ANDERSON LUIZ FERNANDES PEREZ

# Extensão da Programação Genética Distribuída para Suportar a Evolução do Sistema de Controle em uma População de Robôs Móveis

FLORIANÓPOLIS

2010

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

# Extensão da Programação Genética Distribuída para Suportar a Evolução do Sistema de Controle em uma População de Robôs Móveis

Tese de Doutorado submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do Titulo de Doutor em Engenharia Elétrica.

ANDERSON LUIZ FERNANDES PEREZ

Florianópolis, fevereiro de 2010

Catalogação na fonte pela Biblioteca Universitária
da
Universidade Federal de Santa Catarina

P438e Perez, Anderson Luiz Fernandes
Extensão da programação genética distribuída para suportar a evolução do sistema de controle em uma população de robôs móveis [tese] / Anderson Luiz Fernandes Perez ; orientadores, Guilherme Bittencourt, Mauro Roisenberg. - Florianópolis, SC, 2010.
108 p.: il., grafs., tabs.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica.

Inclui referências

1. Engenharia elétrica. 2. Robótica móvel. 3. Evolução embarcada. 4. Robótica evolutiva. 5. Programação genética. 6. Programação genética distribuída. I. Bittencourt, Guilherme. II. Roisenberg, Mauro. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

CDU 621.3

**Extensão da Programação Genética Distribuída para Suportar a Evolução do Sistema de Controle em uma População de Robôs Móveis**

Anderson Luiz Fernandes Perez

Esta tese foi julgada adequada para obtenção do título de Doutor em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Prof. Guilherme Bittencourt, Dr.
Orientador (*in memoriam*)

Prof. Mauro Roisenberg, Dr.
Co-orientador

Prof. Roberto de Souza Salgado, Ph.D.
Coordenador do Programa de
Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Mauro Roisenberg, Dr.
Presidente

Prof. Aluizio Fausto Ribeiro Araújo, Dr.
Universidade Federal de Pernambuco

Prof. Augusto Cesar Pinto Loureiro da Costa, Dr.
Universidade Federal da Bahia

Prof. Leandro Buss Becker, Dr.
Universidade Federal de Santa Catarina

Prof. Ubirajara Franco Moreno, Dr.
Universidade Federal de Santa Catarina

*Aos meus pais, Odair (in memoriam) e Mercedes, e à minha esposa Eliane.*

# AGRADECIMENTOS

À minha família pelo apoio, incentivo e o carinho, que foram fundamentais para a conclusão deste trabalho.

À minha esposa que sempre esteve ao meu lado nos piores e nos melhores momentos que passei durante o desenvolvimento da tese.

Ao meu orientador, professor Guilherme Bittencourt, pelo incentivo, apoio, amizade e principalmente suas contribuições que foram fundamentais para o desenvolvimento do trabalho.

Ao professor Mauro Roisenberg, por ter aceitado o desafio de co-orientar este trabalho e por suas importantes contribuições durante o desenvolvimento do mesmo.

Ao relator, professor Aluizio, pelo excelente trabalho realizado em seu parecer.

Aos membros da banca examinadora pelas sugestões e contribuições ao trabalho.

Aos professores e funcionários do Departamento de Automação e Sistemas e da Pós-Graduação em Engenharia Elétrica, em especial, ao Wilson e ao Marcelo pelo profissionalismo e competência.

Ao CNPQ pela bolsa de doutorado no Brasil e também pela bolsa de doutorado sanduíche na Alemanha.

Ao DAAD pela bolsa de estudos de língua alemã na cidade de Marburg, Alemanha.

Aos amigos, Flávio, Fernando, Adriana, Underléa e Jerusa, por terem me aturado nos momentos de estresse durante o desenvolvimento do trabalho.

Aos colegas de estudo na Alemanha, Alexandre, Ricardo, Vivian, Mauro e Letícia, pela amizade e as excelentes horas de bate papo.

Aos colegas, Bjorn, Suei, Ricardo e Benedikt, do Instituto de Controle de Processos e Robótica (IPR) da Universidade de Karlsruhe, pela acolhida, paciência e amizade, durante minha estada na Alemanha.

Aos colegas de trabalho do Colegiado de Engenharia de Computação da UNIVASF, pelo companheirismo e o incentivo.

A todos que, direta ou indiretamente, contribuíram para a elaboração deste trabalho, meu sincero MUITO OBRIGADO!

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

# Extensão da Programação Genética Distribuída para Suportar a Evolução do Sistema de Controle em uma População de Robôs Móveis

**Anderson Luiz Fernandes Perez**

Fevereiro / 2010

Orientador: Guilherme Bittencourt, Dr.
Co-orientador: Mauro Roisenberg, Dr.
Área de Concentração: Automação e Sistemas
Palavras-chave: robótica móvel, evolução embarcada, robótica evolutiva, programação genética, programação genética distribuída.
Número de Páginas: xiv + 94

As pesquisas em robótica móvel visam o estudo e o desenvolvimento de máquinas capazes de se locomover de forma autônoma ou semi-autônoma. Quando a locomoção se dá em ambientes ruidosos, não controlados ou desconhecidos, é necessário que o sistema de controle seja flexível para permitir a auto-adaptação, conforme se dá a interação do robô com o ambiente. Para tornar o sistema de controle de um robô móvel mais adaptável é necessário utilizar alguma técnica de aprendizado que permita que o sistema se modifique ao longo de sua execução. A robótica evolucionária objetiva o desenvolvimento de sistemas de controle adaptativos baseados em técnicas da computação evolucionária. A evolução embarcada faz com que o processo evolucionário ocorra sobre uma população de robôs móveis, isto é, acontece entre os indivíduos que fazem parte da população de robôs. Neste trabalho é apresentado o algoritmo da xPGD, uma Extensão do algoritmo da Programação Genética Distribuída, capaz de suportar a evolução do sistema de controle de uma população de robôs móveis. Também é apresentado o Sistema de Execução, Gerenciamento e Supervisão da xPGD (SEGS). Para avaliar o algoritmo da xPGD foram realizados três experimentos em ambientes simulados, que são: navegação livre de colisões, forageamento e empurrar uma caixa. A xPGD e o SEGS possuem, respectivamente, as seguintes características: i) estende o algoritmo da programação genética distribuída para suportar a evolução em uma população de robôs; ii) o processo evolucionário acontece de forma assíncrona entre os robôs da população; iii) é tolerante a falhas, pois permite a continuação do processo evolucionário mesmo que só reste um único robô na população de robôs; iv) guarda informações sobre os indivíduos mais aptos em memória; v) possui um ambiente de execução e gerenciamento independente do processo evolucionário.

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

# Distributed Genetic Programming Extension to Support the Control System Evolution in a Mobile Robot Population

**Anderson Luiz Fernandes Perez**

February / 2010

Advisor: Guilherme Bittencourt, Dr.
Co-advisor: Mauro Roisenberg, Dr.
Area of Concentration: Automation and Systems
Keywords: mobile robots, embodied evolution, evolutionary robotics, genetic programming, distributed genetic programming.
Number of Pages: xiv + 94

The research in mobile robotics aims the study and development of machines that are able to move in an autonomous or semi-autonomous way. If the locomotion occurs in noisy, uncontrolled or unknown environments, a flexible control system is necessary, to allow for robot self-adaptation, as its interaction with the environment proceeds. For mobile robot control system to become more adaptable, it is necessary to use a development technique that allows the system to modify itself during its execution. Evolutionary robotics aims the development of adaptive control systems based on evolutionary computation techniques. Embodied evolution makes the evolutionary process occurs over a population of mobile robots, the process occurs among the individuals that participate in the population of robots. In this work, we present the xDGP algorithm, an Extension of the Distributed Genetic Programming algorithm, able to support the evolution of the control system for a mobile robot population. We also present the xDGP Execution, Management and Supervision System (EMSS). To evaluate the xDGP algorithm three experiments were accomplished in simulated environments, namely: collision-free navigation, foraging and box-pushing. xDGP and EMSS have, respectively, the following characteristics: i) it extends the distributed genetic programming algorithm to allow the evolution in a population of robots; ii) the evolutionary process occurs in an asynchronously way among the robots in the population; iii) it is fail-safe, therefore it allows the continuation of the evolutionary process even if only one robot remains in the population of robots; iv) it saves the information about the more adapted individuals in memory; v) it has an execution and management environment that is independent of the evolutionary process.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas

| | |
|---|---|
| **AE** | Algoritmo Evolucionário |
| **ADFs** | *Automatically Defined Functions* |
| **ADM** | *Automatically Defined Macros* |
| **AG** | Algoritmo Genético |
| **ALife** | *Artificial Life* |
| **API** | *Application Programming Interface* |
| **ARL** | *Adaptive Representation through Learning* |
| **CE** | Computação Evolucionária |
| **CTE** | Controle Evolucionário |
| **CPU** | *Central Processing Unit* |
| **DAF** | Definição Automática de Funções |
| **DARPA** | *Defense Advanced Research Projects Agency* |
| **DNA** | *Deoxyribonucleic Acid* |
| **DSP** | *Digital Signal Processing* |
| **EE** | Evolução Embarcada |
| **EH** | Evolução Híbrida |
| **FPGA** | *Field Programmable Gate Array* |
| **GA** | *Genetic Algorithm* |
| **GB** | Gerenciador de Bibliotecas |
| **GC** | Gerenciador de Comunicação |
| **GPL** | *General Public License* |
| **IC** | Inteligência Computacional |
| **JPL** | *Jet Propulsion Laboratory* |
| **MIMD** | *Multiple Instruction-streams, Multiple Data-streams* |
| **PD** | (Controlador) Proporcional-Derivativo |

| | |
|---|---|
| **PEGA** | *Physically Embedded Genetic Algorithm* |
| **PG** | Programação Genética |
| **PGD** | Programação Genética Distribuída |
| **PGP** | Programação Genética Paralela |
| **PGTA** | *Probabilistic Gene Transfer Algorithm* |
| **PTC** | *Probabilistic Tree Creation* |
| **RAM** | *Random Access Memory* |
| **RBC** | Robótica Baseada em Comportamentos |
| **RE** | Robótica Evolucionária |
| **RIA** | *Robotic Industries Association* |
| **RoBIOS** | *Robot Basic Input Output System* |
| **RNA** | Rede Neural Artificial |
| **RUR** | *Rossum's Universal Robots* |
| **SEGS** | Sistema de Execução, Gerenciamento e Supervisão |
| **SMR** | Sistema com Múltiplos Robôs |
| **xPGD** | Extensão da Programação Genética Distribuída |

# Capítulo 1

# Introdução

Ao longo da história da humanidade diversos autômatos (ancestrais dos atuais robôs) foram criados com o objetivo de facilitar algumas tarefas do dia-a-dia. Um marco na história da construção dos autômatos é a primeira máquina de calcular, criada por Blaise Pascal em 1642. A partir desta época inúmeros autômatos foram desenvolvidos, mas foi somente a partir de 1923 que o termo robô começou a ser empregado, tendo sido usado pela primeira vez por Karel Capek em uma peça chamada R.U.R (*Rossum's Universal Robots*) (Murphy, 2000). A palavra robô tem origem na palavra tcheca robota, que significa "trabalho forçado".

De acordo com a RIA (*Robotic Industries Association*) um robô é um manipulador programável multi-funcional capaz de mover materiais, partes, ferramentas ou dispositivos específicos através de movimentos variáveis programados para realizar uma variedade de tarefas (Nehmzow, 2000) e (Pazos, 2002). Os robôs são comumente utilizados na realização de tarefas consideradas repetitivas, estressantes ou perigosas para os seres humanos (Wolf *et al.*, 2009) e (Duffy, 2000).

Os primeiros robôs eram puramente mecânicos e executavam tarefas de modo repetitivo. Estes robôs deram origem aos atuais robôs manipuladores de base fixa, e são largamente adotados na indústria (e.g. indústria automotiva) (Dudek e Jenkin, 2000). Somente mais recentemente surgiram os robôs móveis, que se caracterizam pela capacidade de se deslocar de modo guiado, semi-autônomo ou totalmente autônomo.

De acordo com Wang (2002) a robótica está migrando de um ambiente industrial estruturado para um ambiente não estruturado. Robôs manipuladores estão sendo substituídos por robôs móveis inteligentes que são capazes de se ajustar autonomamente a ambientes

não estruturados (Thakoor *et al.*, 2004). Um robô móvel é um dispositivo mecânico montado sobre uma base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que os permitem interagir com o ambiente (Arkin, 1998).

Ao longo dos anos a robótica móvel vem se tornando uma área cada vez mais consolidada. Em 1950, foram realizados os primeiros experimentos com robôs móveis pelo pesquisador Willian Grey Walter, que construiu os robôs "tortoise" que eram capazes de executar tarefas como desviar de obstáculos e seguir fontes luminosas (Bekey, 2005).

Nos dias atuais um exemplo de sucesso no desenvolvimento de veículos móveis de alta tecnologia são os veículos desenvolvidos pelo JPL (*Jet Propulsion Laboratory*) da NASA para explorar o planeta Marte (Dudek e Jenkin, 2000). As sondas espaciais *Soujourner*, *Spirit* e *Opportunity* são veículos de autonomia limitada, pois necessitam de comandos enviados a distância por seres humanos que controlam a missão. O carro robô *Stanley* (Team, 2006), que venceu o desafio DARPA (*Defense Advanced Research Projects Agency*) de 2005, é outro exemplo de veículo autônomo de alta tecnologia.

## 1.1 Motivação e Relevância

A robótica móvel ocupa-se em estudar e desenvolver máquinas capazes de se locomover, em geral, em ambientes não controlados, ruidosos e desconhecidos, tais como: exploração de outros planetas, resgate de sobreviventes em catástrofes (e.g. terremotos, desabamento de minas, explosões etc) (Holland, 2004). Para operar neste tipo de ambiente o robô deve dispor de um conjunto de funcionalidades, tais como (Hainen, 2002): ser capaz de adquirir e utilizar conhecimento sobre o ambiente, possuir a habilidade de reconhecer obstáculos, e responder em tempo real às situações que possam ocorrer neste ambiente.

Segundo Jung *et al.* (2006), os seguintes itens devem ser considerados na construção de um robô móvel:

- **Locomoção**: como o robô irá se deslocar no ambiente. Os dispositivos de locomoção são de grande importância na caracterização de um robô móvel (e.g. rodas, propulsores, esteiras etc.);
- **Percepção**: como o robô irá perceber o ambiente (e.g. sensores de contato e de distância em relação a obstáculos) e inclusive monitorar parâmetros próprios dele e

de seu comportamento (e.g. carga da bateria, odometria) e então planejar as ações que serão executadas;

- **Planejamento**: como o robô irá transformar suas percepções e conhecimentos prévios adquiridos (e.g. mapa do ambiente) em ações, ou seqüências de ações, a serem executadas;

- **Comunicação**: como o robô irá se comunicar com um operador humano, ou mesmo com outros dispositivos robóticos (e.g. redes sem fio, cabo serial ligado a uma estação, infravermelho).

Um robô interage com o o ambiente através de ciclos de percepção-ação que consistem em três passos fundamentais (Arkin, 1998):

- obtenção de informações sobre o ambiente através dos sensores;
- processamento das informações obtidas e seleção de ações que serão executadas;
- execução das ações selecionadas através do acionamento dos atuadores.

Um robô realiza diversos ciclos de percepção-ação, modificando o estado do ambiente em busca da realização da tarefa (Ribeiro *et al.*, 2001). A Figura 1.1 ilustra o ciclo percepção-ação.

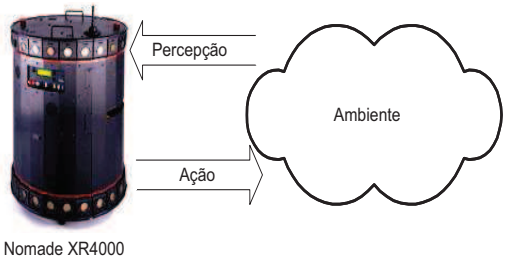


Figura 1.1: Ciclo percepção-ação.

Para que um robô realize alguma tarefa é necessário um sistema computacional e uma arquitetura de software para controlar as ações do robô. Uma arquitetura de controle para um robô móvel é responsável por interpretar as informações do ambiente coletadas pelos sensores e em seguida tomar alguma decisão sobre qual ação deve ser executada através

dos atuadores (Mataric, 2001). Uma arquitetura de controle para um robô define como é organizada a tarefa de gerar ações como função da percepção (Russell e Norvig, 2004).

Uma arquitetura de software é uma estrutura de um programa ou de um sistema computacional composta por componentes de software na qual cada componente representa uma funcionalidade na arquitetura.

De acordo com Alvarez *et al.* (2001) uma arquitetura de software de controle para robôs deve apresentar os seguinte aspectos:

- Funcionalidade e Desempenho: deve suportar a execução de algoritmos de controle bem como outras funções relacionadas que são importantes para um correto funcionamento do sistema de controle.

- Confiabilidade: deve ser projetada cuidadosamente e considerar os aspectos de hardware de forma a evitar e, quando possível tolerar, falhas em diferentes níveis hierárquicos.

Existem diferente técnicas que podem ser aplicadas no projeto de sistemas de controle para robôs móveis. Essa técnicas podem ser controladores clássicos como o PID (Proporcional-Integral-Derivativo) (Rosário, 2005) ou baseadas em Inteligência Computacional como as Redes Neurais (Haykin, 2001) e os Algoritmos Evolucionáios (Whitley, 2001).

A Inteligência Computacional (IC) desempenha um papel importante na área de robótica. É ela a responsável por fazer com que a robótica não produza simples máquinas automáticas com comportamentos fixos, mas robôs totalmente autônomos capazes de tomarem decisões independentemente de um operador humano (Jarvis, 2008). De acordo com Nehmzow (2000), para os pesquisadores de IC, robôs autônomos móveis oferecem meios para testar hipóteses sobre comportamento inteligente, cognição e percepção.

A principal motivação deste trabalho está na aplicação de técnicas de IC para desenvolver sistemas de controle para robôs móveis com o objetivo de torná-los o mais autônomo possível. Em particular, usar Programação Genética para gerar sistemas de controle que possam se adaptar as mais variadas situações que o robô poderá enfrentar durante sua interação com o ambiente.

## 1.2 Contextualização

Programar um sistema de controle para um robô, muitas vezes, exige conhecimentos por parte do programador que não estão disponíveis durante o processo de desenvolvimento. Principalmente se o ambiente no qual o robô irá atuar for ruidoso, não estruturado ou desconhecido (Arkin, 1998). Nesse caso, é importante que o robô tenha um alto grau de autonomia e seja dotado de mecanismos que permitam sua auto-adaptação para poder tomar decisões em situações para as quais ele não foi programado.

Para tornar o sistema de controle de um robô móvel mais adaptável, é necessário utilizar alguma técnica de desenvolvimento que permita que o sistema se modifique ao longo de sua execução. O objetivo é fazer com que comportamentos básicos, tais como: seguir em frente, virar a direita, virar a esquerda, retornar, em outros, sejam rearranjados visando a solução do problema.

A Computação Evolucionária (CE) (Fogel, 2000), através de seus Algoritmos Evolucionários (AEs), possibilita o desenvolvimento de sistemas de controle adaptativos para robôs móveis (Pollack *et al.*, 2000). Os AEs apresentam uma seqüência de procedimentos gerais que podem ser adaptados a cada problema. Os pontos no espaço de busca são modelados através de indivíduos que compõem um conjunto de soluções (população) (Barlow e Oh, 2008). Uma função de aptidão calcula o quanto este indivíduo está apto ao problema ou o quão próximo está da solução do problema (Nelson *et al.*, 2008).

A aplicação de técnicas da CE para o desenvolvimento de sistemas de controle para robôs móveis deu origem à Robótica Evolucionária (RE) (Nolfi e Floreano, 2002). De acordo com Sofge *et al.* (2003) a RE representa, ao mesmo tempo, um desafio e uma grande oportunidade para os roboticistas, pois trata-se da aplicação de algoritmos, que em muitos casos, consomem uma grande quantidade de recursos computacionais tais como memória e tempo de processamento, que muitas vezes podem ser limitados em um robô móvel.

## 1.3 Definição do Problema

Em seu trabalho intitulado *Challenges in Evolving Controllers for Physical Robots* Mataric e Cliff (1996) fizeram algumas críticas com relação ao uso de técnicas de CE para o desenvolvimento de sistemas de controle para robôs móveis e apontaram alguns

desafios para a área de RE. Alguns desses desafios são:

1. **Evolução em robôs reais:** o processo evolucionário demanda um longo perído de tempo de execução para encontrar uma solução ao problema.

2. **Evolução em simulador:** nem sempre é possível representar as características do mundo real em ambiente simulado.

3. **Dificuldades em definir uma função de aptidão:** dependendo do problema, a definição de uma função de aptidão pode ser uma tarefa difícil para o programador.

4. **Dificuladades em codificar o problema usando algoritmos evolucionários:** devido às características dos AEs a codificação inicial do problema pode demandar um esforço extra do programador.

Respondendo a um dos desafios proposto por Mataric e Cliff (1996), a evolução do sistema de controle em robôs reais, Ficici *et al.* (1999) propôs uma nova abordagem para a RE que foi definida como Evolução Embarcada (EE). Na EE um grupo de robôs está situado em um determinado ambiente tentando executar uma tarefa. Neste caso, cada robô, fisicamente, representa um candidato à solução do problema, o processo evolucionário acontece entre os robôs do grupo ou população, isto é, o processo de seleção e reprodução acontecem entre os indivíduos que fazem parte da população de robôs.

A EE é uma grande contribuição à área de RE, porém as soluções desenvolvidas em EE são altamente dependentes do bom funcionamento dos robôs. Em EE os robôs, a cada intervado de tempo, estão em um dos dois seguintes estados: i) explorando o ambiente, ou seja, quando a solução que está embarcada no robô está sendo testada; ii) procurando um parceiro para cruzar e se reproduzir, por isso, é desejável que o grupo ou população tenha um número par de robôs.

Alguns trabalhos em EE utilizam Algoritmos Genéticos (AGs) com redes neurais (Lingfeng Wang, 2006), (Watson *et al.*, 2002). O AG é responsável por ajustar os pesos dos neurônios ou até mesmo em reconfigurar a arquitetura da rede neural (Nolfi e Floreano, 2001). Nesses trabalhos cada gene de um cromossomo é representado por um conjunto de bits que codificam algum parâmetro e a troca de material genético entre a população de robôs se dá através da transmissão de um único gene (Ficici *et al.*, 1999) ou de todo o cromossomo (Nehmzow, 2002) de um robô para outro.

Usar AG para gerar controladores para robôs móveis exige que o programador tenha uma bom domínio do problema para que o mesmo consiga modelar adequadamente o AG. Nem sempre o programador tem o domínio total do problema, o que, muitas vezes, dificulta o desenvolvimento e futuras alterações do sistema.

Considerando os desafios citados por Mataric e Cliff (1996) e algumas desvantagens do uso da EE que são, a ausência de um mecanismo de tolerência a falhas e a dificultade na codificação do AG, este trabalho visa o desenvolvimento de um algoritmo evolucionário que seja capaz de resolver as seguintes questões em RE:

1. Garantir a continuidade do processo evolucionário mesmo quando ocorrerem falhas com os robôs que fazem parte da população de robôs.

2. Permitir que os robôs consigam cruzarem e se reproduzirem indiferentemente do número de robôs ativos no grupo.

3. Facilitar a codificação das soluções sem que o programador tenha que saber detalhes do problema a ser resolvido.

Para solucionar as questões (1) e (2) foi desenvolvido um sistema modular que facilita o processo evolucionário que acontece de maneira embarcada em cada robô. A questão (3) é resolvida com o uso de Programação Genética Distribuída (PGD) (Fernández *et al.*, 2002). A PGD é algoritmo evolucionário baseado na Programação Genética (Koza, 1992a), porém na PGD cada indivíduo da população de candidatos à solução do problema pode estar fisicamente distribuído.

O principal objetivo da PG é ensinar computadores a se programar, isto é, a partir de especificações de comportamentos primários, o computador deve ser capaz de gerar automaticamente um programa que satisfaça alguma condição que visa a solução de alguma tarefa ou problema.

A vantagem em utilizar PG para o desenvolvimento de sistemas de controle para robôs móveis é que as estruturas (funções e terminais) manipuladas são de alto nível, ou seja, o programador/desenvolvedor é responsável por criar rotinas (funções e procedimentos) que farão parte do sistema de controle dos robôs. Essa característica torna o processo de desenvolvimento de controladores evolucionários para robôs móveis mais intuitivo para o programador.

Neste trabalho é apresentado um novo algoritmo evolucionário baseado no algoritmo da PGD. A principal característica deste algoritmo é a capacidade de evoluir tanto o sistema de controle de um único robô, bem como o sistema de controle de uma população de robôs móveis.

