resultados obtidos com os controladores projetados "manualmente" e os controladores projetados com o uso desta

forma de computação evolutiva.

8.2 Controladores Nebulosos do Agente UFSC-Team

Os controladores nebulosos, presentes no nível reativo do agente UFSC-Team, foram projetados e desenvolvidos com o auxílio de uma biblioteca C++, chamada CNCL [29], que auxilia na programação de sistemas especialistas nebulosos. Estão disponíveis na biblioteca conjuntos nebulosos, variáveis nebulosas, e um motor de inferência baseado em regras nebulosas.

Cada um dos controladores é responsável por um comportamento reativo do agente e com auxílio da CNCL, já encontram-se implementados os controladores *drible_opponent*, *drive_to_goal*, *follow_opponent*, *move_to_ball*, *pass_ball*, *tackle*, *watch_ball* e *kick_to_goal*. Na figura 8.1, são apresentadas algumas regras contidas no controlador *kick_to_goal*, como demonstração da sintaxe da CNCL.

```
rule_000.add_lhs(new CNFClause(input_lv.ball.distance,
input_lv.very_near));
rule_000.add_lhs(new CNFClause(input_lv.opp_goal.distance,
input_lv.very_near));
rule_000.add_rhs(new CNFClause(output_lv.kick_power, output_lv.strong));

rule_001.add_lhs(new CNFClause(input_lv.ball.distance,
input_lv.very_near));
rule_001.add_lhs(new CNFClause(input_lv.opp_goal.distance,
input_lv.near));
rule_001.add_rhs(new CNFClause(output_lv.kick_power, output_lv.strong));
```

Figura 8.1: Regras do controlador nebuloso *kick_to_goal*

As variáveis lingüísticas de entrada bem como seus termos lingüísticos são declarados no filtro de entrada assim como as variáveis lingüísticas de saída e seus respectivos termos lingüísticos são declarados no filtro de saída, do nível reativo. Na figura 8.2 é apresentada a variável lingüística de entrada *ball.distance* seguida de seus termos lingüísticos. Na figura 8.3 é definida a variável lingüística de saída *kick_power*.

8.3 Projeto do AG

O AG desenvolvido, visando otimizar os valores dos conjuntos nebulosos, contidos tanto no filtro de entrada como no filtro de saída do nível reativo, foi projetado com o auxílio de uma biblioteca C++,

```
unknown.init ("Unknown", UI1, UI2,  UI3);
very_near.init ("Very_Close", VNI1, VNI2, VNI3, VNI4);
near.init ("Close", NI1, NI2, NI3, NI4);
medium.init ("Medium", MI1, MI2, MI3, MI4);
far.init ("Far", FI1, FI2, FI3, FI4);
very_far.init ("Very_Far", VFI1, VFI2, VFI3, VFI4);

ball.distance.add_value_set(unknown);
ball.distance.add_value_set(very_near);
ball.distance.add_value_set(near);
ball.distance.add_value_set(medium);
ball.distance.add_value_set(far);
ball.distance.add_value_set(very_far);
```

Figura 8.2: Declaração da variável lingüística *ball.distance* e seus termos lingüísticos

```
strong.init("Strong", SI1, SI2, SI3, SI4);
medium_power.init ("Medium_power", MPI1, MPI2, MPI3, MPI4);
weak.init ("Weak", WI1, WI2, WI3, WI4);
back.init ("Back", BI1, BI2, BI3);

kick_power.add_value_set(strong);
kick_power.add_value_set(medium_power);
kick_power.add_value_set(weak);
```

Figura 8.3: Declaração da variável lingüística *kick_power* e seus termos lingüísticos

a GAlib [44], que é uma biblioteca de objetos de algoritmos genéticos. A biblioteca inclui ferramentas para uso de AG's em otimização de qualquer programa C++ usando qualquer tipo de representação e quaisquer operadores genéticos.

8.3.1 Características da GAlib

Quando manipula-se a biblioteca GAlib, primariamente trabalha-se com dois tipos de classes: um genoma e um algoritmo genético. Cada instância de genoma representa uma simples solução para o problema. O objeto algoritmo genético define como dar-se-a a evolução. O AG usa uma função objetivo, definida pelo usuário, para determinar a aptidão de cada genoma. Usa também os operadores de recombinação do genoma e estratégias de seleção para gerar novos indivíduos.

Normalmente, existem três coisas que devem ser feitas para resolver um problema utilizando um AG:

1. Definir uma representação
2. Definir os operadores genéticos
3. Definir a função objetivo

A GAlib auxilia nos dois primeiros ítens fornecendo muitos exemplos a partir dos quais é possível orientar-se na construção da representação e de operadores apropriados. A função objetivo é totalmente dependente do usuário.

