Robótica Evolutiva

Anderson Luiz Fernandes Perez

Resumo

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. Dependendo do ambiente 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 dinâmico, ou seja, adaptável, é necessário utilizar alguma técnica de desenvolvimento que permita que o sistema se modifique ao longo de sua execução. A Computação Evolutiva (CE), através de seus Algoritmos Evolutivos, possibilita o desenvolvimento de sistemas de controle adaptativos para robôs móveis. A robótica evolutiva surgiu da utilização de técnicas de computação evolutiva para sintetizar automaticamente controladores para robôs com o propósito de treiná-los para desenvolver tarefas específicas. A RE pode ser descrita como um processo de evolução e adaptação contínuas 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. 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.

1.1. Introdução

Há muito tempo robôs vem sendo utilizados para auxiliar os seres humanos em tarefas consideradas repetitivas, estressantes ou perigosas. Os primeiros robôs geralmente eram teleoperados e não tinham nenhum grau de autonomia [Arkin 1998]. As pesquisas em robótica vêm evoluindo ao longo dos anos, atualmente robôs são utilizados nas mais variadas tarefas desde a navegação em chão de fábrica até missões de exploração espacial [Murphy 2000] [Thakoor et al. 2004].

As pesquisas em robótica visam cada vez mais dotar robôs de comportamentos inteligentes. A inteligência e a independência de um operador humano determinam o grau de autonomia de um robô. Um robô autônomo deve ser capaz de se adaptar ao ambiente para tomar decisões em situações para as quais ele não foi programado [Arkin

1998]. De acordo com [Bekey 2005] um robô autônomo é uma máquina inteligente capaz de executar tarefas em um determinado ambiente sem o controle explícito de um ser humano sobre seus movimentos.

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 dinâmico, ou seja, adaptável, é necessário utilizar alguma técnica de desenvolvimento que permita que o sistema se modifique ao longo de sua execução. A Computação Evolutiva (CE) [Fogel 2000], através de seus Algoritmos Evolutivos, possibilita o desenvolvimento de sistemas de controle adaptativos para robôs móveis [Pollack et al. 2000].

A Robótica Evolutiva (RE) [Nolfi and Floreano 2002] visa o desenvolvimento de sistemas de controle adaptativos baseado nas técnicas de CE . Uma das áreas de pesquisa da RE é a Evolução Embarcada (EE) [Watson et al. 2002], que une a RE à robótica cooperativa [Ficici et al. 1999], [Cao et al. 1997]. Na EE o processo evolutivo acontece entre os robôs de uma população de robôs móveis, ou seja a reprodução acontece entre os indivíduos que fazem parte da população de robôs.

Este capítulo está organizado como segue: na Seção 1.2 é descrita a teoria sobre a Computação Evolutiva, bem como seus principais algoritmos que são: Algoritmos Genéticos e Programação Genética; a Seção 1.3 descreve sobre a robótica evolutiva abordando seus principais aspectos e abordagens; Na Seção 1.4 são relatados alguns estudos de caso envolvendo a aplicação de algoritmos genéticos e programação genética no controle de robôs móveis; a Seção 1.5 encerra o capítulo com uma breve conclusão sobre os tópicos abordados.

1.2. Computação Evolutiva

A computação evolutiva (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 e otimização 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 adaptados 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 evolutivos (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 evolutiva:

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

- uma função de avaliação que associa a cada indivíduo um valor de adaptação (*fitness*) que determina o quão adaptado o indivíduo está.

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 1.1, descrito por [Michalewicz 1994], demonstra os principais passos de um AE.

Algoritmo 1.1: Algoritmo Evolutivo

$t \leftarrow 0$
inicialize $P(t)$
avalie $P(t)$
enquanto não condição de parada **do**
  $t \leftarrow t + 1$
  selecione $P(t)$ a partir de $P(t-1)$
  aplique os operadores genéticos
  avalie $P(t)$
fim enquanto

Um AE mantém uma população de indivíduos $P(t) = \{x_1^t, \ldots, x_n^t\}$ na geração t. Cada indivíduo representa um candidato à solução do problema em questão, e assume a forma de alguma estrutura de dados S. Cada solução $x^t{}_i$ é avaliada e produz alguma medida de aptidão (*fitness*). Uma nova população é formada na interação $t+1$ pela seleção dos indivíduos mais adaptados. Alguns indivíduos podem sofrer alterações por meio de operadores genéticos.

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 atributos de um dado indivíduo, esperando com isto que estes novos indivíduos estejam melhor adaptados às novas condições do meio.