## 1.4 Objetivo Geral

O objetivo desta tese é propor e implementar uma extensão para o algoritmo da Programação Genética Distribuída para suportar a evolução de sistemas de controle para populações de robôs móveis. O algoritmo proposto deve permitir, ao mesmo tempo, a evolução individual, ou seja, de um único robô, bem como a evolução embarcada, isto é, a evolução em uma população de robôs móveis.

## 1.5 Objetivos Específicos

Visando atingir o objetivo geral da tese os seguintes objetivos específicos devem ser cumpridos:

1. Propor uma extensão para o algoritmo da Programação Genética Distribuída para suportar a evolução individual do sistema de controle de um único robô, bem como a evolução embarcada de uma população de robôs móveis;

2. Desenvolver um sistema de controle que dê suporte à execução, o gerenciamento e à supervisão do algoritmo proposto em (1) em cada robô;

3. Testar o sistema de controle descrito em (2) nas seguintes tarefas: Navegação Livre de Colisões, Forageamento e Empurrar uma Caixa;

4. Descrever e analisar os resultados obtidos com os experimentos descritos em (3).

## 1.6 Organização da Tese

Além desta Introdução, este documento está organizado em mais 5 (cinco) capítulos com o seguinte conteúdo:

No **Capítulo 2** são descritas as principais características da Computação Evolucionária, bem como os principais tipos de algoritmos evolucionários, que são: Algoritmos Genéticos e Programação Genética. É dada ênfase à Programação Genética, bem como suas variações, a Programação Genética Linear e a Programação Genética Distibuída, por se tratarem das técnicas empregadas no algoritmo e no sistema de controle descritos no Capítulo 4. A última seção do capítulo (Seção 2.4) apresenta um resumo sobre os tópicos abordados no mesmo.

O **Capítulo 3** abordada os principais conceitos envolvendo robótica evolucionária, tais como: origem, abordagens (simulada, embarcada e híbrida) e os critérios para a definição de uma função de aptidão. Na Seção 3.4 são descritos alguns trabalhos relacionados ao objetivo geral desta tese onde são relatados experimentos em robótica evolucionária com o uso de Algoritmos Genéticos e com o uso de Programação Genética. A Seção 3.5 traz um resumo sobre os assuntos abordados no capítulo.

No **Capítulo 4** é descrito o algoritmo da xPGD, uma extensão do algoritmo da Programação Genética Distribuída, capaz de suportar a evolução do sistema de controle de uma população de robôs móveis; e o SEGS (Sistema de Execução, Gerenciamento e Supervisão), o sistema responsável pela execução, gerenciamento e supervisão da xPGD. Também são descritas as principais características e a metodologia utilizada para o desenvolvimento da xPGD e do SEGS, respectivamente. No final do capítulo apresenta-se um pequeno resumo sobre o conteúdo do mesmo.

O **Capítulo 5** apresenta os resultados de 3 (três) experimentos realizados com o algoritmo da xPGD (Extensão da Programação Genética Distribuída), que são: *navegação livre de colisões*, *forageamento* e *empurrar uma caixa*. Os experimentos foram realizados em ambiente simulado nos simuladores do robô Khepera e também do robô Eyebot. Ao final do capítulo são feitas algumas considerações adicionais sobre os experimentos realizados.

No **Capítulo 6** são apresentadas as conclusões sobre o trabalho desenvolvido nesta tese. Também são listadas algumas sugestões para trabalhos futuros que possam expandir e/ou contribuir com o trabalho desenvolvido.

# Capítulo 2

# Computação Evolucionária

Neste capítulo são descritas as principais características da Computação Evolucionária, bem como os principais tipos de algoritmos evolucionários, que são: Algoritmos Genéticos e Programação Genética. É dada ênfase à Programação Genética, bem como suas variações, a Programação Genética Linear e a Programação Genética Distibuída, por se tratarem das técnicas empregadas no algoritmo e no sistema de controle descritos no Capítulo 4. A última seção do capítulo (Seção 2.4) apresenta um resumo sobre os tópicos abordados no mesmo.

## 2.1 Algoritmos Evolucionários

A Computação Evolucionária (CE) é um paradigma da computação bioinspirada, que investiga como computadores podem ser utilizados para modelar a natureza e como soluções encontradas pela natureza podem originar novos paradigmas de computação (Carvalho *et al.*, 2004).

A CE compreende um conjunto de técnicas de busca inspiradas na evolução natural das espécies formalizada por Charles Darwin (Fogel, 2000). Estas técnicas são eficazes, principalmente, na resolução de casos pertencentes às classes de problemas cujos espaços de busca têm um caráter combinatório e portanto de cardinalidade explosiva.

A CE propõe um paradigma alternativo para a resolução de problemas e não exige conhecimento prévio para encontrar uma solução (Whitley *et al.*, 1996). Deve ser entendida como um conjunto de técnicas e procedimentos genéricos e adaptáveis, a serem aplicados na solução de problemas complexos (Spears *et al.*, 1993).

Esta eficiência na solução de problemas complexos é o que a diferencia de outras técnicas conhecidas, que, para esses casos, não são capazes de obter uma solução ótima. Segundo Bittencourt (2006) os algoritmos baseados nesse paradigma possuem características como auto-organização e comportamento adaptativo, no qual uma população de indivíduos se reproduz e compete pela sobrevivência, onde os mais aptos sobrevivem e transferem suas características às novas gerações.

A vantagem mais significativa da CE está na possibilidade de resolver problemas pela simples descrição matemática do que se quer ver presente na solução, não havendo necessidade de se indicar explicitamente os passos até o resultado, que certamente seriam específicos para cada caso (Coelho, 2003).

Em CE existem diferentes tipos de Algoritmos Evolucionários (AEs), cada um possui uma peculiaridade de funcionamento e pode ser aplicado em diferentes tipos de problemas (Coelho, 2003). Assim, todos os AEs têm pelo menos três características em comum: (1) utilizam populações de indivíduos (soluções para o problema); (2) introduzem variação genética na população usando um ou mais operadores genéticos como por exemplo a mutação ou a recombinação; e (3) selecionam, de acordo com a aptidão, os indivíduos que depois se reproduzem para criar a nova geração.

Os AEs apresentam uma seqüência de procedimentos gerais que podem ser adaptados a cada problema. Para Carvalho *et al.* (2004) os dois aspectos seguintes são fundamentais para o desempenho de uma abordagem evolucionária:

- codificação dos indivíduos numa representação genotípica, em que cada indivíduo representa um candidato à solução de um dado problema.

- uma função de avaliação que associe a cada indivíduo um valor de aptidão (*fitness*) que determina o quão apto o indivíduo está ao ambiente.

Nos AEs, os pontos no espaço de busca são modelados através de indivíduos que compõem um conjunto de soluções (população) que são manipulados a cada iteração. A chance de um indivíduo da população ser selecionado para a próxima geração depende da função de aptidão ou *fitness* desse indivíduo.

O algoritmo 2.1, descrito por Eiben e Smith (2003), demonstra os principais passos de um AE.

**Algoritmo 2.1** Algoritmo Evolutivo

**inicialize** a população de candidatos à solução aleatoriamente
**avalie** cada candidato
**while não** condição de parada **do**
  **selecione** os pais
  **recombine** pares de pais
  **mute** os filhos gerados
  **avalie** os novos candidatos
  **selecione** os indivíduos para a próxima geração
**end while**

Um AE mantém uma população de indivíduos criada, incialmente, de maneira aleatória. Cada indivíduo representa um candidato à solução do problema em questão. Os candidatos são testados e recebem um valor de aptidão. Alguns candidatos são selecionados, baseado no valor de aptidão, para gerarem novos candidatos (filhos), através do operador de recombinação. Os filhos gerados sofrem mutação e então uma nova população é formada pela seleção dos indivíduos mais aptos.

Os operadores genéticos podem ser (Spears *et al.*, 1993): reprodução, cruzamento e mutação. O operador genético de reprodução copia indivíduos de uma população para outra. O operador de cruzamento ou recombinação cria novos indivíduos, através da transferência de características entre dois ou mais indivíduos. O operador de mutação é responsável pela criação de novos indivíduos, através da modificação de genes de um dado indivíduo, esperando com isto que estes novos indivíduos estejam mais aptos ao meio.

O critério de parada é atendido quando a população converge para uma solução ou quando o número máximo de gerações é atingido, caso em que o resultado pode indicar uma resposta não satisfatória.

A probabilidade de cruzamento e mutação, dentre os operadores genéticos, e o tamanho da população, são parâmetros importantes para gerar uma diversificação dos indivíduos evitando a convergência prematura [1] do algoritmo (Fogel, 2000).

Os AEs também apresentam limitações e seu desempenho varia de execução para execução. A média da convergência sobre diversas execuções do AE é um indicador de desempenho mais útil que uma simples execução (Coelho, 2003).

Os AEs, nas suas configurações usuais, também apresentam dificuldades para a determinação do ótimo global, sem a utilização de uma metodologia de otimização local. A necessidade de análise de todas as amostras do processo a cada avaliação da função

[1] A convergência prematura é a estagnação da população num ótimo local, distante do ótimo global.

de aptidão é outra limitação dos AEs, o que em muitos casos os torna impraticáveis em aplicações de controle em tempo real.

Em CE existem quatro paradigmas principais de AE (Whitley, 2001), são eles: Estratégias Evolucionárias (Spears *et al.*, 1993), Programação Evolucionária (Fogel, 1992), Algoritmos Genéticos (Holland, 1975) e Progamação Genética (Koza, 1992a). As principais diferenças entre eles dizem respeito ao método usado na seleção e nos operadores genéticos. Entretanto existem outros elementos de abordagem que os tornam diferentes tais como estrutura de dados usadas para codificar um indivíduo e o método usado na geração da população inicial.

As Seções 2.2 e 2.3 descrevem as características e o funcionamento dos Algoritmos Genéticos e da Programação Genética, respectivamente. Esses dois algoritmos são descritos em detalhes pois são os mais utilizados em Robótica Evolucionária (ver Capítulo 3; Seção 3.4). Em particular, a Programação Genética é o algoritmo utilizada no desenvolvimento da solução descrita no Capítulo 4.

## 2.2 Algoritmos Genéticos

Algoritmos Genéticos (AGs) foram formalizados inicialmente pelo professor John Holland em 1975, na Universidade de Michigan (Holland, 1975). O objetivo é gerar a partir de uma população de cromossomos artificiais, outros novos, com propriedades genéticas superiores às de seus antecedentes, onde os indivíduos são as possíveis soluções de um problema.

Os AGs são capazes de evoluir soluções para problemas do mundo real, desde que sejam codificados convenientemente. AGs manipulam uma população de cromossos artificiais usualmente criadas a partir de um processo aleatório (Koza, 1995). Cada indivíduo da população é uma solução candidata ao problema em questão. O processo evolucionário é dirigido por uma função de aptidão.

A função de aptidão associa a cada indivíduo um valor de aptidão (*fitness*). Indivídulos são probabilisticamente selecionados, através de seu valor de aptidão, e a eles são aplicados operadores genéticos de reprodução, recombinação e mutação, dando origem a uma nova população. Não é possível garantir o encontro de uma solução ótima global com AGs, mas esta é uma técnica muito boa para encontrar uma resposta aceitavelmente boa em um tempo computacional aceitável (Whitley, 2001).

### 2.2.1 Codificação

Em AGs cada solução parcial de um problema é codificada em uma *string* (Silveira e Barone, 2003). Cada *string* é um cromossomo que tradicionalmente é representado por um vetor de bits, onde cada elemento do vetor denota a presença (1) ou ausência (0) de uma determinada característica (Whitley, 2001).

### 2.2.2 Função de Aptidão

A função de aptidão é muito importante em um AE pois é ela que determina o quão apto ou próximo da solução do problema o indivíduo está. A função de aptidão associa a cada indivíduo uma medida numérica (*fitness*) que determina o quão apto o indivíduo está (Tomassini, 1995). A função de aptidão pode ser uma das seguintes:

- **Aptidão Nata**: (*raw fitness*) é definida diretamente em função do domínio do problema e normalmente de acordo com um conjunto de casos de teste, que são uma amostra do espaço de busca. O método mais comum de aptidão nata é a avaliação do erro cometido, isto é, a soma de todas as diferenças absolutas entre o resultado obtido pelo indivíduo e o seu valor correto.

- **Aptidão Padronizada**: (*standardized fitness*): está relacionada com o fitness básico e o modifica de forma que um valor mais baixo seja sempre melhor. A avaliação da aptidão padronizada é feita com o valor da aptidão nata de forma que quanto melhor o indivíduo, menor deve ser a aptidão padronizada. Na aptidão padronizada o melhor indivíduo terá o valor de aptidão igual a 0 (zero).

- **Aptidão Ajustada** (*adjusted fitness*): é usada somente em casos onde uma resposta exata é exigida para um problema. A aptidão ajustada é obtida pela aptidão padronizada e pode ser calculada com a Equação 2.1:

$$a(i,t) = \frac{1}{1 + s(i,t)} \tag{2.1}$$

onde $i \in \{1, 2, \ldots, N\}$, $a(i,t)$ é o valor da aptidão ajustada e $s(i,t)$ é o valor da aptidão padronizada do indivíduo $i$ na geração $t$. A aptidão ajustada varia entre os valores 0 (zero) e 1 (um), sendo que os maiores valores representam os melhores indivíduos.

- **Aptidão Normalizada** (*normalized fitness*): é calculada por meio da aptidão ajustada com a Equação 2.2:

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{m} a(k,t)} \tag{2.2}$$

onde $i \in \{1, 2, \ldots, N\}$ e $a(i,t)$ é a aptidão ajustada do indivíduo $i$ da população $t$ com tamanho $m$. A soma de todas as aptidões normalizadas numa dada população vale 1 (um).

### 2.2.3 Métodos de Seleção e Operadores Genéticos

Os métodos de seleção são utilizados para escolher quais indivíduos deverão sofrer a ação dos operadores genéticos e compor uma nova geração. Utilizam como parâmetro de escolha o valor da aptidão de cada indivíduo. Os métodos de seleção mais utilizados são: Seleção Proporcional, Seleção por Torneio, Seleção por Truncamento, Seleção por Nivelamento Linear e Seleção por Nivelamento Exponencial (Blickle e Thiele, 1996).

- **Seleção Proporcional**: um dos principais métodos utilizados em AGs. Seu funcionamento se assemelha a uma roleta, cada indivíduo da população ocupa uma fatia proporcional conforme o valor de sua aptidão normalizada. Assim, para indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos indivíduos de aptidão mais baixa, é dada uma porção relativamente menor. Um número entre 0 (zero) e 1 (um) é produzido aleatoriamente e representará a posição ocupada pelo "ponteiro" da roleta. A seleção proporcional é realizada por meio da Equação 2.3:

$$p_i = \frac{f_i}{\sum_{i}^{N} f_i} \tag{2.3}$$

onde $i \in \{1, 2, \ldots, N\}$, $f_i$ representa o valor de aptidão do indivíduo $i$ e $\sum_{i}^{N} f_i$ representa o valor acumulado de aptidão.

Em geral o melhor indivíduo é copiado para a população seguinte. Este processo é chamado de elitismo. Entretanto, se um indivíduo possui uma alta aptidão comparado com os demais, a probabilidade que ele seja selecionado tende a ser alta, fazendo com que seja selecionado muitas vezes, levando a uma convergência prematura, ou seja, quando o sistema encontra um ótimo local.

- **Seleção por Torneio**: a seleção por torneio é feita a partir da escolha aleatória de $t$ indivíduos da população. Dentre estes indivíduos é escolhido aquele que tiver o melhor valor de aptidão. O processo é repetido até que se tenha uma nova população. O parâmetro $t$ é conhecido como tamanho do torneio. Este método é o mais utilizado, pois oferece a vantagem de não exigir que a comparação seja feita entre todos os indivíduos da população.

- **Seleção por Truncamento**: nesse método a seleção é feita entre os $T$ melhores indivíduos da população. O parâmetro $T$ é um valor de limiar entre 0 (zero) e 1 (um). Por exemplo, se $T = 0.3$, então a seleção é feita entre os 30% melhores indivíduos e os outros 70% são descartados.

- **Seleção por Nivelamento Linear**: os indivíduos são ordenados conforme seus valores de aptidão, o nível $N$ é associado ao melhor indivíduo e o nível 1 ao pior. A cada indivíduo $i$ é associada uma probabilidade $p_i$ de ser selecionado. O valor de $p_i$ é calculado com a Equação 2.4:

$$p_i = \frac{1}{N}(n^- + (n^+ - n^-)\frac{i-1}{N-1}) \tag{2.4}$$

  onde $i \in \{1, 2, \ldots, N\}, n^- \geq 0$ e $n^+ + n^- = 2$.

  Os valores $\frac{n^+}{N}$ e $\frac{n^-}{N}$ representam as probabilidades do melhor e do pior indivíduo ser escolhido, respectivamente. Neste caso, mesmo que dois indivíduos tenham a mesma aptidão, eles terão probabilidades diferentes de serem selecionados.

- **Seleção por Nivelamento Exponencial**: nesse método a probabilidade $p_i$ é calculada exponencialmente. Um parâmetro $c$ entre 0 (zero) e 1 (um) é utilizado como base. Quanto mais próximo de um, menor será a "exponencialidade" da seleção. Como na seleção por nivelamento linear, os indivíduos são ordenados conforme seus valores de aptidão, o nível $N$ é associado ao melhor indivíduo e o nível 1 ao pior. A cada indivíduo $i$ é associada uma probabilidade $p_i$ de ser selecionado. O valor de $p_i$ é calculado com a Equação 2.5:

$$p_i = \frac{c-1}{c^{N-1}}c^{N-i} \tag{2.5}$$

onde $i \in \{1, 2, \ldots, N\}$.

Após a seleção dos indivíduos alguns operadores genéticos podem ser aplicados com o objetivo de gerar uma nova população. O princípio básico dos operadores genéticos é transformar a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório.

Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores. A Figura 2.1 ilustra a operação de cruzamento e mutação em AGs.

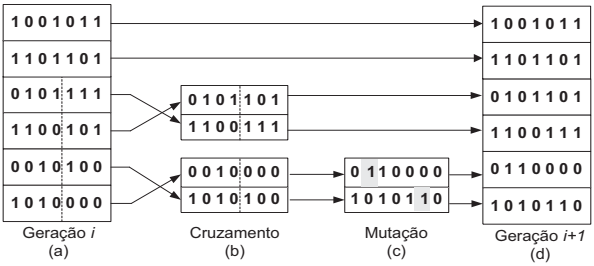


Figura 2.1: Exemplo dos operadores de cruzamento e mutação em AGs.

A Figura 2.1 (a) ilustra um conjunto de indivíduos pertencentes a geração $i$ que darão origem aos indivíduos da geração $i + 1$ (Figura 2.1 (d)). Na Figura 2.1 (b), os indivíduos um e dois, e, três e quatro, cruzaram e geraram novos indivíduos. Na Figura 2.1 (c) houve a mutação genética em dois dos novos indivíduos gerados.

Em AGs o operador de cruzamento pode ser de um ponto, multipontos e uniforme. No primeiro caso, um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais são trocadas. No multipontos, vários pontos são escolhidos e a troca de informações genéticas se dá a partir destes pontos. No cruzamento uniforme um parâmetro global determina a probabilidade de cada variável ser trocada entre os pais.

### 2.2.4 Critérios e Parâmetros de Controle

Os parâmetros de controle de um AG são o tamanho da população, $M$, o número máximo de gerações a ser executadas, $G$, e as taxas de mutação e cruzamento (Tomassini, 1995). O tamanho da população pode afetar o desempenho global e a eficiência dos AGs. Populações

muito pequenas têm grandes chances de perder a diversidade necessária para convergir a uma boa solução, pois fornecem uma pequena cobertura do espaço de busca do problema.

Em uma população com muitos indivíduos, o algoritmo poderá perder grande parte de sua eficiência pela demora em avaliar a função de aptidão de todo o conjunto a cada iteração, além de ser necessário trabalhar com maiores recursos computacionais (Coelho, 2003).

Cada execução do AG requer um critério de parada para decidir quando a execução deve terminar. Um critério de parada pode ser estabelecido de acordo com o número máximo de gerações, nesse caso quando tal número for atingido, ou quando a população convergir.

### 2.2.5 SAGA - *Species Adaptation Genetic Algorithm*

SAGA foi proposto por Harvey (1992) para resolver, entre outros, o problema da convergência prematura (problema do *bootstrap*) e possiblitar o uso de genótipos de tamanhos variados em AGs. No SAGA a evolução é um processo incremental que não termina necessariamente quando se obtém um indivíduo bem adaptado em determinada tarefa. A mesma espécie, após a convergência, pode continuar a sua evolução para tarefas mais complexas através do melhoramento adaptativo.

SAGA não é uma técnica de otimização destinada a resolver um problema específico com um espaço de busca bem definido em termos de um número fixo de parâmetros. De acordo com Harvey (1997) a evolução deve ser vista como uma melhora incremental e não como um otimizador.

No SAGA a população inicial já possui convergência genética, isto é, tem um grau de variabilidade genotípica pequeno. A idéia é evoluir uma espécie, possibilitando o aumento do tamanho do genótipo, ou seja, do número de parâmetros que codificam as características da espécie, ao invés do genótipo de dimensões fixas dos AGs tradicionais.

A vantagem do SAGA sobre os AGs tradicionais é sua aplicação a problemas cujo número de componentes, inicialmente, são desconhecidos e que poderão aumentar ao longo do tempo à medida que a complexidade avança. Ao invés de tratar os AGs como otimizadores, o SAGA permite tratá-los como melhoradores do processo de adaptação.

### 2.2.6 Microbial GA

O Microbial GA (Harvey, 1996) é uma simplificação do AG tradicional. O funcionamento do Microbial GA é semelhante a recombinação (infecção) genética que acontece nas bactérias onde segmentos do DNA (*Deoxyribonucleic Acid*) são transferidos entre dois membros da população.

A reprodução no microbial GA é realizada da seguinte forma: dois indivíduos são selecionados aleatoriamente na população, parte do material genético do indivíduo menos apto é sobrescrito pelo material genético do mais apto. Nessa abordagem o menos adaptado sempre será o filho.

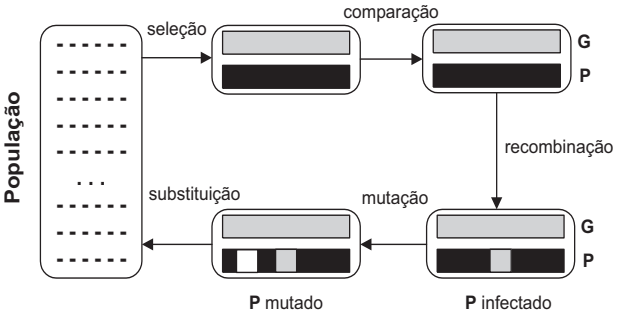


Figura 2.2: Reprodução no Microbial GA.

A Figura 2.2 ilustra o processo de reprodução no Microbial GA. Dois indivíduos da população são selecionados, após a seleção os valores de aptidão de cada indivíduo são comparados para determinar um ganhador (G) e um perdedor (P). O cromossomo do perdedor é "infectado" por uma parte, escolhida aleatoriamente, do cromossomo do ganhador. Após a infecção, um gene do cromossomo do perdedor, escolhido aleatoriamente, é modificado pelo operador de mutação. Após esse processo, os dois indivíduos, o ganhador e o perdedor, voltam para a população.

## 2.3 Programação Genética

A programação genética (PG) é uma técnica de geração automática de programas de computador criada por John Koza (Koza, 1992a), inspirada na teoria de AGs de John Holland. A idéia é ensinar computadores a se programar, isto é, a partir de especificações

de comportamento, o computador deve ser capaz de evoluir um programa que as satisfaça (Koza, 1992b).

Em PG um conjunto de programas de computador, que são soluções candidatas para resolver um determinado problema, são submetidos a um processo evolucionário. Através do processo evolucionário é possível encontrar soluções (programas ou sub-rotinas) que resolvam parte ou todo o problema proposto. A cada programa é associado um valor de mérito (*fitness*) representando o quanto ele é capaz de resolver o problema.

Basicamente a PG mantém uma população de programas de computador, usa métodos de seleção baseados na capacidade de aptidão de cada programa, aplica operadores genéticos para modificá-los e convergir para uma solução. Nessa técnica programas mais aptos sobrevivem e são combinados para gerarem soluções ainda melhores.

O mecanismo de busca da PG pode ser descrito como um ciclo criar-testar-modificar (Rodrigues, 2002). Inicialmente, programas são criados baseados no conhecimento sobre o domínio do problema. Em seguida, são testados para verificar sua funcionalidade. Se os resultados não forem satisfatórios, modificações são feitas para melhorá-los.