Na GAlib, uma estrutura de dados é denomina **GAGenome**, o que também pode ser encontrado na literatura como *cromossomo*. Uma das classes derivadas de **GAGenome**, a **GARealGenome**, permite a representação por meio de números de ponto flutuante, sendo esta uma das razões que levaram a escolha desta biblioteca no auxílio à construção da implementação proposta.

8.3.2 Objetivo da otimização

As funções de pertinência para cada um dos conjuntos nebulosos utilizados, foram estabelecidas pelo velho e conhecido método de "tentativa e erro". Ou seja, estabeleciam-se os valores, compilava-se a biblioteca Expert-Coop++, recompilavam-se os controladores e executava-se o Soccerserver. Era feita uma avaliação intuitiva até que o controlador apresentasse um resultado satisfatório. Contudo, os controladores que fazem uso destes conjuntos obtiveram um resultado não mais que aceitável para uma primeira análise.

Em um primeiro instante, a intenção é utilizar um AG para otimizar os valores deste conjuntos nebulosos, obedecendo os limites máximos e mínimos de cada variável, e atribuindo uma variação máxima de 30% a cada conjunto. Assim, a forma final de cada conjunto permanece a mesma, bem como seus limites extremos, havendo uma variação de mesmo valor proporcional para cada conjunto de uma variável lingüística.

Como exemplo, apresentamos na figura 8.4, a variável lingüística *kick_power*. Na parte superior da figura ela encontra-se com seus valores originais. Na parte inferior a mesma variável foi alterada em 28% de seu formato original. No entanto, observe que os limites extremos (0 e 100) permanecem como na amostra original, bem como a forma dos 3 conjuntos que compõe a dita variável.

Para fins de validação do modelo proposto, foi utilizado o controlador nebuloso *kick_to_goal*, e apenas as variáveis lingüísticas de entrada e saída deste controlador passaram pelo processo de otimização. Este controlador apresenta como variáveis de entrada *ball.distance*, *ball.direction*, *opp_goal.distance*, *opp_goal.direction*, e como variáveis de saída *kick_power* e *kick_direction*. Cabe

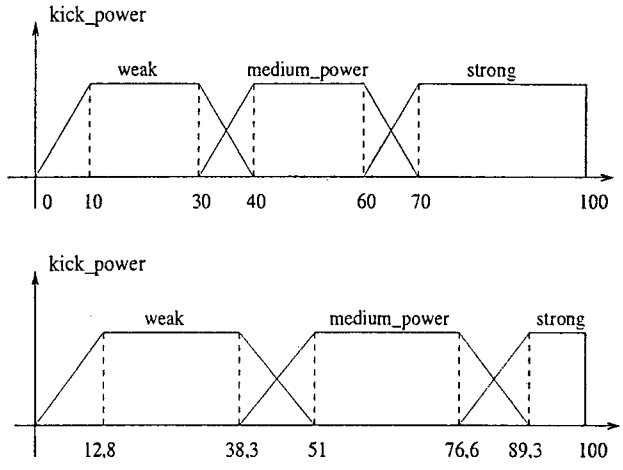

Figura 8.4: A variável lingüística kick_power antes de depois de uma variação

salientar que as variáveis de entrada *ball.distance* e *opp_goal.distance* utilizam o mesmo conjunto de termos lingüísticos, assim como as variáveis *ball.direction* e *opp_goal.direction.*

8.3.3 Implementação do AG

Nesta subseção são apresentados detalhes da implementação do AG utilizado para a otimização dos conjuntos nebulosos presentes nos filtros de entrada e saída do nível reativo.

Com o auxílio da biblioteca GAlib, um AG foi desenvolvido, onde foi associado a cada um dos conjuntos de termos lingüísticos, um gene. Cada gene pode assumir um valor no intervalo entre -1 e 1. O valor gerado como amostra é ainda multiplicado por um fator 0.3, garantindo aos conjuntos uma variação máxima de ±30% do valor original. Como parâmetros do AG foi fixada uma população de 20 indivíduos, com probabilidade de mutação de 0.001 e probabilidade de cruzamento de 0.9.

O critério de parada do AG utiliza o parâmetro de número de gerações. Uma função dentro do AG, denominada **Terminator**, faz um teste verificando se o número da atual geração é maior que o número estabelecido como limite. Caso este teste seja verdadeiro, a função retorna o flag `GATrue`, finalizando o AG e apresentando a solução final. Caso o critério de parada não seja atendido, o flag retornado é `GAFalse`, e o AG segue sua execução. Assim ficou estabelecido um limite de 100 gerações.

8.3.4 Função Objetivo

Para alguns tipos de aplicações, os AG's são mais atrativos que métodos de busca por gradiente uma vez que não exigem equações diferenciais complicadas ou um espaço de busca com propriedades específicas. O AG necessita apenas de uma simples medida da qualidade de um indivíduo em comparação aos demais indivíduos de uma população. A função objetivo fornece esta medida.