O critério de parada é atendido quando a população converge 1 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 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 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 Evolutivas [Spears et al. 1993], Programação Evolutiva [Fogel 1992], Algoritmos Genéticos [Holland 1975] e Progamação Genética [Koza 1992b]. 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 1.2.1 e 1.2.2 descrevem as características e o funcionamento dos Algoritmos Genéticos e da Programação Genética, respectivamente.

1.2.1. 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 cromossomos 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 cromossomos artificiais usualmente criadas a partir de um processo randômico [Koza 1995]. Cada indivíduo da população é uma solução candidata ao problema em questão. O processo evolutivo é dirigido por uma função de avaliação.

A função de avaliação associa a cada indivíduo um valor de aptidão (*fitness*). Indivíduos são probabilisticamente selecionados, através de seu *fitness*, 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 2002].

1.2.1.1. Codificação

Em AGs cada solução parcial de um problema é codificada em uma *string* [Silveira and 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 2002].

1.2.1.2. Função de Avaliação

A função de avaliação 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 avaliação associa a cada indivíduo uma medida numérica (*fitness*) que determina o quão adaptado o indivíduo está [Tomassini 1995]. A função de avaliação 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 a função de adaptação do 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 *fitness* 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 1:

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

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:

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{m} a(k,t)} \qquad (2)$$

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).

1.2.1.3. 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 do *fitness* 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 and 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 3:

$$p_i = \frac{f_i}{\sum_i^N f_i} \tag{3}$$

onde $i \in \{1,2,\ldots,N\}$, f_i representa o valor de aptidão do indivíduo i e $\Sigma_i^N f_i$ 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, quanto 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 4:

$$p_i = \frac{1}{N}(n^- + (n^+ - n^-)\frac{i-1}{N-1}) \tag{4}$$

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 5:

$$p_i = \frac{c-1}{c^{N-1}} c^{N-i} \tag{5}$$

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 1.1 ilustra a operação de cruzamento e mutação em AGs.

A Figura 1.1 (a) ilustra um conjunto de indivíduos pertencentes a geração i que darão origem aos indivíduos da geração i+ 1 (Figura 1.1 (d)). Na Figura 1.1 (b), os indivíduos um e dois, e, três e quatro, cruzaram e geraram novos indivíduos. Na Figura 1.1 (c) houve a mutação genética em dois dos novos indivíduos gerados.

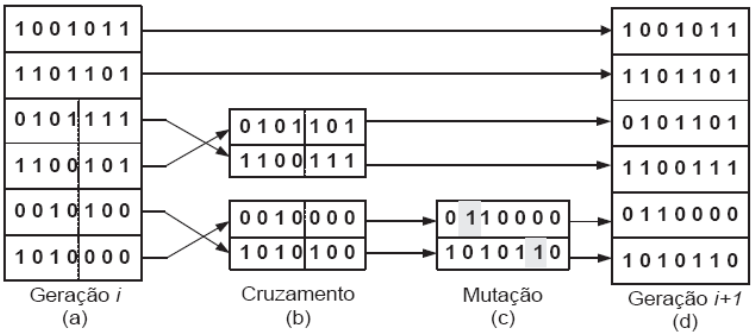


Figura 1.1. Exemplo dos operadores de cruzamento e mutação em AGs.

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.

1.2.1.4. 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.

1.2.1.5. SAGA - *Species Adaptation Genetic Algorithm*

SAGA foi proposto por [Harvey 1992] para resolver, entre outros, o problema da convergência prematura (problema do *bootstrap*) e possibilitar 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 adaptativa 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.

1.2.1.6. 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, o material genético do indivíduo menos adaptado é sobrescrito pelo material genético do mais adaptado. Nessa abordagem o menos adaptado sempre será o filho.

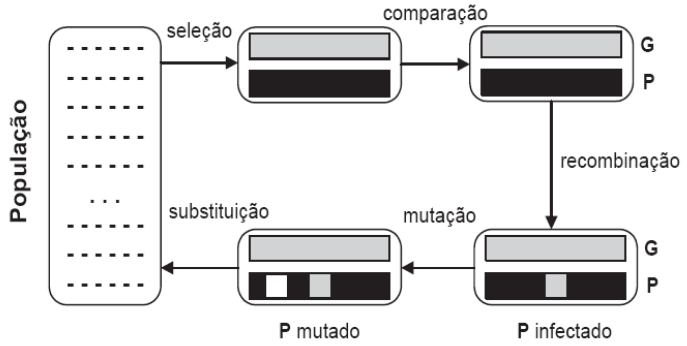


Figura 1.2. Reprodução no Microbial AG.

A Figura 1.2 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.

1.2.2. Programação Genética

A programação genética (PG) é uma técnica de geração automática de programas de computador criada por John Koza [Koza 1992b], 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 1992c].