O ciclo criar-testar-modificar é repetido até que uma solução satisfatória seja encontrada ou um determinado critério seja satisfeito (Yu, 1999). O objetivo é encontrar uma solução no espaço de todos os programas possíveis (candidatos) usando apenas um valor da aptidão como auxílio no processo de busca.

De acordo com Koza (1997) o algoritmo de Programação Genética pode ser descrito resumidamente como:

1. Criar aleatoriamente uma população de programas;
2. Executar iterativamente os seguintes passos até que algum critério de parada seja satisfeito:
   (a) Avaliar cada programa da população através de uma função heurística (*fitness*), que expresse a sua aptidão, ou seja, o quão próximo o programa está da solução ideal;
   (b) Selecionar os melhores programas com base no valor da aptidão;
   (c) Aplicar aos programas selecionados operadores genéticos de reprodução, cruzamento e mutação;

3. Retornar com o melhor programa encontrado.

O processo inicia-se com a geração aleatória de uma população inicial de programas compostos de funções e terminais apropriados para o domínio do problema. Cada programa é avaliado em termos de sua aptidão em relação ao problema. O princípio da reprodução e sobreviência do mais apto é então aplicado para a criação de uma nova população a partir da população atual.

Cada execução representa uma nova geração de programas. Um critério de término é estabelecido para encontrar uma solução satisfatória ou atingir um número máximo de gerações (Koza, 1992a). Abordagens baseadas na análise do processo evolucionário também podem ser usadas como critério de término, neste caso, o laço permanece enquanto houver melhoria na população (Kramer e Zhang, 2000). A Figura 2.3 ilustra o diagrama de blocos do algoritmo da PG.

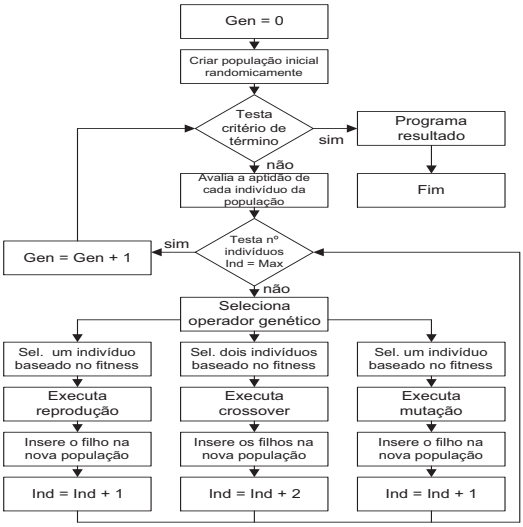


Figura 2.3: Diagrama de blocos do algoritmo da PG. Adaptado de (Martin, 2003).

Programas encontrados nas gerações iniciais tendem a ser menos aptos que indivíduos nas gerações seguintes. A cada nova geração criada os indivíduos tendem exibir melhor aptidão devido às operações de cruzamento e mutação usadas para a criação dos indivíduos que compõem a nova geração.

A única informação disponível para evolução dos programas é o valor da aptidão de cada indivíduo, não existem outras informações. O pré-processamento de entradas ou pós-processamento de saída não é necessário ou exerce uma influência menor, uma vez que entradas, resultados intermediários e saídas são expressas diretamente em termos de instruções, ou seja, os programas em PG são estruturas em sub-rotinas de acordo o domínio do problema (Franzen e Barone, 2003).

### 2.3.1 Estrutura dos Programas em PG

Em PG não é necessário fornecer todos os passos necessários para a resolução do problema. São definidas apenas as instruções básicas necessárias e as regras do processo evolucionário e o sistema se encarrega de descobrir como e qual a seqüência de instruções que devem ser utilizadas. Os programas são estruturados como uma árvore de sintaxe abstrata composta por funções em seus nós internos e por terminais em seus nós folha.

O domínio do problema é especificado através da definição dos conjuntos de funções, $F(f_1, f_2, f_3, \ldots, f_n)$ e um conjunto de terminais, $T(t_1, t_2, t_3, \ldots, t_m)$ (Koza, 1992b). O espaço de busca é determinado por todas as árvores que possam ser criadas pela livre combinação de elementos dos conjuntos $F$ e $T$.

Cada função $F$ requer um número de argumentos e pode conter operadores aritméticos $(+, -, *, etc)$, funções matemáticas $(seno, coseno, log, etc)$, operadores lógicos $(E, OU, NAO, etc)$, dentre outros. Cada $f \in F$ tem associada uma aridade (número de argumentos) superior a zero. O conjunto $T$ é composto pelas variáveis, constantes e funções de aridade zero (sem argumentos).

O número total de nós de um programa em PG é o conjunto formado pelos membros de $F$ e os membros de $T$ descrito como $C = F \cup T$. O espaço de busca é o conjunto de todas as composições possíveis entre os membros de $C$.

A Figura 2.4 ilustra a representação de um programa genético com suas respectivas funções e terminais.

Para que a PG tenha um funcionamento correto (Koza, 1992b) definiu duas propriedades que devem ser satisfeitas na escolha do conjunto de funções e terminais aplicáveis ao problema em questão.

- **Fechamento**: cada membro do conjunto $F$ deve aceitar, como seus argumentos, qualquer membro de $C$. Esta imposição garante que qualquer árvore gerada pode

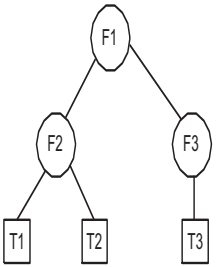


Figura 2.4: Estrutura de um programa em PG.

ser avaliada corretamente. Um caso típico de problema de fechamento é a operação de divisão. Matematicamente, não é possível dividir um valor por zero. Uma abordagem possível é definir uma função alternativa que permita um valor para a divisão por zero. É o caso da função de divisão protegida (*protected division*) % proposta por (Koza, 1992b). A função % recebe dois argumentos e retorna o valor 1 (um) caso seja feita uma divisão por zero e, caso contrário, o seu quociente.

- **Suficiência**: o conjunto de funções $F$ e o de terminais $T$ devem ser capazes de representar uma solução para o problema. Isto implica que o conjunto de funções e terminais escolhidos deve ser capaz de resolver o problema, ou seja, deve existir uma forte evidência de que alguma composição de funções e terminais possa produzir uma solução. Dependendo do problema, esta propriedade pode ser óbvia ou exigir algum conhecimento prévio de como deverá ser a solução.

Ao longo da busca da solução a variabilidade dinâmica dos programas é uma característica importante da PG, regras e instruções simples, combinadas e submetidas a processos evolucionários podem gerar, após um determinado período de tempo, um comportamento capaz de resolver problemas de diversos domínios. É extremamente difícil restringir o tamanho e a forma das soluções encontradas, pois tais restrições poderiam acarretar a diminuição da probabilidade de encontrarem-se melhores soluções.

Para uma correta aplicação da PG em problemas que imponham restrições sintáticas Gruau (1996) propôs o uso de gramáticas. Nesse caso as regras para formação do programas são informadas juntamente com os conjuntos $T$ e $F$.

### 2.3.2 Criação da População Inicial

A população inicial é composta por árvores geradas aleatoriamente a partir dos conjuntos das funções $F$ e dos terminais $T$. O processo inicia-se com a escolha aleatória de uma função $f$. Para cada um dos argumentos de $f$, escolhe-se um elemento de $C$. O processo prossegue até que se tenha apenas terminais como nós-folha da árvore. Para se evitar árvores muitos grandes[2], usualmente se especifica um limite máximo de profundidade.

Segundo Daida (1999) a qualidade da população inicial é um fator crítico para o sucesso do processo evolutivo. A população inicial deve ser uma amostra significativa do espaço de busca, apresentando uma grande variedade de composição nos programas, para que seja possível, através da recombinação de seus códigos, convergir para uma solução.

Existem diversos métodos (algoritmos) usados para melhorar a qualidade da população inicial gerada. A seguir é feita uma revisão dos métodos mais utilizados em PG onde são descritas suas principais características.

- **Ramped Half and Half**: esse método foi proposto por (Koza, 1992a) e é uma variação de dois outros métodos: o *Grow* e o *Full*. O método *grow* cria árvores cuja profundidade é variável respeitando uma profundidade máxima. A escolha dos nós é feita aleatoriamente entre funções e terminais. O método *full* cria árvores com a mesma profundidade. A escolha dos nós é realizada através da seleção de funções e terminais cuja profundidade seja inferior a desejada. O algoritmo *Ramped Half and Half* cria árvores baseado em um valor aleatório que está entre uma faixa de valores para a profundidade máxima, geralmente entre 2 (dois) e 6 (seis) nós. Supondo que a profundidade máxima seja 6 (seis), serão criadas árvores com as profundidas 2, 3, 4, 5 e 6, respectivamente. Sendo que metade das árvores serão criadas usando o algoritmo *Grow* e a outra metade usando o algoritmo *Full*. O usuário não tem controle sobre o formato ou o tamanho das árvores criadas.

- **PTC1**: *Probabilistic Tree Creation 1* ou PTC1 (Luke, 2000) é uma variação do algoritmo *Grow* e garante a geração de árvores com um tamanho finito esperado. Para cada terminal $t \in T$ calcula-se a probabilidade $q_t$ dele ser escolhido, o mesmo acontece com cada função $f \in F$, calculando-se a probabilidade $q_f$. O PTC1 recebe o pedido para criação das árvores com um tamanho máximo esperado $E_{tree}$. O

[2]O tamanho de uma árvore é determinado pela quantidade de elementos ($F$ e $T$) que esta possui.

PTC1 inicia computando o valor de $p$, a probabilidade de escolher uma função ao invés de um terminal para manter o valor esperado $E_{tree}$, através da Equação 2.6:

$$p = \frac{1 - \frac{1}{E_{tree}}}{\sum_{f \in F} q_f b_f} \tag{2.6}$$

onde $F$ é o conjunto de todas a funções e $b_f$ representa a aridade de cada função. Este cálculo pode ser realizado um única vez antes do início da execução do algoritmo da PG. A complexidade computacional do PTC1 é linear.

- **PTC2**: o PTC2 (Luke, 2000) é uma variação do PTC1 e trabalha aumentando a árvore horizontalmente escolhendo pontos aleatórios até que a árvore esteja suficientemente larga. Ao receber uma requisição para a criação de uma árvore, o PTC2 garante que retornará uma árvore menor que a largura máxima e menor que a aridade máxima de qualquer função do conjunto de funções $F$. O PTC2 trabalha com um parâmetro que define um tamanho máximo $S$ para a criação das árvores e uma distribuição de probabilidades $w_1, w_2, \ldots w_s$ para cada árvore de tamanho 1 a $S$. Com o PTC2 é possível ter o controle sobre e tamanho máximo esperado para as árvores e a distribuição de probabilidades. Assim como o PTC1, o PTC2 é de complexidade computacional linear.

- **Random-Branch**: (Chellapilla, 1997) permite que se informe o tamanho máximo $S$ da árvore e não sua profundidade. O random-branch divide igualmente $S$ entre árvores de um nó pai não-terminal (função). Essa característica faz com que seu uso seja muito restritivo, devido ao fato de muitas árvores não serem produzidas.

- **Uniform**: (Bohm e Geyer-Schulz, 1996), garante que as árvores serão geradas uniformemente do conjunto de todas as árvores possíveis. O algoritmo necessita calcular em várias tabelas auxiliares o número de árvores possíveis de serem geradas para cada tamanho desejado. Apesar de ter um um alto custo computacional esse método é de complexidade polinomial.

Luke e Panait (2001) fazem uma avaliação do impacto que os algoritmos descritos acima podem causar na determinação do valor de aptidão dos indivíduos de uma população. A avaliação foi feita com base três problemas usando programação genética:

Multiplexação Booleana, Regressão Simbólica e Formiga Artificial. Os resultados demonstraram que indiferentemente do algoritmo utilizado para a geração da população inicial não influencia na determinação do valor de aptidão dos indivíduos da população.

### 2.3.3 Função de Aptidão

A função de aptidão é fundamental na PG e dependente do domínio do problema (Burke *et al.*, 2004). Usualmente para se proceder a avaliação da aptidão de um indivíduo, é fornecido um conjunto de casos de treinamento, chamado de *fitness cases*, contendo os valores de entrada e saída a serem aprendidos (Rodrigues, 2002). Os valores de entrada são fornecidos para cada indivíduo e sua resposta é confrontada com o valor esperado da saída. Quanto mais próxima a resposta do indivíduo estiver do valor de saída, melhor é o indivíduo.

Em PG, assim como nos AGs, a função de aptidão, dependendo do domínio do problema, pode ser uma das que foram descritas na Seção 2.2.2.

### 2.3.4 Métodos de Seleção e Operadores Genéticos

Os métodos de seleção utilizados em PG são os mesmos que foram descritos na Seção 2.2.3 para AGs. Os operadores genéticos mais utilizados em PG são: reprodução, cruzamento e mutação.

- **Reprodução**: a operação de reprodução seleciona um indivíduo conforme sua aptidão e copia esse indivíduo para a nova geração sem modificá-lo.

- **Cruzamento**: por meio deste operador são criados novos indivíduos misturando características de dois indivíduos "pais". Partes de um indivíduo são trocadas pelas partes de um outro. O resultado desta operação é um indivíduo que potencialmente combine as melhores características dos indivíduos usados como base. São escolhidos dois indivíduos para uma reprodução sexuada, em cada um dos pais é escolhido aleatoriamente um ponto de cruzamento. A subárvore existente a partir do ponto de cruzamento do primeiro pai é inserida no ponto de cruzamento do segundo pai e vice-versa. A Figura 2.5 ilustra um exemplo do uso do operador de cruzamento.

  A operação de cruzamento modifica o tamanho dos programas, pois ao contrário do cruzamento em AGs onde um ponto de cruzamento é igual nos dois pais, na PG o

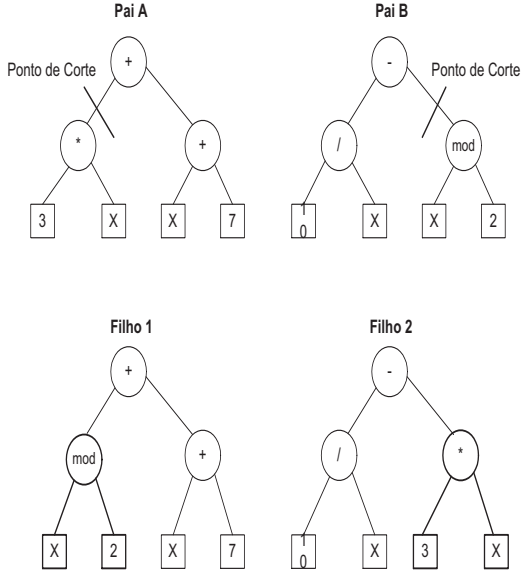


Figura 2.5: Exemplo do operador de cruzamento.

ponto selecionado é, em geral, diferente para cada um dos pais. Se em ambos os pontos de cruzamento existirem terminais, o efeito de uma operação de cruzamento é igual a uma mutação de um ponto (Koza, 1992b).

- **Mutação**: o operador de mutação modifica aleatoriamente alguma característica do indivíduo sobre o qual é aplicado visando restaurar o material genético perdido ou não explorado em uma população. Esta alteração é importante, pois acaba por criar novos valores de características que não existiam ou apareciam em pequena quantidade na população em análise. A Figura 2.6 ilustra um exemplo de aplicação do operador de mutação.

  O operador de mutação é necessário para a existência da diversidade genética da população ao longo da evolução. A mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca possivelmente não será zero (Koza, 1995). O operador de mutação é aplicado aos indivíduos através de uma taxa de mutação geralmente pequena. Este operador, quando projetado de forma apropriada, pode prevenir a convergência prematura para soluções sub-ótimas e manter

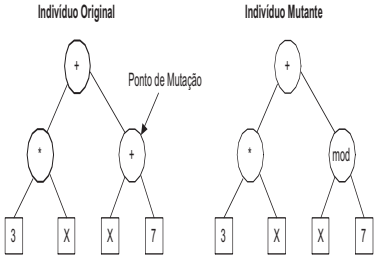


Figura 2.6: Exemplo do operador de mutação.

a diversidade da população (Carvalho *et al.*, 2004).

Em PG existem outros operadores genéticos que são (Koza, 1992b): **Edição**, este operador é utilizado para simplificar expressões. Pode ser aplicado tanto na saída, para facilitar a visualização dos programas, como na execução, para reduzir o tamanho dos programas. A desvantagem deste operador é que consome muito tempo de processamento; **Permutação** também conhecido como inversão, faz a permutação entre os ramos de um nó selecionado aleatoriamente; **Destruição**, este operador é utilizado para reduzir o número de indivíduos nas primeiras gerações. Essa redução é baseada na valor de aptidão de cada indivíduo, nesse caso é estabelecido um percentual que determina quantos indivíduos deve permanecer na população, o restante é eliminado pelo operador de destruição; **Encapsulamento**, utilizado para proteger sub-árvores potencialmente úteis. Nesse caso, seleciona-se uma indivíduo, conforme seu valor de aptidão, seleciona-se um ponto na estrutura deste indivíduo, a sub-árvore formada a partir deste ponto é removida e a ela é dado um nome, formando um nova função. Na estrutura do indivíduo é inserida uma referência a essa nova função. A nova função formada pelo operador de encapsulamento não sofre os efeitos dos operadores de cruzamento e mutação, ou seja, a sub-árvore que forma tal função nunca pode ser modificada por tais operadores.

### 2.3.5 Critérios e Parâmetros de Controle

A aplicação da PG na solução de problemas requer alguns cuidados com certos parâmetros e configurações. Dois parâmetros importantes são o tamanho da população e a quantidade de gerações. É necessário que se estabeleça um bom equilíbrio entre estes dois parâmetros, pois populações pequenas com um número reduzido de gerações podem levar a soluções

locais. Em contrapartida uma população grande e um número de gerações exagerada podem acarretar problemas de desempenho (Kinnear, 1994).

Outros parâmetros importantes estão relacionados a probabilidade do uso dos operadores genéticos de reprodução, cruzamento e mutação. A taxa de uso para cada um dos operadores genéticos deve ser definida separadamente. Reprodução e mutação devem ter taxas pequenas em relação ao operador de cruzamento (Luke e Spector, 1998).

A forma de representação da estrutura dos programas em PG é outro importante critério a ser analisado. Perkis (1994) sugere representar os programas em PG em uma estrutura de dados do tipo pilha. Em estruturas de dados do tipo árvore a quantidade de nós e a profundidade das árvores são valores definidos de acordo com o algoritmo usado para a criação da população inicial. Rosca (1996) faz um estudo sobre a influência na eficiência do processo de busca em relação ao tamanho dos programas e a generalidade da solução.

O critério de parada também é outro importante parâmetro a ser considerado na PG. Um bom parâmetro de término é limitar o número máximo de gerações até que uma solução satisfatório seja encontrada. Uma solução é considerada satisfatória quando a população convergiu, ou seja, encontrou uma solução que satisfaça os requisitos impostos pela função de aptidão. Kramer e Zhang (2000) sugerem que o critério de término pode ser baseado no acompanhamento do processo evolucionário, ou seja, enquanto houver melhoria na média da população, o processo evolucionário prossegue.

### 2.3.6 Definição Automática de Funções

Em muitos casos as melhores soluções para problemas mais difíceis tendem a ser hierárquicas por natureza. A abordagem dividir e conquistar tem sido usada por humanos para lidar com problemas complexos. A Definição Automática de Funções (DAF) fornece à PG um mecanismo para automaticamente decompor um problema em problemas menores e, então, usar a composição das soluções dos problemas menores para solucionar o problema original.

As DAFs representam um mecanismo proposto por John Koza para facilitar a criação e reutilização de módulos (Koza, 1994). Uma DAF é uma função que é desenvolvida durante o processo evolucionário e pode ser chamada pelo próprio programa ou por outra função que esteja sendo desenvolvida simultaneamente. Com DAF a estrutura de uma

solução passará a ser composta de um programa principal e uma ou mais sub-rotinas ou funções.

A DAF é uma extensão à PG tradicional e tem por objetivos permitir à técnica resolver problemas mais complexos através da descoberta de sub-rotinas ou funções que podem ser reutilizadas para resolução de parte do problema.

A técnica se baseia na divisão do problema em problemas menores, mais fáceis de serem resolvidos. Estes subproblemas são resolvidos separadamente e então acha-se uma forma de combinar suas soluções parciais para se obter a solução para a instância original

Uma DAF é uma função que pertence a um programa unicamente. Diferente de outras abordagens, as DAFs não são compartilhadas. Cada DAF tem o seu próprio conjunto de funções $F$ e de terminais $T$ .

Outras extensões foram propostas para prover a Programação Genética da capacidade de evolução de funções, tais como: MA (*Module Acquisition*) (Angeline e Pollack, 1993), ADM (*Automatically Defined Macros*) (Spector, 1995) e ARL (*Adaptive Representation through Learning*) (Rosca, 1997).

### 2.3.7 Programação Genética Linear

Na Programação Genética Linear (PGL) os programas são estruturados de maneira linear, usando uma linguagem de programação imperativa como C, C++, Pascal, ao invés de estruturas em árvores com uma linguagem de programação funcional como LISP, como é o caso da PG tradicional (Broodier e Banzhaf, 2006).

O uso de estruturas lineares para representar programas em PG remontam aos trabalhos de Cramer (1985) e sua linguagem JB. JB é uma linguagem para evoluir programas usando operadores genéticos como mutação, cruzamento e inversão. As instruções de JB são baseadas na linguagem de programação PL, sendo cada instrução identificada com um identificador numérico único.

Em PGL um indivíduo (programa) pode ser representado por um seqüencia variada de simples instruções em linguagem de programação C. As instruções podem operar sobre uma ou mais variáveis indexadas, v, ou sobre constantes, c, definidas nos conjuntos $F$ e $T$, respectivamente. A Figura 2.7 ilustra um exemplo de um programa em PGL.

Brameier e Banzhaf (2001) usaram em seus trabalhos uma estrutura similar ao da Figura 2.7, entretanto foi adicionada a capacidade de remoção de instruções ineficientes,

```
void PGL(double v[8]) {
   v [0] = v[5] + 73;
   v[7] = v[3] . 59;
   if (v[1] > 0)
   v[4] = v[2] . v[1];
   v[2] = v[5] + v[4];
   v[1] = sin(v[6]);
   if (v[0] > v[1])
   v[3] = v[5] . v[5];
   v[5] = v[7] + 115;
   if (v[1] <= v[6])
   v[1] = sin(v[7]);
}
```

Figura 2.7: Exemplo de um programa em PGL. Adaptado de Oltean e Grosan (2003a).

chamadas de *introns*. Tais instruções são dispensáveis e não afetam o comportamento do programa. Por exemplo, uma instrução do tipo *if v[0] > v[0]*, é uma instrução dispensável, pois a condição nunca será satisfeita, nesse caso a instrução ou instruções dependentes desta nunca serão executadas.

Nordin e Banzhaf (1995) propuseram uma técnica de PGL baseada na evolução de instruções de máquina que foi chamada de CGP (*Compiling Genetic Programming*). Em CGP, os indivíduos são manipulados diretamente na memória como código de máquina não necessitando de um interpretador. Essa característica faz com que o CGP tenha um bom desempenho de execução, principalmente quando comparado com soluções que adotam estruturas do tipo árvore.

Existem diferentes variações da PGL, tais como: MEP (*Multi-Expression Programming*) (Oltean e Dumitrescu, 2002), GE (*Gramatical Evolution*) (C. Ryan *et al.*, 1998), GEP (*Gene Expression Programming*) (Ferreira, 2001) e IFGP (*Infix Form Genetic Programming*) (Oltean e Grosan, 2003b). Oltean e Grosan (2003a) e Oltean *et al.* (2009) fizeram um experimento com o objetivo de avaliar e comparar o desempenho de quatro variações da PGL. O problema escolhido para tal foi a regressão simbólica.

### 2.3.8 Programação Genética Distribuída

Existe uma variação da Programação Genética, chamada de Programação Genética Distribuída - PGD, que dá suporte a execução paralela dos indivíduos que fazem parte de uma população de programas candidatos à solução do problema. Essa execução paralela se dá pela distribuição dos programas em vários nós processadores.

Na literatura especializada não existe um consenso sobre como deve ser a imple-

mentação da PGD. Existem diferentes variações da PGD aplicadas aos mais diversos tipos de problemas. Por exemplo, em (Thomas Weise, 2006), é apresentado um framework para o desenvolvimento de aplicações para redes de sensores sem fio baseadas em PGD.