Entretanto, é importante ressaltar a diferença entre valor retornado pela função objetivo e o valor objetivo relativo. O primeiro é retornado pela função definida pelo usuário; é uma avaliação crua da performance de um genoma. O valor relativo, por outro lado, é uma taxa relativa a uma possível transformação usada pelo AG para determinar a aptidão dos indivíduos para reprodução. Este valor relativo é normalmente obtido através de uma escala linear dos valores objetivos brutos. Assim, o AG usa o valor relativo, e não o valor bruto, para fazer a seleção.

O primeiro controlador a passar pelo processo de otimização foi o *kick_to_goal*, que tem como função chutar a gol no exato momento em que é ativado. No que se refere a uma medida de avaliação deste controlador, a melhor forma encontrada foi posicionar um agente em um dado ponto na entrada da área adversária e exigir dele o chute em uma coordenada pré-determinada no fundo do gol adversário, considerado como ponto ideal para um chute a partir daquela posição. A medida de avaliação deu-se em cima da diferença entre a distância considerada como ideal e a distância alcançada.

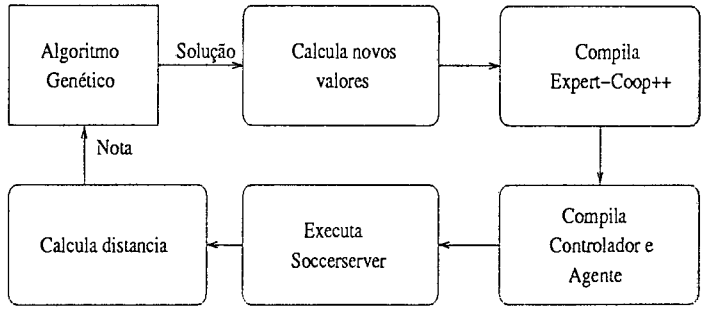

Figura 8.5: Esquema de execução do AG

É importante salientar que os valores a serem modificados por cada amostra gerada estão nos arquivos fontes que implementam os filtros de entrada e saída do nível reativo. Desta forma a cada interação[1] do AG, a função objetivo calcula os novos valores a serem alterados nestes arquivos, compila-os dentro da biblioteca Expert-Coop++, recompila o controlador *kick_to_goal* e o processo

[1]Chamamos interação cada execução da função objetivo

Interface a fim de que eles "linkem" a nova biblioteca gerada, e executa uma seção do Soccerserver, posicionando o agente em uma posição pré-estabelecida e lendo a posição alcançada ao término da execução. Um esquema apresentando este processo é apresentado na figura 8.5.

Em relação aos demais controladores, apenas a forma de avaliação difere da utilizada pelo *kick_to_goal*, devido a características específicas de cada controlador. Ou seja, todo o processo de gerar uma nova biblioteca Expert-Coop++ e recompilar os arquivos que possuem dependência com ela, continua exatamente como no caso do *kick_to_goal*, mas já no caso do controlador *drive_goal*, por exemplo, o fator de avaliação será o número de vezes que o agente tocou na bola para chegar até o gol adversário, a partir de uma posição pré-estabelecida.

Um pequeno entrave com relação ao método proposto é no que se refere ao tempo de simulação. Cada interação consome entre 30 a 35 segundos. No entanto, esforços no sentido de atenuar este problema já estão em desenvolvimento. Uma vez que a maior parte deste tempo é despendido na compilação dos dois arquivos que implementam os filtros, estes estão sendo decompostos em arquivos menores de forma que a cada interação somente sejam recompilados os arquivos contendo as variáveis alteradas.

8.4 Análise dos Resultados

Antes de uma análise dos resultados obtidos por meio da execução do AG, é interessante observar o gráfico com os conjuntos nebulosos projetados da forma convencional, ou seja, utilizando um método intuitivo e empírico. A figura 8.6 apresenta a representação original dos conjuntos nebulosos das variáveis lingüísticas envolvidas com o comportamento reativo *kick_to_goal.*

Por questão de redundância, não foram apresentadas as variáveis *opp_goal.distance* e *opp_goal.direction*, uma vez que estas apresentam os mesmos termos lingüísticos de, respectivamente, *ball.distance* e *ball.direction.*

Em uma primeira simulação, após 1102 interações, o AG convergiu a uma taxa final de 100%, fornecendo os seguintes resultados:

- variação de -3,59% com relação aos valores originais da variável *kick_power*;
- variação de -22,27% com relação à variável *kick_direction*;
- variação de 3,80% com relação às variáveis *ball.distance* e *opp_goal.distance*;

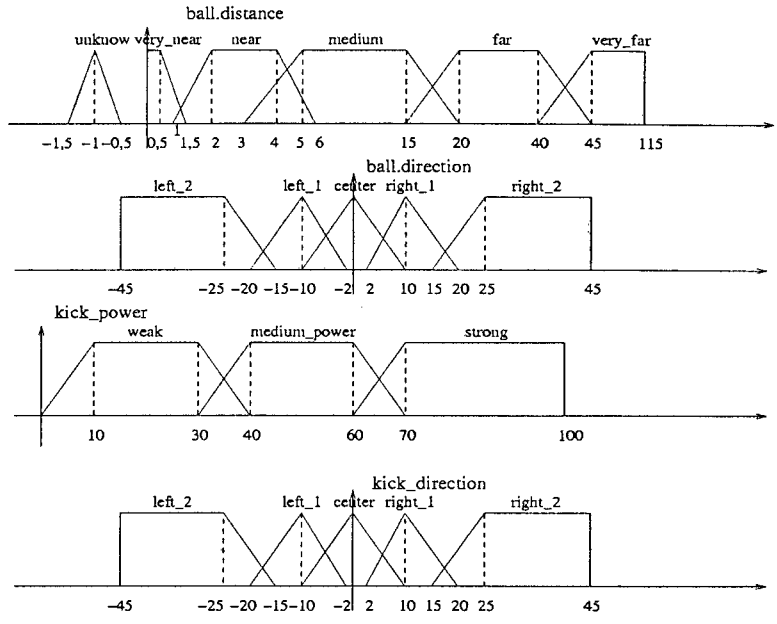

Figura 8.6: Variáveis lingüísticas do controlador *kick_to_goal*

- e, finalmente, uma variação de -10% com relação às variáveis *ball.direction* e *opp_goal.direction*.

Vale lembrar, que a variação máxima permitida para cada conjunto é de 30% do formato original. O formato final destes conjuntos é apresentado na figura 8.7.

Um ponto a ser observado é o fato de o AG ter alcançado uma taxa de convergência de 100% ao final de sua execução. Isto deveu-se ao baixo valor de alguns parâmetros como tamanho da população e o número de gerações utilizado para teste de convergência.

Uma segunda simulação foi executada, onde alguns parâmetros foram modificados. O primeiro deles foi o parâmetro `nConvergence` que determina o número de gerações utilizadas para o teste de convergência do AG. Este valor foi passado de 8 para 20. O outro parâmetro modificado foi o tamanho da população que de 20 foi para 35. O formato final dos conjuntos nebulosos é apresentado na figura 8.8.

Após 1354 interações, o AG convergiu a uma taxa de 94,75%, apresentando uma variação de 7,3, 6,96, 11,59 e 22,82%, respectivamente às variáveis *kick_power*, *kick_direction*, *ball.distance* e *ball.direction*.

Um resultado interessante das simulações é a não existência de uma solução única para os controladores. Aqui foram apresentados somente duas, mas na verdade, mais simulações foram feitas.

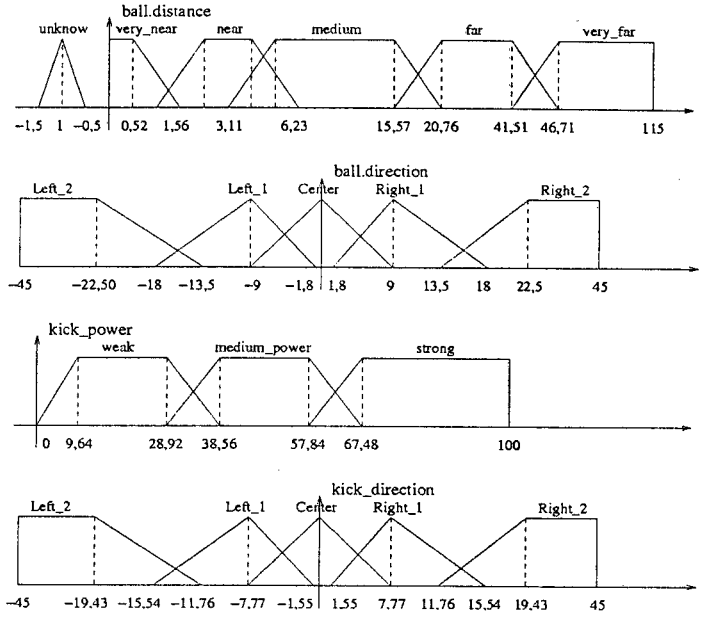

Figura 8.7: Variáveis lingüísticas do controlador *kick_to_goal*, depois do 1^o processo de otimização