Em (Fernández *et al.*, 2002) é realizado um estudo sobre a influência da topologia da comunicação e a migração de programas no cálculo do valor da aptidão de cada indivíduo da população de programas. Os resultados encontrados demonstraram que a migração de programas tem maior influência no valor da aptidão do que a topologia de comunicação.

Cheang e Leung (2006) propoem uma extensão da PG para ser executada em máquinas paralelas do tipo MIMD (*Multiple Instruction-streams, Multiple Data-streams*). O algoritmo criado por Cheang e Leung (2006), chamado de Programação Genética Paralela (PGP), foi testado com 14 (quatorze) tipos de problemas distintos. Os resultados obtidos monstraram que a PGP evolui mais rapidamente os programas em PG.

## 2.4 Resumo do Capítulo

Nesta capítulo foram apresentadas as principais características da Computação Evolucionária (CE). A CE compreende um conjunto de técnicas de busca eficazes para a solução de problemas que são inspiradas na evolução natural das espécies. As técnicas da CE são utilizadas para encontrar uma solução aceitável para problemas cujo espaço de busca têm um caráter combinatório, de cardinalidade explosiva.

Na CE existem diferentes tipos de Algoritmos Evolucionários (AEs) com peculiaridades distintas. Todos os AEs trabalham com populações de indivíduos, onde cada indivíduo representa um candidato à solução do problema. A cada iteração os indivíduos são avaliados em função de sua aptidão, os melhores são selecionados para formarem uma nova população no próxima passo de iteração (geração).

Os Algoritmos Genéticos (AGs) e a Programação Genética (PG) são dois tipos de AEs bastante utilizados, principalmente na área de robótica móvel. Uma AG codifica um indivíduo (candidato à solução do problema), geralmente, por meio de uma estrutura de dados do tipo vetor. O conteúdo do vetor pode indicar a ausência (0) ou a presença (1) de uma determinada característica. O AG pode encontrar uma boa solução para um determinado problema em um tempo computacional aceitável.

A PG difere de um AG na maneira de representar os indivíduos da população. Em PG um indivíduo é um programa de computador, formado por funções e terminais (sub-rotinas) que são manipuladas iterativamente até que uma boa solução seja encontrada. Por essa característica a PG pode se tornar uma pouco menos eficiente que o AG sob o ponto de vista da execução. Uma maneira de contornar essa questão é utilizar uma estrutura vetorial para representar os indivíduos. Essa técnica é conhecida como PG Linear.

Outra variação da PG é a Programação Genética Distribuída (PGD). A PGD aumenta o desempenho da PG, uma vez que o processo evolucionário ocorre paralelamente em diferentes nós processadores.

No Capítulo 4 é descrito um novo algoritmo evolucionário, baseado nos algoritmos da PGD e do Microbial GA. Este algoritmo é aplicado em problemas de robótica móvel, mais precisamente, em problemas de evolução embarcada, onde um grupo de robôs necessitam executar uma determinada tarefa.

# Capítulo 3

# Robótica Evolucionária

Neste capítulo são abordados os principais conceitos envolvendo robótica evolucionária, tais como: origem, abordagens (simulada, embarcada e híbrida) e os critérios para a definição de uma função de aptidão. Na Seção 3.4 são descritos alguns trabalhos relacionados ao objetivo geral desta tese onde são relatados experimentos em robótica evolucionária com o uso de Algoritmos Genéticos e com o uso de Programação Genética. A Seção 3.5 traz um resumo sobre os assuntos abordados no capítulo.

## 3.1 Robótica Evolucionária versus Robótica Baseada em Comportamentos

A Robótica Evolucionária (RE) surgiu da utilização de técnicas de computação evolucionária para sintetizar automaticamente controladores para robôs com o propósito de treiná-los para desenvolver tarefas específicas (Simões, 2000). A RE compartilha muitas características de outras áreas tais como: Vida Artificial (*ALife)*, Aprendizado em Robôs e Robótica Baseada em Comportamento.

Tanto na RE como na Robótica Baseada em Comportamentos (RBC) (Arkin, 1998) a ativação dos comportamentos é feita conforme a interação do robô com o ambiente. A diferença entre as duas técnicas é que na RBC o projetista define alguns comportamentos básicos e constantemente avalia o resultado do comportamento global. Comportamentos básicos são adicionados conforme a necessidade. Já na RE o comportamento global é constantemente avaliado e modificado por um processo de avaliação automático (Nolfi e Floreano, 2001). A Figura 3.1 ilustra a diferença entre as duas abordagens.

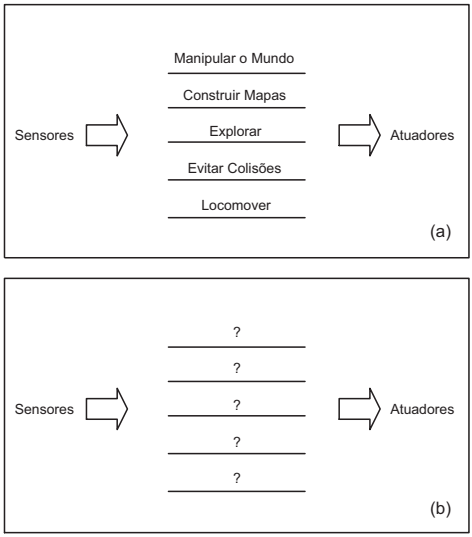


Figura 3.1: Comparação entre RBC (a) e RE (b) (Nolfi e Floreano, 2001).

Na Figura 3.1 (a), um exemplo de RBC, todos os comportamentos básicos descritos são intuitivamente projetados pelo programador ou projetista. Na RE, Figura 3.1 (b), a organização inteira do sistema evolucionário, incluindo sua organização em subcomponentes, é resultado de um processo de evolução que usualmente envolve um grande número de interações entre o sistema e o ambiente.

A Figura 3.2 esquematiza o processo de avaliação na RE. Uma população inicial de cromossomos artificiais, que codificam o sistema de controle do robô, é aleatoriamente gerada e testada no ambiente. Cada robô (físico ou simulado) é colocado no ambiente com o objetivo de realizar alguma tarefa e é avaliado com relação à aptidão para resolver a tarefa. Os melhores indivíduos, ou seja, os robôs com melhor desempenho, são escolhidos para dar origem a uma nova população. Seus genótipos são mantidos ou modificados por meio de operadores genéticos (mutação e reprodução). O processo é repetido para um número $N$ de gerações até que um indivíduo satisfaça algum critério de desempenho pré-estabelecido.

A computação evolucionária oferece meios para automatizar a geração de novas competências em agentes autônomos, por exemplo, um robô móvel situado em um determinado ambiente. As principais questões abordas pela computação evolucionária são (Simões,

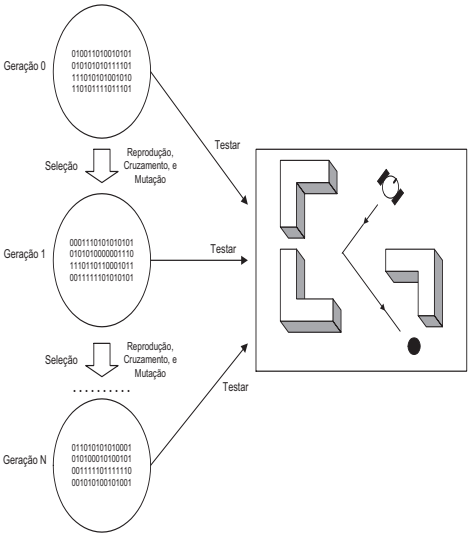


Figura 3.2: Visão geral da robótica evolucionária.

2000):

1. Sintetizar automaticamente comportamentos mais complexos do que aqueles que podem ser produzidos manualmente;

2. Explorar amplamente as características do ambiente e dos indivíduos, mesmo que algumas delas sejam obscuras ao projetista;

3. Produzir o comportamento esperado especificando-se o que o robô deve fazer e não como ele deve operar;

4. Mostrar que técnicas evolucionárias podem reduzir o esforço humano necessário para produzir um sistema de controle em comparação com métodos manuais de projeto.

A RE pode ser descrita como um processo de evolução contínua de um robô situado em um determinado ambiente. Todos os seus componentes estão sujeitos a um processo de evolução artificial, partindo de uma disposição aleatória de certos elementos primitivos (Elfwing, 2007). A emergência de um fenótipo mais bem adaptado ocorre sob pressão de mecanismos de seleção, resultantes da interação com o meio ambiente e com outros sistemas (Wahde, 2004).

## 3.2 Função de Aptidão

Num processo de evolução artificial a função de aptidão é usada para avaliar o desempenho dos indivíduos e selecionar os mais aptos. O resultado do processo de evolução depende muito do formato da função de aptidão (Floreano e Urzelai, 2000).

Algumas abordagens definem a função de aptidão em termos de variáveis e requisitos de desempenho com base em um determinado comportamento. O grau de conhecimento de um determinado comportamento é inversamente proporcional à motivação para o uso de técnicas de evolução artificial. Variáveis e requisitos de desempenho, para um determinado comportamento, são difíceis de escolher porque a evolução de um comportamento muitas vezes não é conhecida (Nelson *et al.*, 2008).

A diferença entre RE e a aplicação de técnicas de CE em problemas de controle da engenharia é o objetivo da evolução. Sob o ponto de vista da engenharia a função de aptidão é funcional, geralmente para otimizar um número de parâmetros para um problema de controle para um ambiente com propriedades bem definidas. Em RE o objetivo é comportamental ou seja, a evolução de comportamentos de robôs autônomos em ambientes desconhecidos ou parcialmente conhecidos (Elfwing, 2007).

Os princípios para o desenvolvimento de uma função de aptidão para um sistema de controle evolucionário para um robô autônomo tem duas conseqüências (Nolfi e Floreano, 2001): 1) os resultados obtidos com funções de aptidão ligeiramente uniformes são difíceis de se comparar; 2) a escolha de uma função de aptidão apropriada pelo método de tentativa e erro demanda um processo de experimentação, uma abordagem que freqüentemente consome tempo quando o sistema evolucionário está embarcado em robôs reais.

Visando auxiliar o projetista a definir uma boa função de aptidão, Floreano e Urzelai (2000) propuseram uma ferramenta gráfica em três dimensões, que descreve e compara possíveis funções de aptidão para sistemas de controle para robôs autônomos. Uma função de aptidão representa um ponto neste espaço 3D. A Figura 3.3 ilustra esta representação tridimensional.

- **Dimensão Funcional-Comportamental**: uma função de aptidão puramente funcional é baseada na média ponderada entre as medidas de cada função do sistema de controle. Uma função de aptidão comportamental mede somente comportamentos individuais. A posição da função de aptidão ao longo da dimensão funcional-

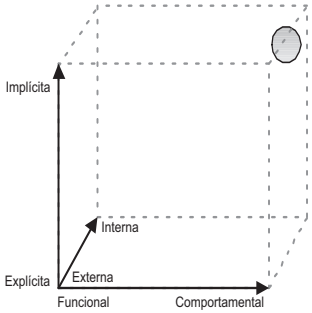


Figura 3.3: Função de aptidão em um espaço tridimensional.

comportamental depende do número de componentes existentes de cada uma e a relação entre eles.

- **Dimensão Explícita-Implícita**: o eixo explícita-implícita refere-se a quantidade de requisitos explicitamente impostos pelo projetista para a seleção dos indivíduos que participarão do processo de reprodução. Estudos em vida artificial adotam a abordagem implícita, pois na vida real não existem requisitos impostos explicitamente para a escolha dos indivíduos mais adaptados.

- **Dimensão Externa-Interna**: essa dimensão indica se a função de aptidão será computada usando informações internas ou externas ao sistema de controle do robô. Usualmente funções de aptidão externas são usadas em ambientes simulados onde todos os aspectos do sistema de controle estão disponíveis para o programador.

A aptidão subjetiva é uma forma de avaliar o desempenho do sistema através de um processo de inspeção visual realizada por um ser humano. Geralmente a aptidão subjetiva está definida no quadrante CEE (Comportamental, Explícita, Externa) (Lund *et al.*, 1998). Neste caso a escolha sobre quais indivíduos serão utilizados para a reprodução é definida pelo operador humano.

A decisão sobre qual dimensão uma função de aptidão deve ser definida depende dos objetivos do sistema. Se o objetivo for a otimização de parâmetros para um problema bem definido, a função de aptidão deve estar no quadrante FEE (Funcional, Explícita, Externa). Entretanto, se o objetivo for o controle de robôs móveis, sem nenhuma intervenção humana, em um ambiente desconhecido e imprevisível, a função de aptidão deve

estar no quadrante CII (Comportamental, Implícita, Interna) (Nolfi e Floreano, 2001).

## 3.3 Abordagens da Robótica Evolucionária

Nesta seção são descritas as duas principais abordagens quanto ao processo de evolução em RE, que são: Evolução em Simulador e Evolução em Robôs Reais (Embarcada). Estas abordagens são conceitualmente aceitas para classificar experimentos realizados na área de Robótica Evolucionária e são descritas em detalhes em: (Mataric e Cliff, 1996), (Pollack *et al.*, 2000) e (Nolfi e Floreano, 2001).

### 3.3.1 Evolução em Simulador

A simulação é uma importante ferramenta em RE pois permite que sistemas sejam evoluídos, geralmente, num tempo menor do que seriam em ambientes reais. Uma das vantagens do processo de evolução simulada é a possibilidade de se fazer estudos preliminares do processo evolutivo, por exemplo, definir uma série de parâmetros iniciais, que mais tarde poderão ser aplicados ao sistema em ambiente real (Nolfi *et al.*, 1994).

Entretanto, em um ambiente simulado é difícil representar virtualmente o ambiente real, onde eventualmente podem acontecer fatos que não foram previstos no simulador. De acordo com Brooks (1992), simular o dinamismo do mundo real, não é uma tarefa trivial, programas que funcionam suficientemente bem em robôs simulados podem não funcionar em robôs reais.

Existem algumas razões para que um sistema de controle desenvolvido em simulador não funcione adequadamente em robôs reais (Miglino *et al.*, 1995):

- simulações numéricas geralmente não consideram aspectos físicos da interação de um robô real com o ambiente, tais como: fricção, inércia, peso, massa etc.
- as leituras dos sensores não devem ser confundidas com a descrição do ambiente, como acontece com alguns modelos de simuladores. Devido ao problema da não linearidade, sensores em robôs reais podem retornar valores incorretos sobre o ambiente, o que resultará em comandos que podem levar os atuadores a tomar ações imprevisíveis. Modelos simulados freqüentemente utilizam uma estrutura matricial para representar o ambiente e com isto, os sensores sempre retornarão informações perfeitas.

- devido às características eletrônicas e mecânicas, ou até mesmo por estarem posicionados de forma diferente no robô, sensores ou atuadores que aparentemente possuem a mesma característica física, podem executar de maneira diferente. Este fato é geralmente ignorado em modelos simulados.

Algumas destas questões podem ser facilmente eliminadas se o simulador for projetado cuidadosamente. Para tentar minimizar as diferenças de leituras entre sensores reais e simulados, Miglino *et al.* (1995) sugere utilizar amostras de leituras realizadas com os sensores reais de diferentes objetos em diferentes ângulos. Essas amostras podem ser utilizadas no simulador para definir os diferentes níveis de ativação dos sensores do robô dependendo de sua localização no ambiente simulado.

Jakobi *et al.* (1995) propôs a adição de ruídos em todos os níveis da simulação objetivando reduzir a diferença de desempenho entre o ambiente simulado e o ambiente real. Ruídos podem ser introduzidos pela simples adição de valores, selecionados aleatoriamente, aos valores computados pelos sensores ou sobre os efeitos (comandos) aos atuadores (Jakobi, 1998a).

Em outro trabalho Jakobi (1998b) propôs uma nova abordagem para o desenvolvimento de simuladores, chamada de simulação mínima, onde os principais elementos são: um conjunto de características mínimas (básicas) e um conjunto de características complementares.

O conjunto de características básicas deve ser cuidadosamente modelado e ser variado de tempos em tempos para permitir que o comportamento esperado tenha um bom desempenho em situações variadas do ambiente. Cada característica do conjunto complementar deve ser adicionada ao simulador de forma aleatória, em diferentes estágios.

Jakobi (1998b) demonstrou, através de alguns experimentos, que a simulação mínima tem um bom desempenho, até mesmo quando o sistema de controle evolucionário é transferido para robôs reais. A dificuldade no uso da simulação mínima é determinar os conjuntos de características básicas e complementares.

### 3.3.2 Evolução em Robôs Reais

A evolução aplicada a robôs reais é uma alternativa às técnicas de evolução em simulador, principalmente se a tarefa que o robô deverá cumprir estiver relacionada a ambientes não estruturados e/ou dinâmicos.

Quando o processo de evolução está implementado em robôs reais o tempo de processamento para se convergir a uma solução aceitável é um fator que deve ser levado em consideração. Alguns experimentos relatam que a convergência para uma solução aceitável pode levar muito tempo (Pollack *et al.*, 2000), (Floreano e Mondada, 1994).

Um outra abordagem, chamada de Evolução Embarcada (EE) ou Imersa (do Inglês *Embodied Evolution*), faz uso de uma população de robôs para evoluir o sistema de controle de cada robô, ao contrário da abordagem clássica, onde todo o processo evolucionário acontece em um único robô (Watson *et al.*, 2002). De acordo com Simões (2000), um sistema evolucionário embarcado é aquele em que o processo evolucionário ocorre sobre uma população de indivíduos (robôs reais), completamente independente de computação externa ou da intervenção do usuário, a fim de avaliar, reproduzir e reposicionar os robôs para novos testes na geração seguinte.

Na EE os novos indivíduos criados pelo processo de reprodução utilizam o "corpo" de outros robôs que fazem parte da mesma população. A reprodução é a troca de material genético que codifica o programa de controle (Pollack *et al.*, 2000). A EE provê uma intersecção entre RE e robótica coletiva (Ficici *et al.*, 1999), (Usui e Arita, 2003).

Uma desvantagem do uso de EE é a possibilidade de um robô não mais se reproduzir devido a problemas que podem acontecer com os outros robôs. Por exemplo, se em um sistema com dois ou mais robôs alguns falharem e restar somente um, esse robô não poderá mais se reproduzir, pois o processo de reprodução em EE é dependente do bom funcionamento de pelos menos dois robôs.

Um outra abordagem utilizada para evoluir controladores para robôs é conhecida como evolução híbrida (EH) (Mataric e Cliff, 1996). A EH utiliza tanto o simulador quanto o robô fisicamente. O objetivo é transferir o resultado da evolução de um sistema de controle, que acontece em ambiente simulado, para um robô real.

Alguns exemplos da abordagem híbrida podem ser encontrados nos trabalhos de Hornby *et al.* (2000) e na seqüência de experimentos realizadas por Nelson, que utilizou AGs para evoluir uma rede neural em simulador para controlar robôs do tipo EvBots (Nelson *et al.*, 2003), (Nelson *et al.*, 2004a) e (Nelson *et al.*, 2004b).

## 3.4 Trabalhos Relacionados

Nesta seção são descritos alguns trabalhos relacionados ao objetivo desta tese, que é a evolução do controlador de um grupo de robôs móveis. Os trabalhos relatados foram separados em dois grupos: Robótica Evolucionária com Algoritmo Genético (Seção 3.4.1) e Robótica Evolucionária com Programação Genética (Seção 3.4.2).

### 3.4.1 Robótica Evolucionária com Algoritmo Genético

Ficici *et al.* (1999), inspirado no trabalho de Mataric e Cliff (1996), propôs uma nova abordagem para evoluir, de maneira embarcada, um conjunto de robôs para solucionar um dado problema. Em seu trabalho Ficici *et al.* (1999) desenvolveu um novo algoritmo chamado de Transferência Genética Probabilística (*Probabilistic Gene Transfer Algorithm (PGTA)*, que é uma variação do algoritmo de Harvey (1996), o microbial GA (ver Capítulo 2, Seção 2.2.6).

No PGTA, a reprodução é um comportamento qualquer, que concorre com os outros comportamentos e pode ser ativado a qualquer momento. Não existe um mecanismo específico de reprodução. Cada robô mantém um nível virtual de energia, que representa seu desempenho ou aptidão. Cada robô, probabilisticamente, difunde mensagens que contém material genético a uma taxa proporcional ao seu nível de energia.

Cada mensagem enviada contém uma versão modificada (mutada) de um gene selecionado aleatoriamente do genótipo do robô. A probabilidade de um outro robô receber a mensagem é inversamente proporcional ao seu nível de energia. Robôs com alto nível de energia aceitam menos mensagens do que os com baixo nível de energia. Ao receber a mensagem o robô substitui o gene correspondente em seu genótipo pelo gene recebido.

Para validar o PGTA, Ficici realizou um experimento utilizando oito robôs do tipo *Cricket* (Resnick *et al.*, 1997), desenvolvido pelo *MIT Media Laboratory*. O ambiente de teste foi um tablado de 130cm x 200cm com um lâmpada posicionada no meio. O objetivo dos robôs era se aproximar da luz, não importando qual a sua posição no tablado.

Sobre a fonte de luz havia um sensor de infra-vermelho que era utilizado pelo robô para saber o quão próximo ele estava da lâmpada. Quando o robô estava muito próximo da lâmpada um comportamento de reposicionamento era ativado, este comportamento fazia com que o robô se reposicionasse no ambiente numa posição aleatória para re-iniciar

o processo de localização da luz.

O nível de energia de cada robô era controlado da seguinte forma: se o robô se aproximasse da fonte de luz, o nível de energia era incrementado até atingir um valor máximo. Para cada mensagem enviada contendo um gene, o nível de energia era decrementado até atingir um valor mínimo.

A arquitetura de controle era formada por uma rede neural do tipo direta, totalmente conectada, com dois neurônios de saída, um para cada motor, um neurônio de entrada, com um codificação binária, que indicava qual dos dois sensores estavam recebendo mais intensidade de luz e um neurônio de linearização (bias). Cada peso sináptico tinha um valor inteiro entre a faixa de valores (-8, 7). Os valores dos neurônios de saída eram calculados através do somatório dos pesos dos neurônios de entrada. A Figura 3.4 ilustra a arquitetura neural utilizada por Ficici.

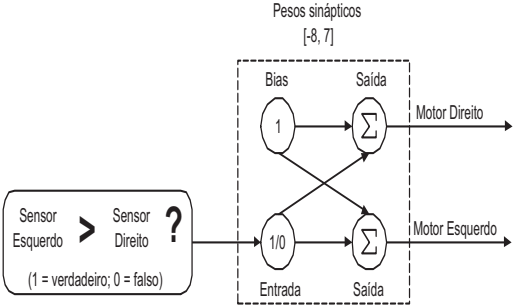


Figura 3.4: Arquitetura de controle utilizada nos experimentos de Ficici *et al.* (1999).

Na arquitetura de controle de Ficici, o PGTA era responsável por ajustar os pesos dos neurônios, para tanto, o PGTA utilizava os dados lidos dos sensores de infra-vermelho.

Com seu trabalho Ficici provou que abordagens descentralizadas e assíncronas, para evoluir uma população de robôs reais, podem gerar programas de controle de alta qualidade.

Nehmzow (2002) realizou alguns experimentos com robôs móveis usando o algoritmo PEGA (*Physically Embedded Genetic Algorithm*) para ativar comportamentos individuais embarcados nos robôs. Os experimentos foram realizados com dois robôs móveis equipados com sensores de infra-vermelho, sensores de toque, sonar e sensores de luz. A comunicação entre os robôs era feita através dos sensores de infra-vermelho.

Cada robô era programado para executar um determinado comportamento por um período de tempo pré-estabelecido. Após esse tempo expirar, um comportamento de busca e localização era ativado, esse comportamento fazia com que os robôs procurassem um ao outro. Quando isso acontecia, eles se posicionavam de maneira a facilitar a troca de material genético através dos sensores de infra-vermelho.

Cada robô era controlado por um vetor de bits que codificava a política de ativação para um determinado comportamento conforme as leituras dos sensores. O vetor de bits era modificado por um AG. Cada robô continha uma população de dois indivíduos (dois vetores de bits para cada comportamento) utilizados pelo processo evolucionário controlado pelo AG.

O cruzamento acontecia quando dois robôs trocavam material genético (vetores de bits) entre si. Quando um robô recebia o vetor de bits de outro robô, era avaliado se o valor da aptidão do emissor era maior que o valor da aptidão do indivíduo local (receptor), se fosse, a metade do vetor de bits local era substituída pela metade do vetor de bits recebido. Caso contrário, essa substituição não era realizada.