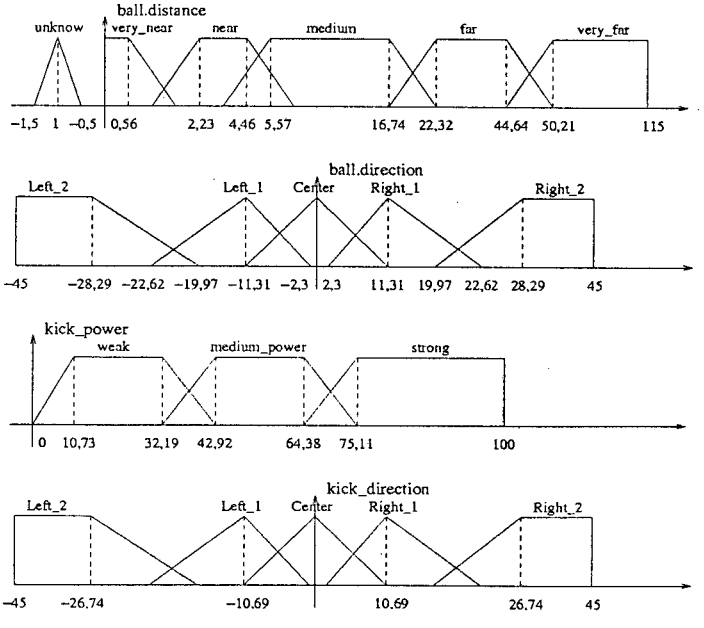

Figura 8.8: Variáveis lingüísticas do controlador *kick_to_goal*, depois do 2^o processo de otimização

Comparando os resultados, não existe um padrão de respostas geradas pelo AG. No entanto, todos os controladores gerados passaram por intensas seções de teste, onde todos apresentaram um resultado satisfatório. O que leva a concluir que não existe um único conjunto ótimo. Isto pode ser melhor compreendido através da observação da execução do AG. Em um determinado momento da execução o AG fixa um dado valor em um dos genes e fica variando os demais. Em seguida ele fixa dois genes, seguindo com a variação dos demais, até que ele fixa todos os genes. Ou seja, ele fixa um dos genes e vai adequando os demais àquele valor fixado, até que todos os genes tenham um valor ótimo. Este fenômeno é conhecido como *epistasis* que é assim definido em [4]:

> "Epistasis é a interação entre diferente genes em um cromossomo. É a expressão que designa o quanto um gene depende do valor de outro gene. O grau de interação será, geralmente, diferente para cada cromossomo. Se uma pequena mudança é feita em um gene espera-se uma mudança na avaliação do cromossomo. Esta mudança pode variar da acordo com os valores de outros genes."

8.5 Extensão dos Resultados

A partir da validação do modelo, este pode ser estendido a outras aplicações, dentro do agente UFSC-Team. O mesmo AG projetado para ajustar os conjuntos nebulosos a um ponto ótimo, pode ser utilizado ainda na construção da base de regras dos mesmos controladores nebulosos do nível reativo, assim como pode também ser utilizado no ajuste dos valores limiares definidos no nível instintivo, que a partir das informações de percepção definem o estado do jogo.

Vamos discutir estas aplicações por partes. Ainda dentro dos comportamentos do nível reativo, um AG pode ser utilizado no auxílio ao projeto das regras dos controladores. O método desenvolvido atende as requisitos exigidos por [30] que sugere que não se faça um aprendizado simultâneo da base de regras e do que ele convencionou chamar de partições nebulosas. Ele sugere esta divisão de tarefas uma vez que uma base de regras mal declarada pode resultar em mudanças drásticas dos conjuntos nebulosos.

Generalizando, [30] afirma que a tarefa de projetar um controlador nebuloso através de um AG pode ser dividida da seguinte forma:

- encontrar uma boa inicialização para a base de regras e para os conjuntos nebulosos;
- gerar uma base de regras que é capaz de manipular o dado problema;

- ajustar os conjuntos nebulosos para otimizar a performance do controlador.

Normalmente um AG inicializa uma base de regras aleatoriamente, não incorporando nenhum tipo de conhecimento sobre o problema. Na maioria das vezes, a base de regras inicial, difere muito da melhor base de regras otimizadas. O uso de um conhecimento implícito do processo pode reduzir o trabalho do AG levando a uma convergência em um tempo menor que o normal. Com a base de regras já construída até agora para os controladores existentes, pode-se afirmar que algum conhecimento prévio já existe. O AG parte daí para um conjunto de regras otimizadas.

Apenas com uma mudança na forma de codificar o problema, o projeto da base de regras dos controladores torna-se possível. Usemos como exemplo o controlador *kick_to_goal*. O processo de avaliação permanece o mesmo com relação ao já existente, apenas com a diferença que deve haver a modificação também no arquivo que implementa o controlador, efetuando as alterações necessárias.

No sistema especialista que é encapsulado pelo nível instintivo, a base de regras apresenta valores limiares, que auxiliam na inferência do estado corrente do jogo. Estes valores são também passíveis de otimização. Por exemplo, já supondo uma situação envolvendo mais de dois agentes, deve ser definida uma distância mínima entre um agente condutor da bola e seu marcador, para que aquele execute um passe. Através do mesmo processo utilizado para otimizar o controlador *kick_to_goal*, é possível otimizar aqui o arquivo que implementa a base de regras do nível instintivo, definindo os valores limiares ótimos para todas as situações ali declaradas.

Capítulo 9

Conclusões e Perspectivas

9.1 Conclusões

Neste trabalho foi apresentada a implementação de um algoritmo genético que tem por objetivo a automatização do projeto de controladores nebulosos e sistemas especialistas reativos presentes no agente desenvolvido para a construção de uma equipe de futebol de robôs virtuais, que participa da RoboCup ("Robo World Cup"), na liga do simulador. O algoritmo genético foi desenvolvido com o auxílio da biblioteca para objetos de algoritmos genéticos GAlib.

Dentro do agente UFSC-Team, os controladores nebulosos constituem-se no conjunto de comportamentos reativos de cada agente, onde cada comportamento é responsável por um dos fundamentos necessários para que um agente execute a tarefa de jogar futebol. Estes controladores fazem parte do nível reativo do agente, que é implementado através do processo *Interface*. Este processo é responsável pela resposta tempo real do agente, isto é, por receber a informação de percepção do Soccerserver e por enviar os comandos apropriados a ele. É importante salientar que em um dado momento somente um controlador está ativo, e este decide que comandos devem ser enviados ao Soccerserver, juntamente com seus respectivos valores.

O agente UFSC-Team, além do nível reativo, é composto por outros dois níveis de decisão. O nível instintivo é implementado no processo *Coordinator* e é responsável por atualizar as variáveis simbólicas usadas pelo nível cognitivo, o terceiro nível, e por escolher os comportamentos adequados, isto é, os controladores nebulosos adequados que devem ser usados no nível reativo a fim de alcançar uma dada meta. Uma meta pode ser alcançada por uma seqüência de comportamentos reativos que levam o agente a uma situação intencionada. A escolha desta seqüência de comportamentos é implementada através de um sistema especialista de um ciclo de inferência que escolhe, a cada vez que o

estado do jogo muda, o comportamento reativo mais adequado.

Finalmente, o terceiro nível, o nível cognitivo, implementado no processo *Expert*, é responsável por determinar as metas locais e globais do agente. Enquanto uma meta não falha ou obtém sucesso, o nível cognitivo não interfere no jogo. Este tempo ocioso é usado para planejamento estratégico. Este planejamento consiste na determinação de futuras metas locais, de acordo com o resultado atual, e na especificação de requisições de cooperação para alcançar metas globais. O nível cognitivo é também implementado por um sistema especialista, mas este pode ser muito mais complexo que aquele do nível instintivo, uma vez que seu tempo de resposta é maior.

Os controladores nebulosos são a utilização mais significativa da teoria dos conjuntos nebulosos em sistemas baseados em conhecimento. Um controlador nebuloso pode ser visto como um sistema especialista simplificado, onde a conseqüência de uma regra não é aplicada como antecedente de outra. Isto porque as ações de controle são baseadas em um único nível de inferência. Ao contrário dos controladores convencionais em que o algoritmo de controle é descrito analiticamente por equações algébricas ou diferenciais, através de um modelo matemático, em controle nebuloso utilizam-se regras lógicas no algoritmo de controle, com a intenção de descrever em uma rotina a experiência humana, intuição e heurística para controlar o processo. Os controladores nebulosos são robustos e de grande adaptabilidade, incorporando conhecimento que outros sistemas nem sempre conseguem acomodar. Também são versáteis, principalmente quando o modelo físico é complexo e de difícil representação matemática.

Embora a idéia de um controlador nebuloso seja muito atrativa, o projeto de uma base de regras apropriada e a especificação de conjuntos nebulosos necessários para um controle ótimo do processo é uma tarefa difícil e bastante tediosa, tomando um tempo valioso de projeto. Hoje, os algoritmos genéticos são considerados como uma possível solução para o problema de projeto e otimização deste tipo de controlador.

Os algoritmos genéticos são considerados como uma técnica de busca simples e ao mesmo tempo eficiente, produzindo resultados de qualidade comparáveis aos métodos de busca exaustiva, nos quais todas as possibilidades são exploradas. Sua maior deficiência porém, está relacionada com o tempo de processamento necessário na obtenção da convergência para uma solução satisfatória.