Foram realizados dois conjuntos de experimentos. No primeiro conjunto, cada robô deveria realizar uma única tarefa (competência). As tarefas do primeiro conjunto eram: detecção e desvio de obstáculos, seguir a luz e seguir o líder. No segundo conjunto de experimentos, as tarefas eram mais complexas, pois agrupavam tarefas do primeiro conjunto. As tarefas do segundo conjunto eram: seguir a luz e detectar e desviar de obstáculos; e, seguir a luz, detectar e desviar de obstáculos e seguir o líder.

Os resultados obtidos com os experimentos demonstraram que a população foi capaz de convergir uma determinada competencia (tarefa) em pouco menos de 30 minutos.

Simões (1999) realizou experimentos para evoluir o sistema de controle de um grupo de 5 (cinco) robôs. Cada robô tinha a seguinte configuração: microcontrolador Motorola modelo 68HC11 de 2MHz, 128Kb de memória RAM, bateria com autonomia de 4 (quatro) horas, dois motores com controle diferencial, 8 (oito) sensores de colisão e 8 (oito) sensores de infravermelho. A troca de mensagens entre os robôs era feita por um rádio AM de 418MHz com taxa de transmissão de 1.2Kbps.

O controle de cada robô foi implementado com uma rede neural do tipo RAM com a seguinte configuração: 64 x 4 neurônios de entrada conectados ao módulo de controle dos 8 (oito) sensores. As leituras de cada sensor eram o convertidas em um sinal de 2

bits. A saída da rede consiste de 8 (oito) comandos para cada motor, que são: S (*stop*), FS (*front slow*), FM (*front medium*), FF (*front fast*), TRS (*turn right sharp*), TRL (*turn right long*), TLS (*turn left sharp*) e TLL (*turn left long*).

Foi utilizado AG para determinar o comando de saída para os motores, bem como para determinar a configuração sensorial dos robôs. Em cada robô dois genes determinam a presença ou a ausência de um determinado sensor. Essa configuração não somente faz com que o sistema de controle se adapte mas também permite alterar a morfologia de cada robô.

Cada ciclo do processo evolucionário é dividido em duas partes: a primeira trata da tarefa a ser executada por cada robô, e é chamada de sessão de trabalho; a segunda trata do processo de cruzamento, onde cada robô deve procurar um parceiro para cruzar, baseado no valor da aptidão, chamada de sessão de cruzamento.

Em cada sessão de cruzamento o melhor robô, como o maior valor de aptidão, envie o seu cromossomo para todos os outros robôs da população. Os robôs que receberam o cromossomo iniciam o processo de cruzamento, ou seja, substituirão parte do seu cromossomo por uma parte do cromossomo do melhor robô. Após o cruzamento o operador de mutação é aplicado ao novo cromossomo. Após esse processo os robôs reiniciam a sessão de trabalho dando início a uma nova geração.

Para avaliar o sistema de controle foi realizado um experimento com o grupo de 5 (cinco) robôs em um ambiente de 4m x 4m contendo paredes e obstáculos de tamanhos variados. O objetivo de cada robô era de navegar por este ambiente evitando colisões. O experimento foi realizado em 4 (quatro) períodos de duas horas, no final do terceiro período o ambiente foi modificado para tornar-se mais complexo.

A função de aptidão foi definida como uma forma de punição e recompensa onde, para cada colisão, cada robô perde 8 (oito) pontos em seu valor de aptidão, caso contrário, a cada segundo, ganha um ponto. Durante o processo evolucionário três distintas espécies surgiram: na primeira, um grupo de robôs aprendeu a usar o sensor frontal para desviar de obstáculos; na segunda, um grupo de robôs aprendeu a usar dois sensores, frontal e um lateral para desviar de obstáculos; e finalmente no terceiro tipo de espécie um grupo de robôs aprendeu a usar três sensores, o frontal e dois laterais. Conforme o sistema foi evoluindo a terceira espécie foi se tornando dominante, até a completa extinção das espécies um e dois.

Em outro trabalho Simões e Dimond (2001) relatam um experimento realizado com seis robôs, entretanto não foi utilizado uma rede neural no sistema de controle. Nesta caso, o controle foi considerado não estruturado, ou seja, o controle é representado como uma espécie de caixa preta na qual pode representar na saída qualquer função binária lógica, conforme os valores de entrada lidos dos oito sensores.

O mesmo experimento de navegação livre de colisões foi realizado por Simões (2002) usando algoritmo genético para configurar o sistema de controle, que utiliza uma rede neural do tipo RAM, bem como a morfologia de cada robô. O experimento foi realizado com seis robôs tanto em simulador quanto com robôs reais. Diferentes taxas de mutação foram testadas com o objetivo de avaliar a diversidade da população. O experimento mostrou que em 200 gerações foi possível evoluir um sistema de controle para a tarefa de navegação livre de colisões.

### 3.4.2 Robótica Evolucionária com Programação Genética

Em sua dissertação de mestrado, Kofod-Petersen (2002), utilizou PG para evoluir um conjunto de comportamentos totalmente embarcado em um robô real. O robô utilizado foi um Khepera (Mondada *et al.*, 1993) equipado com oito sensores de proximidade, distribuídos ao redor de sua estrutura, e dois motores. O robô enviava informações estatísticas para uma estação de trabalho através de uma conexão serial.

O principal objetivo do trabalho foi demonstrar a possibilidade de utilizar PG para controlar um robô móvel de maneira totalmente autônoma. O trabalho foi baseado nos experimentos realizados por Nordin (Nordin e Banzhaf, 1997a), (Nordin e Banzhaf, 1997b), que utilizou PG para evoluir um sistema de controle para resolver alguns problemas em robótica móvel, tais como, desviar de obstáculos e seguir paredes.

Ao contrário de Nordin, que desenvolveu o sistema de controle em Assembly (Nordin e Banzhaf, 1995), Petersen utilizou a linguagem de programação C. Os indivíduos da população eram representados de maneira linear, consistindo de duas partes: 1) informações sobre o indivíduo como tamanho, aptidão, identificação etc; e 2) um programa em C. A Figura 3.5 ilustra um exemplo de representação linear dos indivíduos.

As configurações utilizadas para cada ciclo de execução foram: seleção por torneio, com dois indivíduos por torneio; preservação dos pais após cruzamento (*steady-state gp*); mutação alterando somente um terminal ou uma função; e, população de no máximo 50

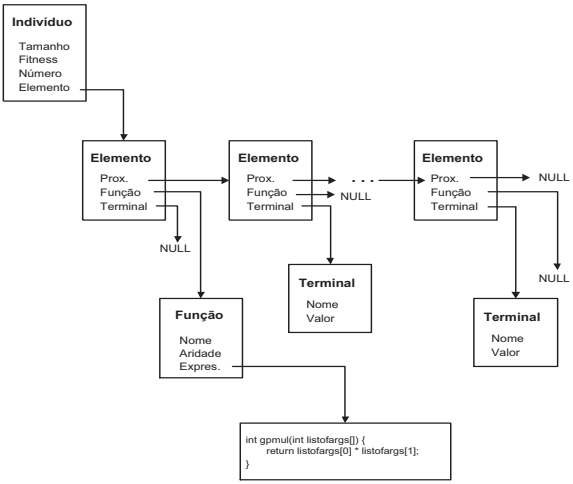


Figura 3.5: Representação linear dos indivíduos da população.

(cinquenta) indivíduos.

A função de aptidão utilizada nos experimentos de Petersen é representada na Equação 3.1.

$$f = \alpha \times ((m_1 + m_2) - |m_1 - m_2| - \beta \times \sum_{0}^{7} s_i) \tag{3.1}$$

onde $\alpha$ e $\beta$ são constantes que determinam os valores para os dois motores e $s_i$ os sensores.

Petersen não obteve discrepâncias entre os seus resultados e os resultados de Nordin, apenas uma pequena variação de desempenho com relação ao tempo de execução. Isso deu-se devido a escolha da linguagem de programação C ao invés do uso de Assembly.

Em (Sim *et al.*, 2002) foi utilizado programação genética num hardware FPGA (*Field Programmable Gate Array*) acoplado a robôs do tipo Khepera para resolver o problema de empurrar a caixa. O problema de empurrar a caixa consiste em um ambiente composto por alguns obstáculos e uma caixa. O objetivo dos robôs é, através de um processo de cooperação, deslocar (empurrar) a caixa da posição de origem para uma outra posição no ambiente.

Para a realização do experimento, Sim utilizou dois robôs do tipo Khepera equipados com um hardware FPGA da XILINX modelo XC6216. O conjunto de terminais representava os movimentos possíveis do robô e o conjunto de funções representava as

interpretações das leituras dos sensores.

Os programas, candidatos à solução do problema, foram representados em formato de árvores. Para evitar o crescimento exagerado dessas árvores, que poderiam exceder as capacidades do hardware, Sim criou uma maneira de dividir a árvore em sub-árvores para poder ser alocada em porções individuais no hardware. O processo de divisão e execução da árvore foi chamada de troca de contexto.

Na troca de contexto cada árvore é dividida em partes menores, sub-árvores, que são alocadas a uma porção de hardware. Após o processo de divisão, cada sub-árvore, chamada de sub-processo, é selecionada, conforme a ordem de execução, e convertida em uma forma de representação de hardware. Após a execução individual de cada sub-processo, os resultados parciais foram combinados para produzir o resultado final.

Em cinquenta gerações foi possível encontrar uma solução satisfatória para o problema. O mesmo experimento foi realizado por Lee *et al.* (1997), entretanto o ambiente utilizado não continha obstáculos e o sistema de controle não executava diretamente sobre um hardware FPGA.

Em (Barlow e Oh, 2006) e (Barlow e Oh, 2008) é relatado um experimento usando programação genética para evoluir o sistema de controle de um veículo aéreo não tripulado. O experimento foi realizado em um ambiente simulado, um quadrado de 100 milhas náuticas em cada lado.

A cada execução o veículo era posicionado aleatoriamente no ambiente. A altitude e velocidade eram parâmetros definidos com valores constantes na simulação, no caso a velocidade foi definida em 80 nós.

O objetivo da simulação era encontrar um conjunto de 10 (controladores) que pudessem ser comparados com outras técnicas para avaliar qual o melhor controlador a ser utilizado em veículos aéreos não tripulados.

Foram gerados 500 indivíduos por geração em 50 (cinqüenta) execuções da simulação. Os dados foram processados em um *cluster* Beowulf contendo 94 (noventa e quatro) computadores. Foram definidas 4 (quatro) funções de aptidão para as seguintes tarefas: voar em direção ao radar, voar ao redor do radar, vôo eficiente e vôo estável.

Foram selecionados os 10 (dez) melhores controladores após o processo de evolução. Esses controladores foram comparados com um controlador PD (Proporcional-Derivativo) e com um controlador manual desenvolvido especificamente para o problema em questão.

Os 12 (doze) controladores foram testados em diferentes situações, inclusive com variação da velocidade do vento. O melhor controlador gerado, após o processo evolucionário, foi o melhor em todos os testes realizados, mesmo quando perturbações eram inseridas no sistema. Neste caso, tanto o controlador desenvolvido manualmente como o controlador PD falharam.

Em (Francisco e Jorge dos Reis, 2008) foi realizado um experimento utilizando programação genética no problema da presa e predador. Este problema se caracteriza pela existência de duas espécies de indivíduos: a presa que normalmente é mais lenta que o predador precisa constantemente fugir dos ataques do predador; e o predador que tem por objetivo primário capturar a presa.

O problema da presa e predador possui uma característica interessante, pois permite que dois comportamentos distintos, e ao mesmo tempo dependentes um do outro, sejam evoluídos concorrentemente, isto é são co-evoluídos. Nesse caso, conforme se dá a adaptação da presa também acontece a adaptação do predador e vice-versa.

Para o experimento os autores desenvolveram um simulador espacial onde existem diferentes naves com as seguintes características: com controle de velocidade variável; podem se deslocar em quatro direções, para frente, para trás, para a direita e para a esquerda; o predator possui um mecanismo de tiro que é utilizado para capturar as presas.

O objetivo do predator era atirar e destruir o maior número de presas possíveis. Conseqüentemente as presas necessitavam se deslocar no espaço visando ficar fora de alcance dos predadores. Os seguintes parâmetros foram utilizados no experimento: 200 predadores, 50 presas, o ambiente era definido por uma matriz de 5000 por 5000 pixels.

Para avaliar os resultados obtidos com a programação genética os autores desenvolveram controladores manuais para a presa e para o predador. Foram feitos testes com a presa evoluída e o predator com o controlador manual e também com a presa com controlador manual e o predator com o controlador evoluído. Em ambos os casos, os controladores que foram evoluídos se mostraram mais eficientes do que aqueles desenvolvidos manualmente.

## 3.5 Resumo do Capítulo

A Robótica Evolucionária é uma importante área de estudos no campo da robótica, principalmente da robótica móvel. Na RE é possível, com a utilização de técnicas de computação evolucionária, sintetizar automaticamente sistemas de controle para robôs com o propósito de treiná-los para desenvolver tarefas específicas. A RE pode ser classificada em duas principais abordagens: RE Simulada e RE com Robôs Reais.

Na abordagem simulada todo o processo evolucionário acontece em um simulador. As vantagens em se utilizar um simulador são a velocidade, o baixo custo e a possibilidade de se fazer estudos preliminares do processo evolucionário.

Na RE com robôs reais o processo evolucionário acontece embarcado nos robôs. Também é possível evoluir um sistema de controle entre vários robôs, nesse caso o processo evolucionário ocorre sobre uma população de indivíduos (robôs reais), completamente independente de computação externa ou da intervenção do usuário.

Uma terceira abordagem, que faz uso das duas anteriores, é a evolução híbrida, que utiliza tanto o simulador quanto o robô real. Na RE híbrida o processo evolucionário pode acontecer totalmente em um simulador e o resultado ser transferido para o robô real a fim de ser avaliado.

No capítulo foram apresentados alguns estudos de caso, tanto com o uso de Algoritmos Genéticos como com o uso de Programação Genética. Em particular, em Evolução Embarcada (EE) foram descritos alguns experimentos com AG. Em Programação Genética (PG), os experimentos descritos consideravam que cada robô possuia internamente uma população de programas locais e que cada robô, no caso dos experimentos com mais de um robô, não tinha conhecimento da existência dos outros robôs.

# Capítulo 4

# Extensão da Programação Genética Distribuída

Este capítulo descreve o algoritmo da xPGD, uma extensão do algoritmo da Programação Genética Distribuída, capaz de suportar a evolução do sistema de controle de uma população de robôs móveis; e o SEGS (Sistema de Execução, Gerenciamento e Supervisão), o sistema responsável pela execução, gerenciamento e supervisão da xPGD. Também são descritas as principais características e a metodologia utilizada para o desenvolvimento da xPGD e do SEGS, respectivamente. No final do capítulo apresenta-se um pequeno resumo sobre o conteúdo do mesmo.

## 4.1 Descrição do Algoritmo da xPGD

A xPGD é uma extensão do algoritmo da Programação Genética Distribuída (ver Capítulo 2; Seção 2.3.8). A xPGD é baseada no Microbial GA (ver Capítulo 2; Seção 2.2.6), uma variação do AG, seu funcionamento é semelhante à recombinação (infecção) genética que acontece nas bactérias onde segmentos do DNA (*Deoxyribonucleic Acid - Ácido Desoxirribonucleico*) são transferidos entre dois membros da população.

Na xPGD são considerados dois conjuntos de populações. O primeiro, chamado de conjunto local ou $P_{local_{R_i}}$, refere-se à população local de cada robô $R_i$, isto é, o conjunto de indivíduos ou soluções candidatas que estão embarcadas no robô. Cada $x \in P_{local_{R_i}}$ representa uma solução candidata a um problema. O segundo conjunto, chamado de conjunto total ou $P_{total}$, é formado pela união de todas as populações locais de cada robô,

isto é, $P_{total} = P_{local_{R_1}} \cup P_{local_{R_2}} \cup P_{local_{R_3}} \cup \ldots, P_{local_{R_n}}$. O processo evolucionário ocorre sempre considerando a população total, ou seja, partes (sub-árvores) de um indivíduo local de um determinado robô podem ser consideradas no processo evolucionário da população local de outro robô.

Ao contrário de outras abordagens em Evolução Embarcada (EE), tais como as descritas em Simões e Dimond (2001) e Watson *et al.* (2002), na xPGD o processo evolucionário é assíncrono, isto é, não é necessário que dois robôs se sincronizem para se reproduzirem. Esta característica é possível porque a xPGD utiliza uma população de indivíduos (programas) em cada robô que faz parte de uma população de robôs. Isso evita que cada robô, ao final de cada ciclo de execução (geração), procure um parceiro para cruzar, como acontece nas soluções síncronas.

Na xPGD partes de um indivíduo mais adaptado são enviados para todos os outros robôs. A idéia é que cada robô possa infectar os outros robôs da população com algumas partes do melhor programa gerado a partir de sua população de programas local.

A seqüência de passos do algoritmo da xPGD é a seguinte:

1. Criar aleatoriamente uma população de programas;
2. Executar iterativamente os seguintes passos até que algum critério de parada seja satisfeito:
   (a) Avaliar cada programa da população através de uma função de avaliação, que expressa a sua aptidão;
   (b) Receber uma mensagem $M$ [1] de um indivíduo remoto[2] enviada por outro robô;
   (c) Selecionar os $t$ melhores indivíduos da população local usando o método de seleção por torneio [3];
   (d) Selecionar aleatoriamente uma parte do melhor indivíduo local (mais adaptado) e enviar uma mensagem $M$, contendo a parte selecionada mais o valor da aptidão, em *broadcast* (difusão) para os outros robôs;
   (e) Comparar se a aptidão do pior indivíduo selecionado localmente é menor que a aptidão do indivíduo remoto. Se verdade, execute o operador de mutação

[1] Cada mensagem recebida contém o valor da aptidão e uma parte, selecionada aleatoriamente, da árvore do indivíduo remoto.
[2] Um indivíduo remoto é um programa que faz parte da população local de outro robô.
[3] O tamanho do torneio é um parâmetro definido antes da execução do algoritmo.

remota substituindo uma parte, selecionada aleatoriamente, do indivíduo local pela parte recebida do indivíduo remoto;

(f) Executar os operadores de cruzamento e mutação;

3. Retornar com o melhor programa encontrado.

No passo (d) do algoritmo da xPGD, selecionar uma parte do melhor indivíduo local, refere-se a selecionar aleatoriamente uma função ou terminal que faz parte da estrutura do programa (indivíduo). Caso a parte selecionada seja uma função, então toda a estrutura dependente desta função, de acordo com o valor de sua aridade, deve ser também enviada na mensagem.

As partes recebidas remotamente são adicionados a estrutura do pior indivíduo, dos $t$ melhores selecionados, obedecendo os critérios da Equação 4.1:

$$P_{local}(I) = \left\{ \begin{array}{ll} OMR(I) & \text{if } Fitness(I_{remoto}) > Fitness(I_{local}) \\ I & \text{if } Fitness(I_{remoto}) \leq Fitness(I_{local}) \end{array} \right\} \quad (4.1)$$

onde, $P_{local}(I)$ representa o indivíduo $I$ selecionado da população local ($P_{local}$). O operador de mutação remota ($OMR$) é executado sobre o indivíduo $I$ se o valor da aptidão do indivíduo remoto ($Fitness(I_{remoto})$) for maior que o valor do aptidão de $I$ ($Fitness(I_{local})$). Caso contrário, o indivíduo local $I$ não sofre a mutação e permanece com sua estrutura inalterada.

O método de seleção empregado na xPGD é a seleção por torneio com a manutenção dos pais após o cruzamento. Esta é uma técnica elitista conhecida na bibliografia da área como *steady-state genetic programming* (Watson e Parmee, 1997).

A Figura 4.1 ilustra o diagrama de blocos do xPGD. O diagrama é praticamente o mesmo do algoritmo tradicional da PG (ver Capítulo 2; Seção 2.3) com excessão dos retângulos de cor cinza que representam as modificações necessárias para permitir que o processo evolucionário também ocorra de forma distribuída.

As mensagens trocadas entre os robôs devem conter a aptidão e uma parte da estrutura que representa o indivíduo da população local. Para isso, todas as funções e terminais recebem uma identificação numérica única.

Diferentemente de outras abordagens em EE, a xPGD garante a continuidade do processo evolucionário do sistema de controle, mesmo quando houver algum problema

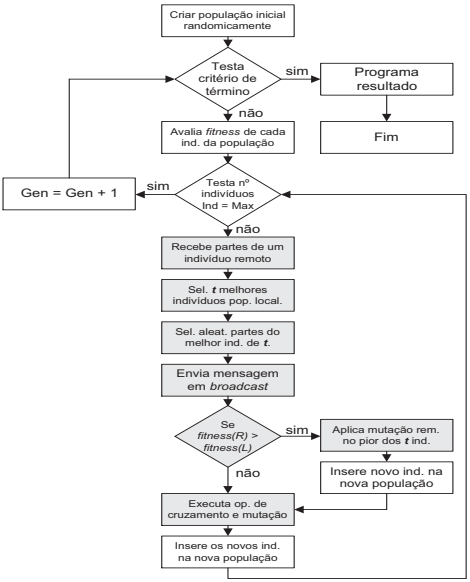


Figura 4.1: Diagrama de blocos da xPGD.

com os outros robôs que fazem parte da população de robôs. Isto é possível porque cada robô possui uma população de programas local, o que garante a continuidade do processo evolucionário.

Para uma melhor compreensão do processo de funcionamento da xPGD, na Seção 4.1.1 é descrito um exemplo em que um conjunto de robôs deve explorar um ambiente qualquer a procura de comida.

### 4.1.1 Exemplo de Funcionamento da xPGD

Na tarefa descrita neste exemplo um conjunto de robôs deve navegar por um ambiente a procura de comida. Esse problema é conhecido como forageamento. A Tabela 4.1 lista o conjunto de funções e terminais usados nesse exemplo.

Na Tabela 4.1 a coluna *Id.* representa a identificação de cada função e cada terminal. Para facilitar o entendimento e melhorar a apresentação do problema, todas as funções foram identificadas com um número inteiro par e todos os terminais com um número inteiro ímpar.

A Figura 4.2 ilustra o mecanismo de mutação remota da xPGD. Nela são representados

Tabela 4.1: Conjunto de funções e terminais para o problema do forageamento.

| **Funções** | | | |
|---|---|---|---|
| **Nome** | **Aridade** | **Id.** | **Definição** |
| ComidaFrente | 2 | 2 | Se encontrou comida, executa o ramo da esquerda; senão, executa o ramo da direita. |
| Prog2 | 2 | 4 | Executa dois ramos da árvore |
| Prog3 | 3 | 6 | Executa três ramos da árvore |
| **Terminais** | | | |
| VirarDireita | 0 | 1 | Faz o robô virar a direita (15 graus). |
| VirarEsquerda | 0 | 3 | Faz o robô virar a esquerda (15 graus). |
| SeguirEmFrente | 0 | 5 | Faz o robô seguir em frente (300ms). |
| Retornar | 0 | 7 | Faz o robô retornar, dar a ré (300ms). |

dois robôs e um indivíduo da população local de cada um.

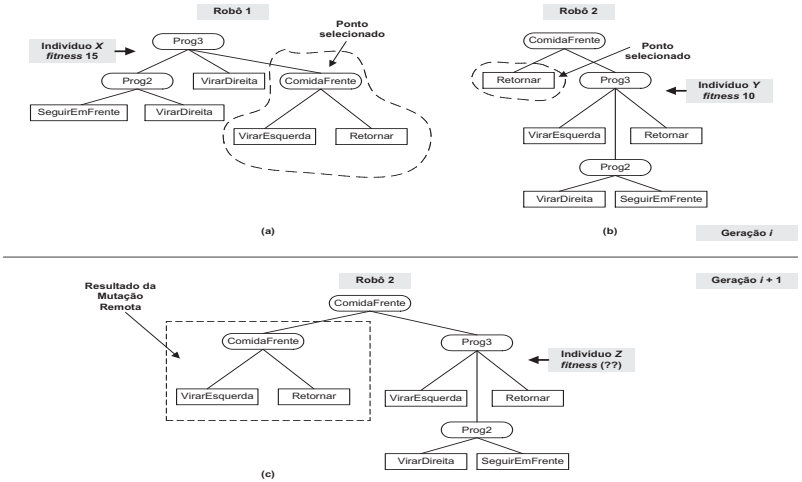


Figura 4.2: Exemplo de funcionamento da operação de mutação remota da xPGD.

Na Figura 4.2 (a), o robô 1, seleciona, aleatoriamente, um ponto da estrutura do seu melhor indivíduo local (*indivíduo X*). O ponto selecionado é a função *ComidaFrente* que de acordo com a Tabela 4.1, possui aridade 2. Nesse caso, todos os itens (funções e/ou terminais) ligados a função *ComidaFrente* devem ser enviados para o robô 2.