9.2 Perspectivas

Tomando como base o algoritmo genético já desenvolvido, bem como o *script* também já desenvolvido como parte da função objetivo do mesmo, os seguintes trabalhos devem ser desenvolvidos a

seguir:

- A inclusão de novos termos lingüísticos, especificamente no filtro de entrada do nível reativo a fim de diferenciar o tratamento dos vários objetos espalhados pelo campo. Por exemplo, atualmente, a variável lingüística *ball.distance* usa os mesmos termos lingüísticos da variável *opp_goal.distance*. Neste caso, uma distância que significa "perto" no caso da distância do gol adversário não significa necessariamente que é "perto" no caso da bola.

- A reformulação dos controladores nebulosos a partir de novos conjuntos de termos lingüísticos, adequando as regras existentes, e se necessário, a construção de novas regras.

- O desenvolvimento da parte do método de avaliação que é específico para cada controlador. No caso do *kick_to_goal* foi considerado a distância entre um ponto considerado como ideal e o ponto realmente alcançado.

- A extensão do trabalho do algoritmo genético a fim de operar também no projeto das regras de cada um dos controladores, na tentativa de automatizar praticamente todo o processo de construção dos controladores nebulosos.

- A extensão do uso dos algoritmos genéticos também na otimização dos valores limiares, estabelecidos na base de regras do sistemas especialista contido no nível instintivo, a fim de encontrar os valores ótimos para identificação dos possíveis estados do jogo declarados na base.

- O encapsulamento do algoritmo genético final, bem como da biblioteca GAlib, na biblioteca Expert-Coop++, após um processo de engenharia de software adequando-o à metodologia de orientação a objeto, que é característica desta última.

Referências Bibliográficas

[1] ANDREWS, G. R. *Concurrent Programing: Principles and Pratice.* The Benjamim/Cumming Publishing Co., 1991.

[2] BARR, A., AND FEIGENBAUM, E. A. *The Handbook of Artificial Intelligence*, vol. 1. Willian Kaufman, 1981.

[3] BEASLEY, D., BULL, D. R., AND MARTHIN, R. R. An overview of genetic algorithms: Part 1, fundamentals. Tech. rep., Inter-University Committee on Computing, 1993.

[4] BEASLEY, D., BULL, D. R., AND MARTHIN, R. R. An overview of genetic algorithms: Part 2, research topics. Tech. rep., Inter-University Committee on Computing, 1993.

[5] BERENJI, H. R., AND KHEDKAR, P. Learning and tuning fuzzy logic controllers through reinforcemnets. *IEEE Trans. Neural Networks 3* (1992), 274–740.

[6] BERRY, A. Soccer robots with local vision. Master's thesis, University of Western Australia, Austrália, October 1999.

[7] BITTENCOURT, G. In the quest of the missing link. In *International Joint Conference on Artificial Intelligence (IJCAI'97)* (Nagoya, Japan, August, 23 - 29 1997), pp. 310 – 315.

[8] BITTENCOURT, G. *Inteligência Artificial: Ferramentas e Teorias.* Editora da UFSC, Florianópolis, 1998.

[9] BOUCHON-MEUNIER, B. *La Logique Floue.* Presses Universitaires de France, 1993.

[10] BUCHANAN, B., BARSTOW, D., BECHTEL, R., BENNET, J., CLANCEY, W., KULIKOWSKI, C., MITCHELL, T., AND WATERMAN, D. A. *Building Expert Systems.* Addison-Wesley, 1983, ch. 5.

[11] CORREA, C. Uso de algoritmos genéticos na construção de controlador nebuloso para o controle de altitude de um satélite artificial durante a fase de apontamento. Master's thesis, Instituto Nacional de Pesquisas Espaciais, 1999.

[12] CORTEN, E., DORER, K., HEINTZ, F., KOSTIADIS, K., KUMMENEJE, J., MYRITZ, H., NODA, I., RIEKKI, J., RILEY, P., STONE, P., AND YEAP, T. *Soccerserver Manual*, July 1999.

[13] DA COSTA, A. C. P. L., AND BITTENCOURT, G. Parla: A cooperation language for cognitive multi-agent systems. In *8th Portuguese Conference of Artificial Inteligence* (1997), pp. 492–497.

[14] DA COSTA, A. C. P. L., AND BITTENCOURT, G. Soccer server: Um simulador para o futebol de robôs da robocup federation tutorial-3. In *Encontro Nacional de Inteligência Artificial* (1999).

[15] DA COSTA, A. L. Expert coop: Um ambiente para desenvolvimento de sistemas multi-agentes cognitivos. Master's thesis, UFSC - Universidade Federal de Santa Catarina, 1997.

[16] DA COSTA, A. L., AND BITTENCOURT, G. Ufsc-team: A cognitive multi-agent approach to the robocup'98 league. In *RoboCup'98 Workshop* (1998), pp. 371 – 376.

[17] DA COSTA, A. L., AND BITTENCOURT, G. From a concurrent architecture to a concurrent autonomous agents architecture. In *International Joint Conference on Artificial Intelligence (IJCAI'99)* (1999).