O robô 1 cria uma mensagem formada pelos seguintes elementos: $\mathbf{M = \{15,2,3,7\}}$. Isto é, o valor da aptidão do indivíduo $X$ (15), a função *ComidaFrente* e os terminais

*VirarEsquerda* e *Retornar*. Essa mensagem é então enviada para o robô 2.

Ao receber a mensagem, o robô 2, Figura 4.2 (b), irá comparar o valor da aptidão da mensagem com o valor da aptidão do indivíduo local ($Y$). Se a condição imposta pela Equação 4.1 for satisfeita, isto é, se o valor da aptidão da mensagem, ou seja, do indivíduo remoto, for maior que o valor da aptidão do indivíduo local, a operação de mutação remota é executada. Nesse caso, um ponto da estrutura do indivíduo local é selecionado aleatoriamente e então substituído pelas partes recebidas do indivíduo remoto. O resultado da mutação remota gera um novo indivíduo que será testado na próxima geração ($i + 1$).

É importante ressaltar que para um correto funcionamento da xPGD todos os robôs devem conter os mesmos conjuntos de funções e terminais, ou pelo menos, devem utilizar as mesmas funções e terminais para um problema em particular.

Uma versão do problema do forageamento foi testada com o algoritmo da xPGD em ambiente simulado e seus resultados estão descritos na Seção 5.1.2 do Capítulo 5.

Para a execução da xPGD, em cada robô de um Sistema com Múltiplos Robos (SMR), é necessário a existência de um sistema que dê suporte a execução e ao gerenciamento de todo o processo evolucionário, que ocorre de forma embarcada. O sistema desenvolvido com esse propósito chama-se SEGS (Sistema de Execução, Gerenciamento e Supervisão) e é descrito na próxima seção (4.2).

## 4.2 Sistema de Execução, Gerenciamento e Supervisão - SEGS

O SEGS é responsável pela execução, gerência e supervisão de todo o processo evolucionário em cada robô de um SMR (Perez *et al.*, 2007a). A Figura 4.3 ilustra, genericamente, o funcionamento do SEGS.

O SEGS é executado em cada robô que faz parte de um SMR, onde existe uma população local [4] que interage com a população local dos outros robôs. A cada geração, partes do melhor indivíduo de cada robô são enviadas (difusão) para todos os outros robôs que fazem parte do SMR.

[4] A população local é um conjunto de programas candidatos a resolverem um problema. Esses programas estão embarcados no robô. Um problema é uma tarefa qualquer que o robô deve executar. Por exemplo, navegação livre de colisões.

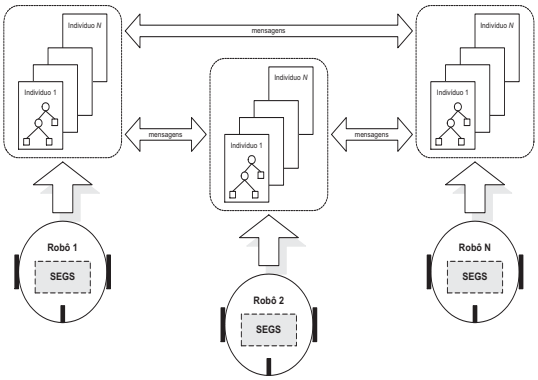


Figura 4.3: Representação esquemática do SEGS.

O SEGS é estruturado em módulos que são ilustrados na Figura 4.4. Cada módulo é responsável por uma etapa do processo evolucionário.

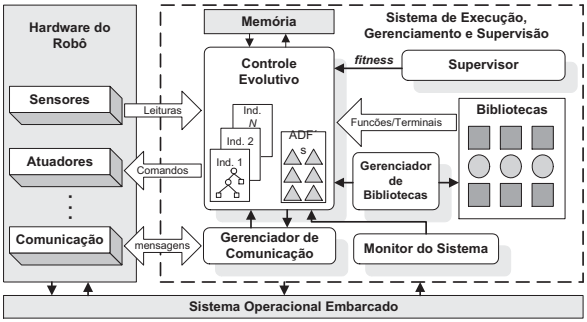


Figura 4.4: Estrutura modular do SEGS.

Abaixo segue a descrição completa do objetivo e o funcionamento de cada módulo que compõe o SEGS, bem como a interligação entre eles (quando houver) (Perez *et al.*, 2007b):

- **Controle Evolucionário (CTE)**: é o componente principal do SEGS, é nele que está implementado a xPGD. O CTE é responsável por criar, aleatoriamente, a população local de indivíduos através das bibliotecas de funções e terminais. A cada nova geração os indivíduos gerados são testados, isto é, executados pelo CTE. As FADs (Funções Automaticamente Definidas ou ADFs (*Automatically Defined Functions*)

do Inglês) também são gerenciadas por esse componente. O CTE está ligado ao Gerenciador de Comunicação, ao Gerenciador de Bibliotecas e à Memória.

- **Memória**: o objetivo da memória é armazenar a representação dos indivíduos mais aptos em cada geração. Por exemplo, considerando o exemplo descrito na Seção 4.1.1, o melhor indivíduo do Robô 1, de acordo com a coluna *Id.* da Tabela 4.1, é representado da seguinte forma: **{6, 4, 5, 1, 1, 2, 3, 7}**, que refere-se a seguinte estrutura em termos de funções e terminais: (*Prog3, Prog2, SeguirEmFrente, VirarDireita, VirarDireita, ComidaFrente, VirarEsquerda, Retornar*). A representação **{6, 4, 5, 1, 1, 2, 3, 7}** é armazenada em memória para poder ser utilizada num processo de recuperação em caso de falhas do CTE ou até mesmo para otimizar tarefas que envolvem mais de uma competência. Nesse caso, a memória funciona como uma espécie de "fotografia" do sistema, podendo ser utilizada para acelerar o processo de aprendizagem através de experiências realizadas no passado. Por exemplo, se a tarefa que o robô deverá cumprir for empurrar uma caixa em um ambiente com diversos obstáculos, primeiro ele aprenderá a navegar por esse ambiente desviando de obstáculos, depois ele aprenderá a encontrar a caixa e empurrá-la para uma determinada posição. Neste caso, a segunda tarefa, empurrar a caixa, se dará a partir do programa que foi gerado na primeira, desviar de obstáculos.

- **Gerenciador de Comunicação (GC)**: é o componente responsável pelo envio e o recebimento das mensagens que são trocadas entre os robôs. No processo de envio de mensagens, o CTE repassa para o GC a seqüência de funções e terminais e o valor da aptidão que serão enviados para os outros robôs. O GC cria uma mensagem contendo essas informações e a envia para os outros robôs. Na recepção, o GC recebe as mensagens enviadas pelos outros robôs e as repassa para o CTE.

- **Gerenciador de Bibliotecas (GB)**: esse componente gerencia os conjuntos de terminais e funções de cada robô. Nesse componente todas as funções e terminais são identificadas por um identificador numérico único para facilitar o seu envio para outros robôs pelo GC. Ter as bibliotecas separadas do sistema evolucionário representa uma vantagem adicional, pois é possível, a qualquer momento adicionar ou remover funções ou terminais sem a necessidade de redefinições no sistema de controle. Por exemplo, caso o robô receba um conjunto de sensores novos com novas

funcionalidades, basta adicionar essas informações ao conjunto de terminais nas bibliotecas do SEGS. Isso faz com que os indivíduos gerados pelo CTE se adaptem às mudanças de características no hardware do robô (morfologia).

- **Supervisor**: é responsável por avaliar e atribuir um valor de aptidão para cada indivíduo, isso é feito através de um método de punição e recompensa. Para um correto funcionamento desse método, para cada tarefa a ser realizada pelo robô, deve ser definido como e quando acontece a recompensa e a punição. Por exemplo, se a tarefa é a navegação livre de colisões, a recompensa e a punição podem ser definidas de acordo com a quantidade de colisões do robô. A cada choque com um obstáculo, o valor da aptidão é decrementado (punição), caso contrário, o valor vai sendo incrementado de tempos em tempos (recompensa).

- **Monitor**: é o componente responsável por avaliar a execução do sistema. Caso o sistema fique inativo, isto é, o robô fique parado por um longo período de tempo, o monitor ativa um processo de re-inicialização do CTE. Quando acontece a reinicialização do CTE, a imagem do melhor indivíduo, que está armazenada na memória, é recuperada e o processo evolucionário inicia-se a partir deste indivíduo, isto é, a população local é completada a partir deste indivíduo ao invés de começar de uma população aleatória como acontece no algoritmo tradicional da PG. Essa operação é realizada com o auxílio da função *Headless Chicken Crossover* (HCC) (Poli e McPhee, 2001). A HCC é uma operação de cruzamento onde somente um dos pais é escolhido da população, o outro é criado de forma aleatória toda a vez que a função é executada. A função HCC foi escolhida porque na memória existirá somente um programa armazenado, logo este será o parâmetro para que ela possa recompor a população local do robô.

O SEGS pode ser implementado diretamente sobre o hardware do robô ou como uma tarefa executando sobre um sistema operacional embarcado. A próxima seção (4.3) descreve em detalhes o processo de implementação do SEGS e da xPGD.

## 4.3 Implementação da xPGD e do SEGS

O SEGS foi implementado para a plataforma EyeBot (Bräunl, 2006) em linguagem de programação C. A plataforma EyeBot é composta pelos seguintes itens (Bräunl, 2003):

- Robôs EyeBot: compreende uma família de robôs móveis baseado no controlador EyeCon (*Eyebot Controller*) (Bräunl, 1999). Os robôs usados nos experimentos desta tese são os *SoccerBot*, robôs desenvolvidos para serem utilizados em competições de futebol de robôs na categoria F180 da *Robocup*. Cada *SoccerBot* é composto por três sensores de infra-vermelho, dois motores e rodas, um câmera colorida de 24 bits por *pixel* com resolução de 80x60 *pixel* e um servo que possibilita que a câmera faça movimentos de *pan* (movimentos para a direita e esquerda), um mecanismo frontal de chute e um rádio para comunicação bidirecional, além do controlador EyeCon que possui 1MB de memória RAM, 512KB de memória Flash ROM e um microcontrolador motorola 68332 de 32 bits e 25MHz.

- Sistema operacional embarcado RoBIOS: O *Robot Basic Input Output System* (RoBIOS, 2006) é um sistema operacional multitarefa responsável pela gerência de todos os dispositivos acoplados aos robôs do tipo EyeBot, bem como as aplicações dos usuários. O RoBIOS provê suporte a programação concorrente através de threads e utiliza semáforo como mecanismo de sincronização de tarefas.

- Simulador EyeBot: permite simular todos os robôs da família EyeBot (Koestler e Bräunl, 2004) com as mesmas características dos robôs reais. É possível simular no mesmo ambiente um grupo com vários robôs. O simulador EyeBot utiliza a mesma biblioteca do RoBIOS o que facilita a migração do ambiente simulado para os robôs reais, necessitando apenas re-compilar o programa.

A xPGD foi implementada como uma biblioteca estática baseada no conceito de Programação Genética Linear (ver Capítulo 2; Seção 2.3.7). A estrutura utilizada para a representação dos indivíduos (programas) é um vetor de tamanho $N$, sendo $N$ um parâmetro definido antes da execução do sistema.

O parâmetro $N$ é utilizado pela função de criação da população inicial e pela função de mutação. Ambas as funções utilizam o algoritmo *Grow* (ver Capítulo 2, Seção 2.3.2),

que cria programas de tamanho variável respeitando um tamanho máximo, nesse caso, o parâmetro $N$.

O operador de mutação remota (OMR) também utiliza o parâmetro $N$ para controlar o tamanho dos programas. Considerando que a condição imposta pela Equação 4.1 foi satisfeita, as seguintes regras devem ser consideradas para a execução do OMR:

1. Todo programa gerado pela xPGD deve possuir um tamanho ($P_{size}$) máximo igual a $N$ e tamanho mínimo igual a $f_{min_{arity}} + 1$, ou seja, formado por apenas uma função de menor aridade, do conjunto de funções disponíveis, mais ela própria.

2. Toda mensagem transmitida e/ou recebida por um robô deve ter tamanho máximo ($M_{size}$) igual a $N - 1$ e mínimo igual a 1. Como regra da xPGD, nunca a primeira função de um programa é selecionada para ser mutada pelo OMR ou para ser enviada a outros robôs.

3. O espaço livre na estrutura do programa, isto é, no vetor que o representa, é calculado da seguinte forma: $P_{free} = N - P_{size} - P_{s_{size}}$. Onde, $P_{s_{size}}$ representa o valor, ou, o espaço ocupado pelo item (função ou terminal) que foi selecionado aleatoriamente para ser mutado pelo OMR.

4. Se $M_{size}$ for menor ou igual a $P_{free}$ o OMR é executado normalmente. Caso contrário, uma das seguinte alternativas será considerada para a execução do OMR:

   (a) conserva-se a primeira função da estrutura do programa e substitui o restante pelos itens recebidos em $M$, desde que $f_{arity} + M_{size} + 1$ seja menor ou igual a $P_{free}$, ou seja, o valor da aridade da primeira função somado com o tamanho de $M + 1$, deve ser menor ou igual ao espaço livre do vetor que representa $P$. Se a aridade da função for maior que um, completa-se sua estrutura com terminais selecionados aleatoriamente do conjunto de terminais do GB.

   (b) substitui toda a estrutura de $P$ pelos itens recebidos em $M$. Nesse caso, $P$ passará a ser igual a $M$.

É importante ressaltar que o OMR somente executará conforme as regras descritas em (a) e (b), isto é, em modo excepcional, após 3 (três) tentativas em encontrar um ponto na estrutura de $P$ que satisfaça a regra descrita em (4). Essa quantidade de vezes

foi estabelecida para evitar que o sistema fique indefinidamente tentando encontrar um ponto de corte na estrutura do indivíduo selecionado.

A Figura 4.5 ilustra o mesmo exemplo descrito na Seção 4.1.1 (Figura 4.2) com a representação dos indivíduos locais dos robôs 1 (a) e 2 (b) em formato vetorial. Os destaques na cor cinza do robô 1 (a) representam os itens (funções e terminais) selecionados para serem enviados para o robô 2 (b). A Figura 4.5 (c) ilustra a nova estrutura do indivíduo local do robô 2, que passa a ser um novo indivíduo que será testado na próxima geração $(i + 1)$.

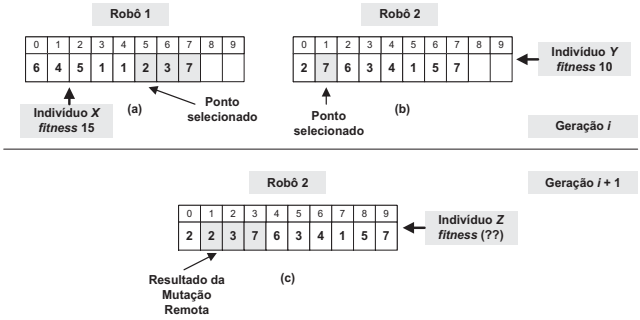


Figura 4.5: Representação vetorial dos indivíduos na xPGD.

Uma das vantagens em se utilizar uma estrutura vetorial para representar os indivíduos é a eliminação de um fenômeno conhecido por *bloat* (Langdon, 1998), ou seja, o crescimento indiscriminado do tamanho dos programas ou indivíduos sem melhora no valor da aptidão. Outra vantagem é o desempenho dos algoritmos dos operadores de cruzamento e mutação que não precisam manipular estruturas complexas como listas encadeadas alocadas dinamicamente em memória, como acontece com outras implementações da PG.

As mensagens trocadas entre os robôs são formadas pelo valor da aptidão do indivíduo local mais os itens a serem enviados, isto é, funções e/ou terminais. Todo o gerenciamento de envio e recebimento de mensagens é realizado pelo próprio RoBIOS. Cada robô EyeBot é equipado com um rádio de 433MHz com taxa de transferência de 9600 bauds. A rede formada entre os robôs é do tipo *token ring* virtual e sua configuração é feita de forma automática, tanto para a inserção quanto para a remoção de robôs, não sendo necessário a definição de um nodo mestre (Bräunl e Wilke, 2001).

Apesar de o SEGS ter sido desenvolvido para a plataforma EyeBot, tomou-se o cuidado de fazê-lo o mais modular e portável possível, para que com poucos modificações o sistema possa ser utilizado em outras plataformas. Com o propósito de verificar a portabilidade e avaliar o desempenho, os primeiros experimentos com o SEGS foram realizados com o simulador do robô Khepera e estão descritos no Capítulo 5 nas Seções 5.1.1 e 5.1.2, respectivamente.

## 4.4 Resumo do Capítulo

Este capítulo descreveu o algoritmo da xPGD. A xPGD é uma extensão do algoritmo da Programação Genética Distribuída para suportar a evolução do sistema de controle de uma população de robôs móveis.

Na xPGD os indivíduos são representados de forma linear, isto é, a estrutura de um indivíduo é um vetor contendo números inteiros. Cada número representa a identificação de uma função ou terminal. Os algoritmos de seleção e os operadores genéticos são executados sobre o vetor de inteiros.

O sistema que dá suporte a execução e ao gerenciamento da xPGD é chamado de SEGS (Sistema de Execução, Gerenciamento e Supervisão). Este sistema foi desenvolvido de forma modular para facilitar o processo de execução e gerenciamento do processo evolucionário que acontece de maneira embarcada em cada robô.

Cada robô pertencente a uma população de robôs móveis deve executar o SEGS. No SEGS o módulo de controle evolucionário é o responsável pela execução do algoritmo da xPGD. Internamente a este módulo existe uma população de programas, onde cada indivíduo é representado linearmente (vetor de inteiros), que a cada iteração são testados, selecionados e reproduzidos. No processo evolucionário a xPGD compartilha partes do melhor indivíduo local a um robô com todos os outros robôs da população de robôs.

A xPGD difere de outras abordagens em Evolução Embarcada pelas seguintes cacarterísticas: (1) cada robô da população de robôs possui internamente, de maneira embarcada, uma população de programas candidatos à solução do problema; (2) a configuração inicial do sistema é mais simples, pois basta o programador definir um conjunto de funções e de terminais no SEGS (o Sistema de Execução, Gerenciamento e Supervisão), a função de adaptação e outros parâmetros como, número máximo de gerações, tamanho

da população, probabilidade de cruzamento e mutação e o tamanho do vetor de codificação, e então recompilar o sistema; (3) uso da memória para guardar a estrutura do melhor indivíduo encontrado em cada geração, esta característica facilita o processo de recuperação de falhas.

# Capítulo 5

# Avaliação do Algoritmo da xPGD

Neste capítulo são apresentados os resultados de 3 (três) experimentos realizados com o algoritmo da xPGD (Extensão da Programação Genética Distribuída), que são: *navegação livre de colisões*, *forageamento* e *empurrar uma caixa*. Os experimentos foram realizados em ambiente simulado nos simuladores do robô Khepera e também do robô Eyebot. Ao final do capítulo são feitas algumas considerações adicionais sobre os experimentos realizados.

## 5.1 Descrição e Resultados dos Experimentos

Os experimentos escolhidos para a validação do algoritmo da xPGD possuem graus de complexidade distintos o que contribui para a validação do algoritmo proposto. Também vale ressaltar que tais experimentos são bem aceitos entre a comunidade de robótica móvel principalmente na área de robótica evolucionária.

Os experimentos (1) e (2) foram realizados no simulador do robô Khepera. Já o experimento (3) foi realizado no simulador do robô EyeBot. A troca do simulador do Khepera pelo simulador do EyeBot deu-se devido ao fato de o último apresentar mais recursos, tais como: suporte a programação concorrente, câmera CCD e comunicação via rádio enlace.

Para cada experimento foram definidos os seguintes parâmetros: número de robôs, número de gerações, tamanho da população local, tamanho do vetor da xPGD, probabilidade de cruzamento, probabilidade de mutação, método de seleção e a definição da função de avaliação para o cálculo do valor da aptidão. Cada experimento foi executado

uma determinada quantidade de vezes.

Como não existe uma maneira formal para definir parâmetros como: probabilidade de mutação, probabilidade de cruzamento e quantidade de gerações; os valores para estes parâmetros foram ajustados em cada execução. Desta forma os parâmetros que estão definidos em cada experimento foram resultados dos ajustes das diferentes execuções do experimento.

Os gráficos que demonstram os valores da média da aptidão da população em cada geração e o valor da aptidão do melhor indivíduo por geração foram gerados a partir dos valores obtidos com o resultado da melhor execução.

### 5.1.1 Experimento 1: Navegação Livre de Colisões

Nesse experimento um grupo composto por cinco robôs deve navegar ao longo de um ambiente não conhecido evitando colisões com obstáculos e com outros robôs (Perez *et al.*, 2008a). A simulação foi realizada com o simulador do robô Khepera versão 2.0 (Michel, 1996). Essa versão permite a simulação de um grupo de robôs capaz de se comunicaram entre si.

A Figura 5.1 ilustra a interface principal do simulador do Khepera. O ambiente simulado é composto por 5 (cinco) robôs dispostos aleatoriamente ao longo do ambiente e um conjunto de retângulos que representam os obstáculos a serem evitados pelos robôs.

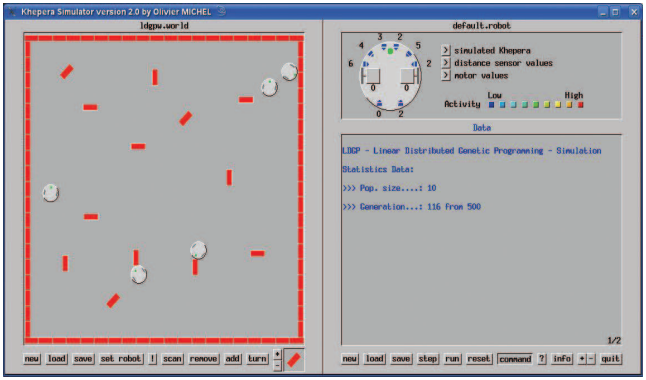


Figura 5.1: Interface principal do simulador do Khepera.

Cada robô Khepera possui oito (8) sensores de infravermelho para medir proximidade, oito (8) sensores de luz para medir a luminosidade do ambiente e duas rodas e motores

para movimentação. Cada sensor de infravermelho retorna um valor entre uma faixa de [0 - 1023], onde 0 significa que o obstáculo está muito próximo do robô e 1023 que o obstáculo não foi detectado. Os sensores de luz podem medir os níveis de luminosidade do ambiente e os valores retornados por cada leitura estão compreendidos entre a faixa [0 - 525], onde 0 significa máxima luminosidade e 525 ausência de luminosidade.

O conjunto de funções e de terminais utilizados nesse experimento estão descritos na Tabela 5.1. A coluna *Id.* representa a identificação de cada função e cada terminal.

Tabela 5.1: Conjunto de funções e de terminais do experimento 1.

| **Funções** | | | |
|---|---|---|---|
| **Nome** | **Aridade** | **Id.** | **Definição** |
| Prog1 | 1 | 2 | Executa um ramo da árvore. |
| Prog2 | 2 | 4 | Executa dois ramos da árvore. |
| Prog3 | 3 | 6 | Executa três ramos da árvore. |
| **Terminais** | | | |
| TurnRight | 0 | 1 | Virar a direita (15 graus). |
| TurnLeft | 0 | 3 | Virar a esquerda (15 graus). |
| GoForward | 0 | 5 | Seguir em frente. |
| Return | 0 | 7 | Retornar (300ms). |

As funções *Prog1*, *Prog2* e *Prog3* disparam a execução de outros elementos do programa, que podem ser as próprias funções ou os terminais. Os terminais *TurnRight* e *TurnLeft*, respectivamente, fazem o robô virar a direita e esquerda, somente se a leitura dos sensores, no caso os sensores dos lados direito e esquerdo não acusarem a existência de obstáculos. O terminal *GoForward* faz com que o robô vá para a frente somente se os sensores frontais não estiverem acusando a existência de obstáculos. O *Return* faz o robô retornar, dar a marcha à ré.

Os parâmetros usados na configuração do experimento da navegação livre de colisões estão descritos abaixo:

- **Número de robôs**: 5
- **Número de gerações**: 500
- **Tamanho da população local**: 10 indivíduos
- **Tamanho do vetor**: 50 posições

- **Probabilidade de cruzamento**: 60%
- **Probabilidade de mutação**: 10%
- **Método de seleção**: seleção por torneio com tamanho 6
- **Número de execuções**: 10 vezes
- **Função de avaliação**: método de punição e recompensa. `se CHOCOU COM OBSTÁCULO; fitness -= 3; senão fitness += 2`

O cálculo da função de avaliação é baseado no método de punição e recompensa. Basicamente quando um robô se choca com um obstáculo ele é punido em 3 pontos. Caso contrário, é recompensado em 2 pontos. Nenhum limite foi imposto para o valor da aptidão (*fitness*) de cada robô. O valor da aptidão poderia ser tanto negativo quanto positivo, nesse caso, os valores mais altos representam os melhores indivíduos em cada geração.

Em cada execução os robôs foram alocados em posições aleatórias do ambiente de simulação. Os gráficos da Figura 5.2 e 5.3 ilustram, respectivamente, a média do valor de aptidão da população e o valor de aptidão do melhor indivíduo em cada geração para cada robô.

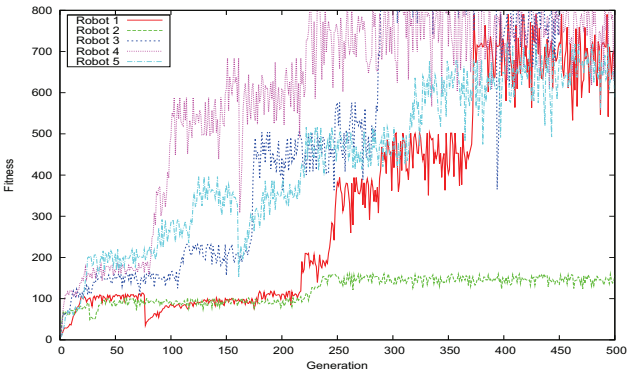


Figura 5.2: Média do valor de aptidão da população em cada geração para cada robô.

O gráfico da Figura 5.2 mostra que a média do valor de aptidão da população aumenta para cada robô. Entretanto, diferentemente dos outros robôs, o Robô número 2, não teve

um bom desempenho porque o mesmo ficou por um longo período de tempo executando em um dos cantos do ambiente.

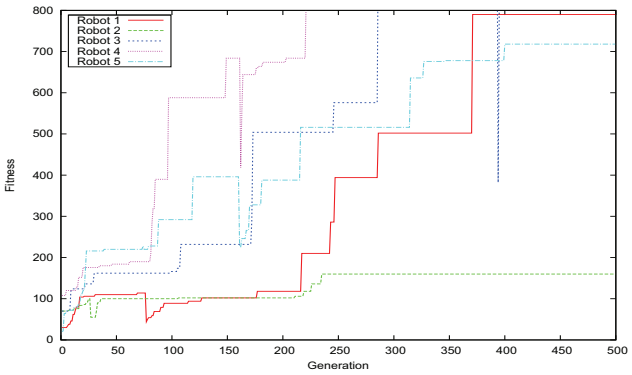


Figura 5.3: Valor de aptidão do melhor indivíduo em cada geração de cada robô.

O gráfico de desempenho do melhor indivíduo em cada geração de cada robô é ilustrado na Figura 5.3. O valor da aptidão do melhor indivíduo aumenta rapidamente por causa da variação genética causada toda vez que uma mensagem de outro robô e recebida e aceita para mutar um indivíduo dos $(t)$ melhores selecionados na população local.

A navegação livre de colisões é considerado um experimento de baixa complexidade na área de robótica móvel porque é possível desenvolver um sistema de controle simples que resolva bem o problema. Entretanto, para a área de robótica evolucionária, é considerado um bom problema para testar algoritmos evolucionários.

Nesse experimento os robôs deveriam aprender a navegar pelo ambiente sem se chocar com os obstáculos fixos e também se chocar com os outros robôs (obstáculos móveis). Os resultados obtidos após a simulação demonstraram que a xPGD, a partir da geração 200, foi capaz de evoluir o sistema de controle dos robôs para o problema proposto no experimento.

### 5.1.2 Experimento 2: Forageamento

Nesse experimento um grupo de robôs deve navegar por um ambiente desconhecido a procura de comida. Cada robô também deve evitar colisões com obstáculos e com outros robôs (Perez *et al.*, 2008b). O experimento foi executado no simulador do Khepera versão 2.0, o mesmo utilizado no Experimento 1.

A Figura 5.4 ilustra a interface principal do simulador do Khepera. No ambiente simulado existem 5 (cinco) robôs dispersos em posições aleatórias, um conjunto de retângulos que formam paredes e uma lâmpada que representa a comida.

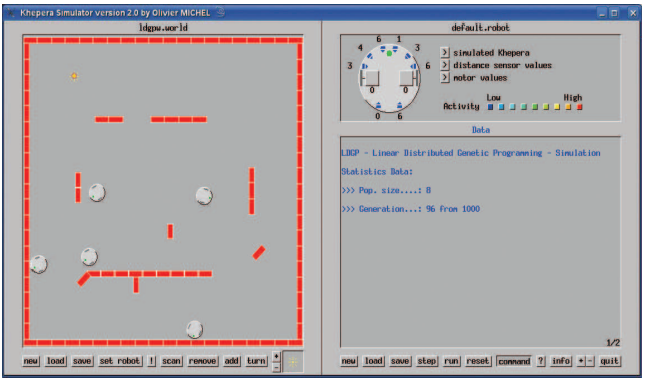


Figura 5.4: Interface principal do simulador do Khepera.

O objetivo de cada robô é navegar pelo ambiente procurando pela comida (lâmpada) e desviando de obstáculos (paredes e outros robôs). A Tabela 5.2 descreve os conjuntos de funções e de terminais usados no experimento. A coluna *Id.* representa a identificação de cada função e terminal.

Tabela 5.2: Conjuntos de funções e de terminais do experimento 2

| **Funções** | | | |
|---|---|---|---|
| **Nome** | **Aridade** | **Id.** | **Definição** |
| FoodAhead | 2 | 2 | Se encontrou comida, executa o nodo da esquerda; senão, executa o nodo da direita. |
| Prog2 | 2 | 4 | Executa dois ramos da árvore. |
| Prog3 | 3 | 6 | Executa três ramos da árvore. |
| **Terminais** | | | |
| TurnRight | 0 | 1 | Virar a direita (15 graus). |
| TurnLeft | 0 | 3 | Virar a esquerda (15 graus). |
| GoForward | 0 | 5 | Seguir em frente. |
| Return | 0 | 7 | Retornar (300ms). |

No experimento de forageamento as funções *Prog1* e *Prog2* e os terminais *TurnRight*, *TurnLeft*, *GoForward* e *Return* tem o mesmo comportamento das funções e terminais

definidos no primeiro experimento (Seção 5.1.1). A função *FoodAhead* faz a leitura dos sensores de luz para verificar o quão próximo o robô está da comida, caso a comida esteja próxima, é disparado a execução do nodo da esquerda, ou seja, o primeiro parâmetro da função, caso contrário, é executado o nodo da direita, o segundo parâmetro da função.

O experimento foi realizado tanto com o algoritmo da xPGD como o algoritmo tradicional da PG. O objetivo da comparação da xPGD com a PG foi o de avaliar o quão eficiente é o algoritmo da xPGD face ao algoritmo da PG. A PG foi escolhida para comparação porque a xPGD é um extensão da PG.

Os parâmetros usados na configuração do experimento de forageamento estão descritos abaixo:

- **Número de robôs**: 5
- **Número de gerações**: 1000
- **Tamanho da população**: 8 indivíduos
- **Tamanho do vetor**: 50 posições
- **Probabilidade de cruzamento**: 50%
- **Probabilidade de mutação**: 20%
- **Método de seleção**: seleção por torneio com tamanho 4
- **Número de execuções**: 10
- **Função de avaliação**: método de punição e recompensa. `se FOODAHEAD; fitness += 10; se SE CHOCOU COM OBSTÁCULOS; fitness -= 1; senão fitness += 1`

Assim como no Experimento 1 (Seção 5.1.1), não foi imposto nenhum limite para o valor da aptidão, sendo que o melhor indivíduo da população em cada geração possui um valor de aptidão maior e o pior um valor de aptidão menor, podendo este inclusive ser negativo.

As Figuras 5.5 e 5.6 ilustram os gráficos com o valor da aptidão do melhor indivíduo em cada geração para cada robô, com ambos os algoritmos, xPGD e PG, respectivamente.

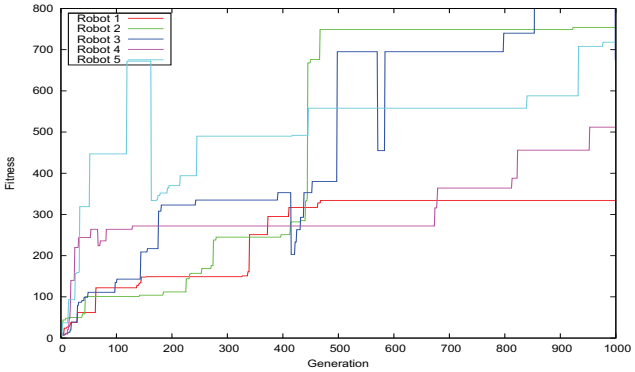


Figura 5.5: Aptidão do melhor indivídio em cada geração para cada robô com a xPGD.

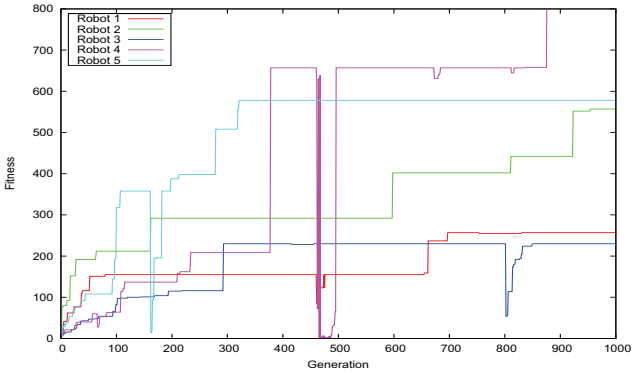


Figura 5.6: Aptidão do melhor indivídio em cada geração para cada robô com a PG.

Os resultados com o algoritmo da PG (Figura 5.6), demonstram que existe uma certa oscilação entre os valores de aptidão dos melhores indivíduos em cada geração. Os resultados obtidos com o algoritmo da xPGD (Figura 5.5) o valor da aptidão do melhor indivíduo tem uma tendência de crescimento contínuo.

Em ambos os algoritmos o valor da aptidão do melhor indivíduo não muda durante algumas gerações. Isso acontece por que um máximo local foi encontrado e o método de seleção empregado é a seleção por torneio com a manutenção dos pais após o cruzamento.

Os gráficos das médias do valor de aptidão de cada geração estão ilustrados nas Figuras 5.7 e 5.8 para ambos os algoritmos, xPGD e PG, respectivamente.

No algoritmo da PG (Figura 5.8) existem gerações onde a média do valor da aptidão

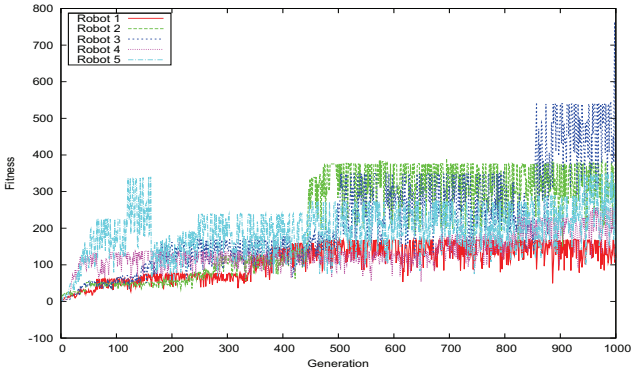


Figura 5.7: Média do valor de aptidão de cada geração para cada robô com a xPGD.

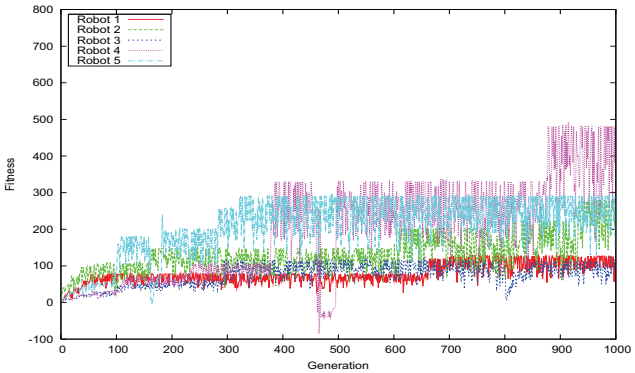


Figura 5.8: Média do valor de aptidão de cada geração para cada robô com a PG.

da população possui valor negativo. Esta situação não acontece com o algoritmo da xPGD (Figura 5.7) por causa da variação genética causada pela execução da operação de mutação remota, que faz com que o pior indivíduo da população local seja mudado com partes de um indivíduo que tenha um alto valor de aptidão em um outro robô.

Os resultados obtidos com o experimento demonstram que a xPGD conseguiu evoluir um sistema de controle para a tarefa de forageamento, que envolve a tarefa de navegação livre de colisões e a tarefa de localizar comida. A xPGD obteve resultados melhores que os conseguido com o uso da PG, ou seja, com o mesmo número de gerações a xPGD evoluiu um sistema de controle superior ao obtido com o uso da PG.

### 5.1.3 Experimento 3: Empurrar uma Caixa

Neste experimento um grupo de 3 (três) robôs precisam empurrar uma caixa, localizada no centro do ambiente, para qualquer direção (Perez *et al.*, 2009). O experimento foi realizado no simulador do robô Eyebot o EyeSim (*Eyebot Simulator*).

O simulador EyeSim (Koestler e Bräunl, 2004) é um ambiente *multithreading* que utiliza a mesma API (*Application Programming Interface*) da família de robôs Eyebot (Bräunl, 1999). O EyeSim permite que as aplicações utilizem uma câmera CCD, o sistema de comunicação de rádio frequência e os sensores de infravermelho. Para que a simulação se aproxime de um ambiente real o EyeSim possui um mecanismo de injeção de erros para os sensores de infravermelho, para a câmera e para o rádio.

A Figura 5.9 ilustra a interface do simulador Eyebot bem como a disposição dos robôs e da caixa no ambiente. Vale ressaltar que a interface principal do simulador se assemelha a um campo de futebol pois os robôs Eyebot são projetados para disputarem competições de futebol de robôs.

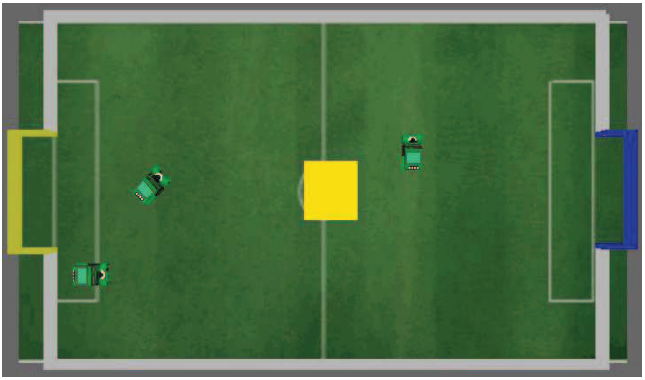

Figura 5.9: Interface principal do simulador Eyebot.

O conjunto de funções e de terminais utilizados nesse experimento estão descritos na Tabela 5.3. A coluna *Id.* representa a identificação de cada função e cada terminal.

No experimento de empurrar uma caixa as funções *Prog1*, *Prog2* e *Prog3* e os terminais *TurnRight*, *TurnLeft*, *GoForward* e *Return* tem o mesmo comportamento das funções e terminais definidos no primeiro e no segundo experimentos (Seção 5.1.1 e Seção 5.1.2).

Os terminais *TurnAroundRight* e *TurnAroundLeft* faz com que o robô gire para a direita ou a esquerda somente se os sensores do lado direito e do lado esquerdo não estiverem

Tabela 5.3: Conjunto de funções e de terminais para o experimento 3.

| **Funções** | | | |
|---|---|---|---|
| **Nome** | **Aridade** | **Id.** | **Definição** |
| Prog1 | 1 | 2 | Executa um ramo da árvore. |
| Prog2 | 2 | 4 | Executa dois ramos da árvore. |
| Prog3 | 3 | 6 | Executa três ramos da árvore. |
| **Terminais** | | | |
| TurnRight | 0 | 1 | Virar a direita (15 graus). |
| TurnLeft | 0 | 3 | Virar a esquerda (15 graus). |
| GoForward | 0 | 5 | Seguir em frente. |
| Return | 0 | 7 | Retornar (300ms). |
| BoxPushing | 0 | 9 | Empurrar a caixa. |
| TurnAroundRight | 0 | 11 | Girar para o lado direito. |
| TurnAroundLeft | 0 | 13 | Girar para o lado esquerdo. |

acusando a existência de obstáculos. O terminal *BoxPushing* (Empurrar a Caixa) é o responsável por detectar a localização da caixa. Para isso, nele está implementado uma rotina de detecção de cor que é utilizada para detectar a posição da caixa. Quando a cor alaranjada estiver sendo detectada pela câmera, o terminal Empurrar a Caixa, faz com que o robô vá em direção a ela, no caso, em direção a caixa, fazendo com que o mesmo a desloque de posição.

Os parâmetros usados na configuração do experimento de empurrar uma caixa estão descritos abaixo:

- **Número de robôs**: 3
- **Número de gerações**: 50
- **Tamanho da população**: 5 indivíduos
- **Tamanho do vetor**: 60 posições
- **Probabilidade de cruzamento**: 60%
- **Probabilidade de mutação**: 10%
- **Método de seleção**: seleção por torneio com tamanho 2
- **Número de execuções**: 5

- **Função de avaliação**: método de punição e recompensa. `se BOXPUSHING; fitness += 5; se SE CHOCOU COM OBSTÁCULOS; fitness -= 1; senão fitness += 2`

A função de avaliação é baseada no método de punição e recompensa, isto é, sempre que o robô encontrar a caixa e conseguir empurrá-la o valor da aptidão (*fitness*) é incrementado em 5. No caso de um choque com obstáculos, paredes ou outros robôs, a aptidão é decrementada em 1, caso contrário será incrementado em 2.

As Figuras 5.10 e 5.11 ilustram, respectivamente, os gráficos com o valor médio da aptidão por geração e o valor da aptidão do melhor indivíduo por geração.

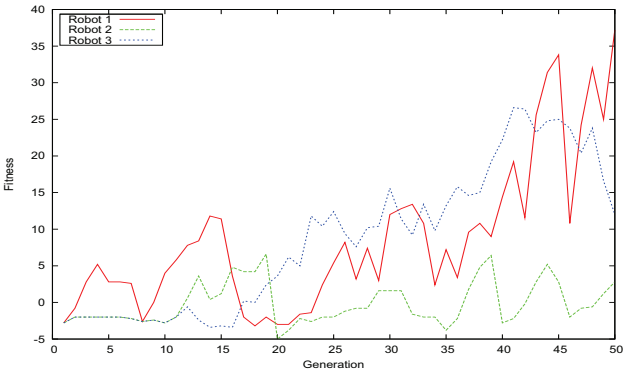


Figura 5.10: Média do valor de aptidão da população em cada geração para cada robô.

Durante a simulação todos os três robôs conseguiram tocar/empurrar a caixa. Entretanto, os robôs 1 e 3 tiveram um desempenho melhor, sendo que ambos conseguiram empurrar a caixa por um longo período de tempo, o que resultou num melhor valor de aptidão e conseqüentemente numa melhor média de aptidão da população local de cada robô.

O robô 2 conseguiu empurrar a caixa algumas vezes, mas na média geral teve um desempenho inferior aos outros dois. Mesmo tendo influência dos melhor indivíduos dos outros dois robôs, ainda assim não foi o suficiente para acompanhar a média global do valor de aptidão.

No gráfico da Figura 5.11 é possível perceber que a curva gerada pelo valor de aptidão do melhor indivíduo por geração de cada robô não difere muito da curva do gráfico da Figura 5.10, com os valores médios por população. O fato da xPGD ser um algoritmo

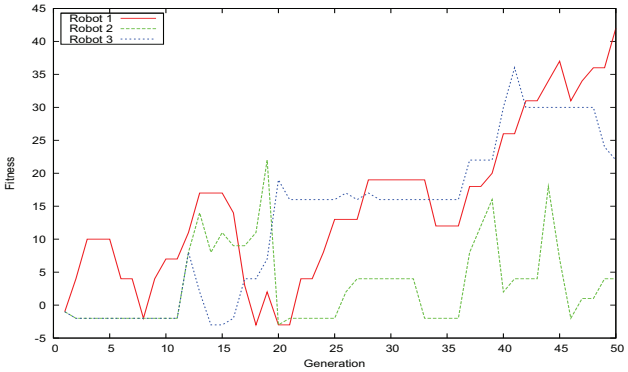


Figura 5.11: Valor de aptidão do melhor indivíduo em cada geração de cada robô.

elitista, isso é, mantém o melhor indivíduo de uma geração para outra, faz com que a melhor configuração para uma solução seja mantida e compartilhada entre os robôs da população de robôs.

O problema de empurrar uma caixa é parecido com o problema do forageamento, ou seja, é necessário que cada robô execute duas tarefas, que são: navegar sem se chocar com obstáculos e localizar e empurrar a caixa. Nesse caso não havia obstáculos fixos, somente obstáculos móveis, que eram os outros robôs do grupo.

## 5.2 Considerações Adicionais sobre os Experimentos com o xPGD

Como foi mencionado no início do capítulo os experimentos foram escolhidos baseado no grau de complexidade e também por serem bem aceitos na comunidade de pesquisadores de robótica evolucionária.

Os experimentos foram realizados somente em simulador, inicialmente foi utilizado o simulador dos robôs Khepera, e então o simulador foi substituído pelo simulador dos robôs Eyebot. A escolha pela simulador dos robôs Eyebot foi feita devido às funcionalidades do mesmo e também porque pretendia-se fazer os experimentos utilizando os robôs Eyebot.

Os experimentos com os robôs Eyebot não foram feitos por conta de alguns problemas encontrados durante o processo de testes. Um dos problemas era a pouca durabilidade das cargas das baterias, o que fazia com que os experimentos tivessem que ser reiniciados

a cada falha da bateria. Um outro problema enfrentado foi com relação ao rádio. Como é impossível gravar arquivos texto nos robôs, logo era necessário que os mesmos enviassem a cada geração uma mensagem para um servidor (computador *desktop*) contendo as seguintes informações: código do robôs, *fitness* do melhor indivíduo, média do *fitness* da população e número da geração. Entretanto, devido ao grande número de perda de pacotes, não foi possível obter informações confiáveis de cada robô, o que impossibilitou a geração de gráficos comparativos.

# Capítulo 6

# Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as conclusões sobre o trabalho desenvolvido nesta tese. Também são listadas algumas sugestões para trabalhos futuros que possam expandir e/ou contribuir com o trabalho desenvolvido.

## 6.1 Conclusões

As pesquisas em Robótica Evolucionária (RE) tratam da definição, implementação e experimentação de algoritmos evolucionários aplicados ao controle de robôs móveis. Em particular, a *Evolução Embarcada* (EE) trata de aplicar algoritmos evolucionários em uma população de robôs móveis, nesse caso, a população de soluções passa a ser a própria população de robôs.

Neste trabalho foi apresentado uma extensão do algoritmo da Programação Genética Distribuída (xPGD), capaz de suportar a evolução do sistema de controle de uma população de robôs móveis. Também foi apresentado o Sistema de Execução, Gerenciamento e Supervisão (SEGS), responsável pela execução e o gerenciamento da xPGD.

A xPGD foi desenvolvida como uma técnica para ser empregada na EE. Na EE em cada robô existe apenas um único indivíduo por população. Essa característica faz com que caso ocorra uma falha em um ou alguns dos robôs que compõem a população de robôs reais a busca pela solução pode ser prejudicada e até mesmo não ser concluída.

Uma das características da xPGD é a utilização de mais de um indivíduo por população em cada robô que compõem um grupo ou população de robôs reais. Isso faz com que, mesmo que ocorra um problema com um ou mais de um robô, a busca pela solução

continua mesmo com um desempenho menor. Se restar somente um único robô no grupo de robôs, esse é capaz de encontrar uma solução para o problema em questão.

Para avaliar o algoritmo da xPGD e o SEGS foram realizados três experimentos distintos envolvendo um grupo de robôs em ambiente simulado. No primeiro experimento um grupo de cinco robôs deveriam navegar por um ambiente não conhecido aprendendo a desviar de obstáculos. No segundo, também um grupo de cinco robôs deveriam navegar por um ambiente a procura de comida (forageamento), neste caso, além de localizar a comida, os robôs também deveriam aprender a desviar de obstáculos. O último experimento foi realizado com três robôs onde a tarefa principal era empurrar uma caixa localizada no centro do ambiente para qualquer direção.

Em todos os experimentos os robôs conseguiram atingir o objetivo proposto. Em um dos experimentos, o forageamento, os testes foram realizado tanto com a xPGD como com a PG clássica. Neste caso, os resultados obtidos com a xPGD foram mais satisfatórios dos que os obtidos com a PG.