[18] DRIANKOV, D., HELLENDOORN, H., AND REINFRANK, M. *An Introduction to Fuzzy Control.* Springer-Verlag, 1993.

[19] DUBOIS, D., AND PRADE, H. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, 1980.

[20] DUBOIS, D., AND PRADE, H. *Possibility Theory - An Approach to the Computerized Processing of Uncertainty.* Academic Press, 1988.

[21] FAIN, J., HAYES-ROTH, F., SOWIZRAL, H., AND WATERMAN, D. A. Programming in roise: An introduction by means of examples. Tech. rep., Rand Corporation, 1982. Technical Report N-1646-ARPA.

[22] FEDERATION, T. R. Robocup. www.robocup.org. Home Page.

[23] FEIGENABAUM, E. A., BUCHANAN, B. G., AND LEDERBERG, J. On generality and problem solving: A case study using the dendral program. In *Machine Intelligence* (1971), B. Meltzer and D. Michie, Eds., vol. 6, Edimburgh University Press, pp. 165 – 190.

[24] FORGY, C. L., AND MCDERMOTT, J. Ops: A domain independent production system. In *Proc. of IJCAI 5* (1977), pp. 933–939.

[25] GENSYM. *G2 Reference Manual.* GENSYM Coorporation, 1992.

[26] GIARRATANO, J., AND RILEY, G. *Expert Systems: Principles and Programming*, second ed. PWS Publishing Company, 1994.

[27] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.

[28] JACKSON, P. *Expert Systems*, third ed. Addison Wesley, 1998.

[29] JUNIUS, M., AND STEPPLE, M. *CNCL Reference Manual.* Universität Aachen, 1997. http://www.comnets.rwth-aachen.de/cnroot_engl.html.

[30] KINZEL, J., KLAWONN, F., AND KRUSE, R. Modifications of genetic algorithms for designing and optimizing fuzzy controllers. Tech. rep., Technical University of Braunschweig, 1999. ftp://ftp.et-inf.fho-emden.de/pub/FHO/Informatik/klawonn/eci3e94.ps.gz.

[31] KITANO, H., ASADA, M., KUNIYOSHI, Y., NODA, I., AND OSAWA, E. Robocup: The robot world cup initiave. In *The First International Conference on Autonomous Agent (Agents-97)* (1997).

[32] KITANO, H., VELOSO, M., MATSUBARA, H., TAMBE, M., CORADESCHI, S., NODA, I., STONE, P., OSAWA, E., AND ASADA, M. The robocup syntethic agent challenge 97. In *International Joint Conference on Artificial Intelligence (IJCAI 97)* (Nagoya, Japão, 1997).

[33] KOSKO, B. *Neural Networks and Fuzzy Systems.* Prentice Hall, 1992.

[34] LEE, C. Fuzzy logic in control systems: Fuzzy logic controller - part. i e ii. *IEEE Transactions on Systems, Man and Cybernetics* (1990).

[35] MACKWORTH, A. On seeing robots. In *Computer Vision: System, Theory, and Applications.* World Scientific Press, Singapore, 1993, pp. 1 – 13.

[36] MINSKY, M. A framework to represent knowledge. *The Psychology of Computer Vision* (1975), 211–277.

[37] MUELLER, F. Pthreads library interface. Tech. rep., Institut für Informatik - Humboldt - Universitat zu Berlim, Berlim, November 1999.

[38] POST, E. Formal reductions of the general combinatorial problem. *American Journal of Mathematics* (1943), 65:197–268.

[39] QUILLIAN, M. R. Semantic memory. *Semantic Information Processing* (1968), 216–270.

[40] RUSSEL, S., AND NORVIG, P. *Artificial Intelligence, A Modern Approach.* Alan Apt, 1995.

[41] SANDRI, S., AND CORRÊA, C. Lógica nebulosa. Tech. rep., INPE - Instito Nacional de Pesquisas Espaciais, São José dos Campos, SP, 1999.

[42] SHORTLIFFE, E. H. *Computer-Based Medical Consultations: MYCIM.* Americam Elsevier, New York, 1976.

[43] SUN MICROSYSTEMS. *Networking Programming Guide*, 1990.

[44] WALL, M. *GAlib: A C++ Library of Genetic Algorithm Components.* Massachusetts Institute of Technology, August 1996. http://lancet.mit.edu/ga/.

[45] WATERMAN, D. A. *A Guide to Expert Systems.* Addison-Wesley, 1986.

[46] WHITLEY, D. *A Genetic Algorithm Tutorial.* Computer Science Department, Colorado State University.

[47] WINSTON, P. *Artificial Intelligence*, third ed. Addison-Wesley, 1992.

[48] ZADEH, L. A. Fuzzy ses. *Information and Control* (1965).