O algoritmo da xPGD e o SEGS possuem as seguintes características:

- um novo algoritmo evolutivo baseado no algoritmo da Programação Genética Distribuída. O algoritmo desenvolvido pode ser aplicado tanto na evolução embarcada quanto na robótica evolutiva clássica, que conta com um único robô;

- um sistema para a Execução, o Gerenciamento e a Supervisão do algoritmo da PGD. Esse sistema é modular, visando facilitar o uso da PGD em problemas de robótica móvel;

- armazena em memória as informações sobre a estrutura dos indivíduos mais aptos para prover um mecanismo de recuperação de falhas;

- em cada robô, que compõe um grupo de robôs, é possível existir um ou mais indivíduos (programas), nesse caso, além de representar uma forma de prover a tolerância a falhas, quando robôs que fazem parte do grupo falharem, também ajuda a maximizar a evolução do controlador em cada robô;

- o SEGS provê um Gerenciador de Bibliotecas separado do módulo evolutivo. Esse gerenciador permite a inserção ou remoção de funções ou terminais dependo do problema que está sendo tratado;

- para facilitar o uso da PG em EE, foi criada uma maneira de representar os programas em cada robô baseado na Programação Genética Linear. Na xPGD cada programa é representado por um vetor de número inteiros onde cada número representa uma função ou terminal que está contido na estrutura do mesmo.

Como contribuição a área de Robótica Evolucionária, em particular a área de Evolução Embarcada, o trabalho desenvolvido nesta tese apresenta o uso de Programação Genética, no caso uma Extensão da Programação Genética Distribuída (xPGD), para evoluir o sistema de controle de uma população de robôs móveis, bem como, um sistema de controle capaz de se recuperar de erros causados na execução dos programas da população da xPGD.

## 6.2 Sugestões para Trabalhos Futuros

Visando aprofundar os estudos relativos a este trabalho propõem-se os seguinte tópicos como trabalhos futuros:

1. Adaptar o módulo de Controle Evolutivo do SEGS para que o mesmo possa avaliar semanticamente partes do programa de controle a fim de preservá-las em operações futuras de cruzamento e mutação;

2. Adaptar o módulo de Gerência de Bibliotecas do SEGS para que o mesmo seja capaz de aceitar uma nova função ou terminal em tempo de execução;

3. Permitir que o módulo de Controle Evolutivo trabalhe com "Definição Automática de Funções", técnica que permite decompor o programa em programas menores ou sub-rotinas;

4. Fazer outros experimentos com a PGD e o SEGS envolvendo um número maior de robôs e que permita a verificação da co-evolução, onde a adaptação de um indivíduo implica na adaptação de outro. A co-evolução acontece tipicamente em problemas como presa e predador, onde a adaptação do predador implica na adaptação da presa e vice versa.

Sugere-se também como trabalho futuro a reprodução dos experimentos descritos neste trabalho em robôs reais. Cada um dos experimentos pode ser reproduzido em alguma plataforma de robótica móvel com poucas alterações.

# Referências Bibliográficas


Alvarez, B., Iborra, A., Pastor, J. A., Fernández, C., Alonso, A., e de la Puenta, J. A. (2001). Software architecture for development of mechatronic systems: Service robots. *Dedicated Systems Magazine*, Vol. 4pp. 17–22.

Angeline, P. J. e Pollack, J. (1993). Evolutionary module acquisition. Proc. *of the 2nd Annual Conference on Evolutionary Programming*, pp. 154–163.

Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press.

Barlow, G. J. e Oh, C. K. (2006). Robustness analysis of genetic programming controllers for unmanned aerial vehicles. Proc. *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, pp. 135–142, Seattle, WA.

Barlow, G. J. e Oh, C. K. (2008). Evolved navigation control for unmanned aerial vehicles. Em Iba, H., editor, *Frontiers in Evolutionary Robotics*, chapter 20, pp. 353–378. I-Tech Education and Publishing, Vienna.

Bekey, G. A. (2005). *Autonomous Robots. From Biological Inspiration to Implementation and Control*. MIT Press. ISBN: 0-262-02578-7.

Bittencourt, G. (2006). *Inteligência Artificial Ferramentas e Teorias*. Editora da UFSC, Florianópolis - SC, 3ª edição. ISBN: 85-328-0138-2.

Blickle, T. e Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, Vol. 4, No. 4, pp. 361–394.

Bohm, W. e Geyer-Schulz, A. (1996). Exact uniform initialization for genetic programming. Proc. *Foundations of Genetic Algorithms IV*, pp. 379–407. Morgan Kaufmann, University of San Diego, CA, USA.

Brameier, M. e Banzhaf, W. (2001). A comparation of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionarz Computation*, Vol. 5, No. 1, pp. 17–26.

Broodier, M. e Banzhaf, W. (2006). *Linear Genetic Programming*. Springer.

Brooks, R. (1992). Artificial life and real robots. Proc. *European Conference on Artificial Life*, pp. 3–10.

Bräunl, T. (1999). Eyebot: A family of autonomous robots. Proc. *6th International Conference on Neural Information Processing*, pp. 645–649, Perth.

Bräunl, T. (2003). *Embedded Robotics*. Springer, Germany. ISBN: 3-540-03436-6.

Bräunl, T. (2006). Eyebot online documentation. http://robotics.ee.uwa.edu.au/eyebot/.

Bräunl, T. e Wilke, P. (2001). A self-configuring network for mobile robots. *Industrial Robot International Journal*, Vol. 28, No. 3, pp. 220–232.

Burke, E., Gustafson, S., e Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 1, pp. 47–62.

C. Ryan, J., Collins, J., e Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. Em W. Banzhaf, R. Poli, M. S. e Fogarty, T. C., editors, *First European Workshop on Genetic Programming*. Springer-Verlag.

Carvalho, A. C. P. L. F., Delbem, A. C. B., Romero, R. A. F., Simões, E. V., e Telles, G. P. (2004). Computação bioinspirada. Proc. *XXIV Congresso da Sociedade Brasileira de Computação*, Vol. 2 of *XXIII JAI*, pp. 145–191, Salvador, BA.

Cheang, S. M. e Leung, K. (2006). Genetic parallel programming: design and implementation. Proc. *Evolutionary Computation*. SUM.

Chellapilla, K. (1997). Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 3, pp. 209–216.

Coelho, L. d. S. (2003). *Fundamentos, Potencialidades e Aplicações de Algoritmos Evolutivos*. Notas em Matemática Aplicada. Sociedade Brasileira de Matemática Aplicada e Computacional.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequencial programs. Proc. *International Conference on Genetic Algorithms and their Applications*, pp. 183–187.

Daida, J. M. (1999). Challenges with verification, repeatability and meaningful comparisions in genetic programming. Proc. *of the 4th Annual Conference in Genetic Programming (GECCO-99)*, pp. 1069–1076. Morgan Kaufmann.

Dudek, G. e Jenkin, M. (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press.

Duffy, B. R. (2000). *The Social Robo*. Tese de Doutorado, University College Dublin, Dublin.

Eiben, A. E. e Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer.

Elfwing, S. (2007). *Embodied Evolution of Learning Ability*. Tese de Doutorado, KTH School of Computer Science and Communication, Stockholm, Sweden.

Fernández, F., Tomassini, M., e Vanneschi, L. (2002). Genetic algorithms and evolutionary computing. Proc. *Van Nostrand's Scientific Encyclopedia*.

Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, Vol. 13pp. 87–129.

Ficici, S., Watson, R., e Pollack, J. (1999). Embodied evolution: A response to challenges in evolutionary robotics. Em Wyatt, J. L. e Demiris, J., editors, *of the Eighth European Workshop on Learning Robots*, pp. 14–22.

Floreano, D. e Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. Proc. *of the Conference on Simulation of Adaptive Behavior*.

Floreano, D. e Urzelai, J. (2000). Evolutionary Robotics: The Next Generation. Em Gomi, T., editor, *Evolutionary Robotics III*. AAI Books, Ontario (Canada).

Fogel, D. (2000). What is evolutionary computation? *IEEE Spectrum*, Vol. 37, No. 2, pp. 28–32.

Fogel, D. B. (1992). *Evolving Artificial Intelligence*. Tese de Doutorado, The University of California, San Diego, CA, USA.

Francisco, T. e Jorge dos Reis, G. M. (2008). Evolving predator and prey behaviours with co-evolution using genetic programming and decision trees. Proc. *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pp. 1893–1900, New York, NY, USA. ACM.

Franzen, E. e Barone, D. A. (2003). Máquina inteligentes e a programação genética. Proc. *Sociedade Artificiais: a nova fronteira da inteligência nas máquinas*, pp. 75–91. Bookman.

Gruau, F. (1996). On syntactic constraints with genetic programming. Proc. *Advances in Genetic Programming II*, pp. 377–394. MIT Press.

Hainen, F. J. (2002). Sistema de controle híbrido para robôs móveis autônomos. Dissertação de Mestrado, UNISINOS, São Leopoldo, RS. Mestrado em Computação Aplicada - PIPCA.

Harvey, I. (1992). Species adaptation genetic algorithms: a basis for a continuing SAGA. Em Varela, F. J. e Bourgine, P., editors, *of the First European Conference on Artificial Life. Toward a Practice of Autonomous Systems*, pp. 346–354, Paris, France. MIT Press, Cambridge, MA.

Harvey, I. (1996). The microbial genetic algorithm. unpublished manuscript.

Harvey, I. (1997). Cognition is not computation; evolution is not optimisation. Em Gerstner, W., Hasler, M., e Nicoud, J.-D., editors, *Artificial Neural Networks – ICANN97, Proc. of 7th Int. Conf. on Artificial Neural Networks*, pp. 685–690, Berlin. Springer.

Haykin, S. (2001). *Redes Neurais: princípios e prática*. Bookman, 2ª edição.

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

Holland, J. M. (2004). *Designing Autonomous Mobile Robots - inside the mind of an intelligent machine*. Elsevier.

Hornby, G., Takamura, S., Hanagata, O., Fujita, M., e Pollack, J. B. (2000). Evolution of controllers from a high-level simulator to a high DOF robot. Proc. *ICES*, pp. 80–89.

Jakobi, N. (1998a). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, Vol. 6, No. 2, pp. 325–368.

Jakobi, N. (1998b). *Minimal Simulations for Evolutionary Robotics*. Tese de Doutorado, University of Sussex, England.

Jakobi, N., Husbands, P., e Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Lecture Notes in Computer Science*, Vol. 929pp. 704–720.

Jarvis, R. (2008). Intelligent robotics: Past, present and future. *International Journal of Computer Science and Applications*, Vol. 5, No. 3, pp. 23–35.

Jung, C. R., Osório, F. S., Kelber, C. R., e Heinen, F. J. (2006). Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. Proc. *XXV Congresso da Sociedade Brasileira de Computação*, pp. 1358–1406. SBC.

Kinnear, K. E. (1994). *Advances in Genetic Programming*. MIT Press.

Koestler, A. e Bräunl, T. (2004). Mobile robot simulation with realistic error models. Proc. *2nd International Conference on Autonomous Robots and Agents*, pp. 46–50, Palmerston North, New Zealand.

Kofod-Petersen, A. (2002). Adaptive behavior based robotics using on-board genetic programming. Dissertação de Mestrado, Norwegian University of Science and Technology.

Koza, J. R. (1992a). *Genetic Programming: A Paradigm for Genetically Breeding Computer Population of Computer Programs to Solve Problems*. MIT Press.

Koza, J. R. (1992b). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press. ISBN 0262111705.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Koza, J. R. (1995). Survey of genetic algorithms and genetic programming. San Francisco, CA. IEEE.

Koza, J. R. (1997). Genetic programming. Proc. *Encyclopedia of Computer Science and Technology*, pp. 1–26.

Kramer, M. D. e Zhang, D. (2000). Gaps: a genetic programming system. Em Press, I., editor, *The 24th Annual International Computer Software and Applications Conference*, pp. 614–619.

Langdon, W. B. (1998). The evolution of size in variable length representations. Proc. *IEEE International Conference on Evolutionarz Compuations (ICEC 1998)*, pp. 633–638.

Lee, W.-P., Hallam, J., e Lund, H. H. (1997). Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. Proc. *of IEEE 4th International Conference on Evolutionary Computation*, Vol. 1. IEEE Press.

Lingfeng Wang, Kay Chen Tan, C. M. C. (2006). *Evolutionary Robotics: from algorithms to implementations*. World Scientific Series In Robotics And Intelligent Systems - Vol. 28. World Scientific Publishing Company.

Luke, S. (2000). Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, pp. 274–283.

Luke, S. e Panait, L. (2001). A survey and comparison of tree generation algorithms. Em Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., e Burke, E., editors, *of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 81–88, San Francisco, California, USA. Morgan Kaufmann.

Luke, S. e Spector, L. (1998). A revised comparison of crossover and mutation in genetic programming. Em Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., e Riolo, R., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 208–213, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann.

Lund, H., Miglino, O., Pagliarini, L., Billard, A., e Ijspeert, A. (1998). Evolutionary robotics — a children's game. Proc. *of IEEE Fifth International Conference on Evolutionary Computation, NJ, 1998. IEEE Press.*

Martin, P. N. (2003). *Genetic Programming in Hardware*. Tese de Doutorado, University of Essex, UK.

Mataric, M. (2001). Situated robotics. Em Nadel, L., editor, *Encyclopedia of cognitive Science*. Nature Publishing Group, London.

Mataric, M. e Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems - Special Issue on Evolutional Robotics*, Vol. 19, No. 1, pp. 67–83.

Michel, O. (1996). Khepera simulator package version 2.0: Freeware mobile robot simulator written by oliver michel. http://www.epfl.ch/lami/team/michel/khep-sim/. Acessado em: 20/06/2007.

Miglino, O., Lund, H. H., e Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, Vol. 2, No. 4, pp. 417–434.

Mondada, F., Franzi, E., e Ienne, P. (1993). Mobile robot miniaturisation: A tool for investigation in control algorithms. Em Springer-Verlag, editor, *of the 3rd International Symposium on Experimental Robotics*, Berlin.

Murphy, R. R. (2000). *Introduction to AI Robotics*. The MIT Press.

Nehmzow, U. (2000). *Mobile Robotics: A Practical Introduction*. Springer.

Nehmzow, U. (2002). Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. Proc. *Second International Conference on Epigenetic Robotics*, pp. 115–123, Edinburgh, Scotland.

Nelson, A., Grant, E., Barlow, G., e White, M. (2003). Evolution of complex autonomous robot behaviors using competitive fitness. Proc. *IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, Boston, MA.

Nelson, A. L., Barlow, G. J., e Doitsidis, L. (2008). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, Vol. 57pp. 345 – 370.

Nelson, A. L., Grant, E., Galeotti, J., e Rhody, S. (2004a). Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and Autonomous Systems*, Vol. 46pp. 159–173.

Nelson, A. L., Grant, E., e Henderson, T. C. (2004b). Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, Vol. 46pp. 135–150.

Nolfi, S. e Floreano, D. (2001). *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press. ISBN: 0-262-14070-5.

Nolfi, S. e Floreano, D. (2002). Synthesis of autonomous robots through evolution. *Trends in Cognitive Science*, Vol. 6, No. 1, pp. 31–36.

Nolfi, S., Floreano, D., Miglino, O., e Mondada, F. (1994). How to evolve autonomous robots: different approaches in evolutionary robotics. Em R.Brooks, I. e (Eds.), P., editors, *of the International Conference Artificial Life IV*, pp. 190–197. MIT Press.

Nordin, P. e Banzhaf, W. (1995). Evolving turing-complete programs for a register machine with self-modifying code. Em Eshelman, L., editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pp. 318–325, Pittsburgh, PA, USA. Morgan Kaufmann.

Nordin, P. e Banzhaf, W. (1997a). An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour*, Vol. 5, No. 2, pp. 107–140.

Nordin, P. e Banzhaf, W. (1997b). Real time control of a khepera robot using genetic programming. *Cybernetics and Control*, Vol. 26pp. 533–561.

Oltean, M. e Dumitrescu, D. (2002). Multi expression programming. Relatório Técnico UBB-01-2002, Babes-Bolyai University, Romania.

Oltean, M. e Grosan, C. (2003a). A comparation of several linear genetic programming techniques. *Complex Systems*, Vol. 14, No. 4, pp. 285–313.

Oltean, M. e Grosan, C. (2003b). Solving classification problems using infix form genetic programming. Proc. *The Fifth International Symposium on Intelligent Data Analysis*, pp. 242;252, Berlin. M. Berthold (et. al ) ; LNCS Springer-Verlag.

Oltean, M., Grosan, C., Diosan, L., e Mihaila, C. (2009). Genetic programming with linear representation: a survey. *International Journal on Artificial Intelligence Tools*, Vol. 18, No. 2, pp. 197–238.

Pazos, F. (2002). *Automação de Sistemas e Robótica*. Editora Axcel.

Perez, A. L. F., Bittencourt, G., e Roisenberg, M. (2007a). An embodied evolutionary system to control a population of mobile robots using genetic programming. Proc. *Ninth Argentine Symposium on Artificial Intelligence (ASAI 2007)*, Mar del Plata, Argentina. Ana Maguitam and Daniela Godoy.

Perez, A. L. F., Bittencourt, G., e Roisenberg, M. (2007b). Um sistema evolutivo embarcado para controlar um população de robôs móveis usando programação genética. Proc. *VIII Simpósio Brasileiro de Automação Inteligente (SBAI 2007)*, Florianópolis, SC, Brasil.

Perez, A. L. F., Bittencourt, G., e Roisenberg, M. (2008a). Embodied evolution with a new genetic programming variation algorithm. Proc. *ICAS 2008 - Fourth International Conference on Automatic and Autonomous Systems*, pp. 118–123, Gosier, Guadeloupe. IEEE Computer Society Press.

Perez, A. L. F., Bittencourt, G., e Roisenberg, M. (2008b). A new approach to control a population of mobile robots using genetic programming. Proc. *SAC 2008 - Symposium on Applied Computing*, pp. 1602–1606, Fortaleza, Ceará, Brazil. ACM.

Perez, A. L. F., Bittencourt, G., e Roisenberg, M. (2009). Programação genética distribuída: um algoritmo evolutivo embarcado para controlar uma população de robôs móveis. Proc. *XXXV Conferência Latino-americana de Informática (CLEI 2009)*, Pelotas - RS.

Perkis, T. (1994). Stack-based genetic programming. Proc. *of the 1994 IEEE World Congress on Computational Intelligence*, Vol. 1, pp. 148–153, Orlando, Florida, USA. IEEE Press.

Poli, R. e McPhee, N. F. (2001). Exact GP schema theory for headless chicken crossover and subtree mutation. Proc. *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pp. 1062–1069, Seoul, Korea. IEEE Press.

Pollack, J. B., Lipson, H., Ficci, S., Funes, P., Hornby, G., e Watson, R. A. (2000). Evolutionary techniques in physical robotics. Proc. *ICES*, pp. 175–186.

Resnick, M., Berg, R., Eisenberg, M., e Turkle, S. (1997). Beyond black boxes: Bringing transparency and aesthetics back to scientific instuments. mit project funded by the national science fundation (1997-1999).

Ribeiro, C., Costa, A., e Romero, R. (2001). Robôs móveis inteligentes: Princípios e técnicas. Proc. *Anais do XXI Congresso da Sociedade Brasileira de Computação - SBC-2001*, pp. 258–306.

RoBIOS (2006). Robios library functions. http://robotics.ee.uwa.edu.au/eyebot/. Acessado em: 05/01/2008.

Rodrigues, E. L. M. (2002). Evolução de funções em programação genética orientada a gramáticas. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, PR. Programa de Pós-Graduação em Informática.

Rosca, J. P. (1996). Generality versus size in genetic programming. Proc. *Genetic Programming 1996: Proceedings of the First Annual Conference.*, pp. 381–387.

Rosca, J. P. (1997). *Hierarquical learning with procedural abstraction mechanisms*. Tese de Doutorado, University of Rochester, Rochester, NY.

Rosário, J. M. (2005). *Princípios de Mecatrônica*. Pearson. ISBN: 978-85-7605-010-0.

Russell, S. e Norvig, P. (2004). *Inteligência Artificial*. Editora Campus. ISBN: 85-352-1177-2.

Silveira, S. R. e Barone, D. A. C. (2003). Modelando comportamento inteligente com algoritmos genéticos. Em Barone, D. e. a., editor, *Sociedades Artificiais - A Nova Fronteira da Inteligência nas Máquinas*, pp. 61–73. Bookman.

Sim, K.-B., Lee, D. W., e Zhang, B.-T. (2002). Behavior evolution of autonomous mobile robot (amr) using genetic programming based on evolvable hardware. *International Journal of Fuzzy Logic and Intelligent Systems*, Vol. 2pp. 20–25.

Simões, E.V., D. A. C. B. (2002). Predation: an approach to improving the evolution of real robots with a distributed evolutionary controller. Proc. *International Conference on Robot Automation - ICRA*, pp. 664–669. IEEE Press.

Simões, E. d. V. (2000). *Development of an Embedded Evolutionary Controller to Enable Collision-free Navigation of a Population of Autonomous Mobile Robots*. Tese de Doutorado, The University of Kent at Canterbury, UK.

Simões, E. D. V. e Dimond, K. R. (2001). Embedding a distributed evolutionary system into population of autonomous mobile robots. Proc. *Proceeding of Internationa Conference on Systems, Man, and Cybernetics*, pp. 1069–1074, New York. IEEE Press.

Simões, E.V., K. R. D. (1999). An evolutionary controller for autonomous multi-robot systems. Proc. *International Conference on Systems, Man and Cybernetics*, pp. 664–669, Tokyo, Japan. IEEE Press.

Sofge, D. A., Potter, M. A., Bugajska, M. D., e Schultz, A. C. (2003). Challenge and opportunities of evolutionary robotics. Proc. *Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore.

Spears, W. M., Jong, K. A. D., Bäck, T., Fogel, D. B., e de Garis, H. (1993). An overview of evolutionary computation. Em Brazdil, P. B., editor, *of the European Conference on Machine Learning (ECML-93)*, Vol. 667, pp. 442–459, Vienna, Austria. Springer Verlag.

Spector, L. (1995). Evolving control structures with automatically defined macros. Proc. *Working Notes of the AAAI Fall Symposium on Genetic Programming 1995*, pp. 99–105. The American Association for Artificial Intelligence.

Team, S. R. (2006). Robot car stanley. http://cs.stanford.edu/group/roadrunner/stanley.html. Acessado em: 07/11/2006.

Thakoor, S., Morookian, J. M., Chahl, J., Hine, B., e Zormetzer, S. (2004). Bees: Exploring mars with bioinspired technologies. *Computer*, Vol. 37, No. 9, pp. 38–47.

Thomas Weise, K. G. (2006). Dgpf - an adaptable framework for distributed multi-objective search algorithms applied to the genetic programming of sensor networks.

Em Filipič, B. e Šilc, J., editors, *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, International Conference on Bioinspired Optimization Methods and their Application (BIOMA), pp. 157–166. Jožef Stefan Institute, Ljubljana, Slovenia.

Tomassini, M. (1995). A survey of genetic algorithms. *Annual Reviews of Computational Physics*, Vol. 3pp. 87–118.

Usui, Y. e Arita, T. (2003). Situated and embodied evolution in collective evolutionary robotics. Proc. *In Proc. of the 8th International Symposium on Artificial Life and Robotics*, pp. 212–215.

Wahde, M. (2004). Evolutionary robotics: the use of artificial evolution in robotics. Relatório Técnico TR-BBR-2004-001, Chalmers University of Technology.

Wang, L. (2002). Computational intelligence in autonomous mobile robotics-a review. Proc. *International Symposium on Micromechatronics and Human Science*, pp. 227–235.

Watson, A. H. e Parmee, I. C. (1997). Steady state genetic programming with constrained complexity crossover using species sub population. Em Back, T., editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pp. 315–321. Morgan Kaufmann.

Watson, R., Ficici, S., e Pollack, J. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, Vol. 39, No. 1, pp. 1–18.

Whitley, D. (2001). An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, Vol. 43, No. 14, pp. 817–831.

Whitley, D., Rana, S. B., Dzubera, J., e Mathias, K. E. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, Vol. 85, No. 1-2, pp. 245–276.

Wolf, D. F., Simões, E. d. V., Osório, F. S., e Trindade, O. (2009). Robótica móvel inteligente: da simulação às aplicações do mundo real. Proc. *XXIX Congresso da Sociedade Brasileira da Computação - Atualizações em Informática*, pp. 279–330, Bento Gonçalves - RS.

Yu, T. (1999). Structure abstraction and genetic programming. Em Press, I., editor, *of the 1999 Congress on Evolutionary Computation*, pp. 652–659.