Em PG um conjunto de programas de computador, que são soluções candidatas para resolver um determinado problema, são submetidos a um processo evolutivo. Através do processo evolutivo 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 adaptação de cada programa, aplica operadores genéticos para modificá-los e convergir para uma solução. Nessa técnica programas mais adaptados 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 de *fitness* 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 expressa 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 randômica 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 adaptação ao problema. O princípio da reprodução e sobrevivência do mais adaptado 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 1992b]. Abordagens baseadas na análise do processo evolutivo também podem ser usadas como critério de término, neste caso, o laço permanece enquanto houver melhoria na população [Kramer and Zhang 2000]. A Figura 1.3 ilustra o diagrama de blocos do algoritmo da PG.

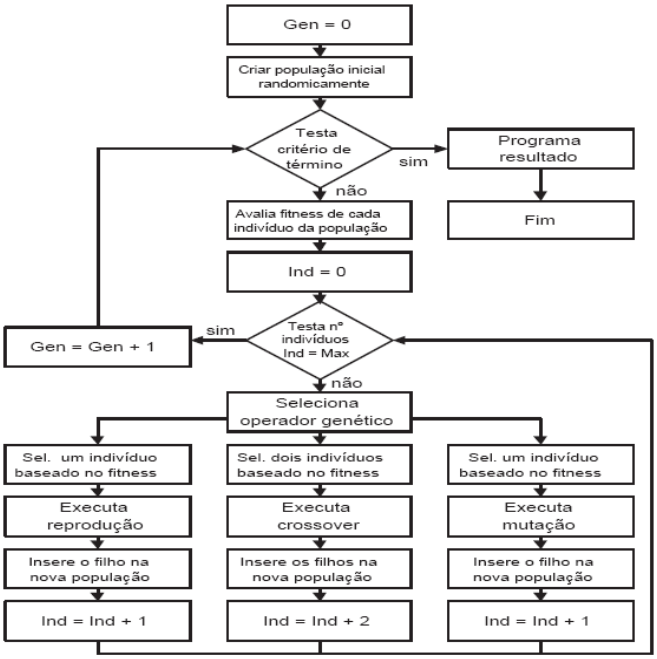


Figura 1.3. Diagrama de blocos do algoritmo da PG.

Programas encontrados nas gerações iniciais tendem a ser menos adaptados que indivíduos nas gerações seguintes. A cada nova geração criada os indivíduos tendem exibir uma melhor adaptação devido a pressão do *fitness* e 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 *fitness* 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 valores relativos ao domínio do problema [Franzen and Barone 2003].

1.2.2.1. 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 evolutivo 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 1992c]. 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 1.4 ilustra a representação de um programa genético com suas respectivas funções e terminais.

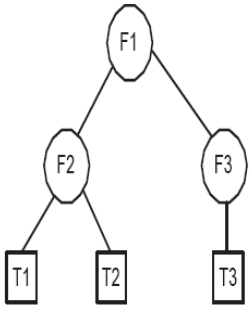


Figura 1.4. Estrutura de um programa em PG.

Para que a PG tenha um funcionamento correto [Koza 1992c] 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 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 1992c]. 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 evolutivos 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*.

1.2.2.2. 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, 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 1992b] 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 completas, ou seja, 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 randômico 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 umtamanhomáximo esperado E_{tree}. O 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 6:

$$p = \frac{1 - \frac{1}{E_{tree}}}{\sum_{n \in F} q_n b_n} \quad \textbf{(6)}$$

onde F é o conjunto de todas a funções e b_n 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. 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 and 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 and Panait 2001] fazem uma avaliação de desempenho dos algoritmos descritos acima em três problemas usando programação genética: Multiplexação Booleana, Regressão Simbólica e Formiga Artificial.

1.2.2.3. Função de Avaliação

Em PG, os programas que melhor resolverem o problema receberão melhores valores de *fitness*. O valor do *fitness* estabelece uma forma de diferenciar os melhores dos piores indivíduos em uma dada população [Koza 1992c]. Quanto mais adaptado está um indivíduo, maior a sua chance de permanecer ou ter suas características propagadas para as próximas gerações.

Em PG, assim como nos AGs, a função de avaliação, dependendo do domínio do problema, pode ser uma das que foram descritas na Seção 1.2.1.2.

1.2.3. Métodos de Seleção e Operadores Genéticos

Os métodos de seleção utilizados em PG são os mesmos que foram descritos na Seção 1.2.1.3 para AGs. Os operadores genéticos mais utilizados em PG são: reprodução, cruzamento e mutação.

- **Reprodução**: a operação de reprodução, também chamada de reprodução assexuada, 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 1.5 ilustra um exemplo do uso do operador de cruzamento.

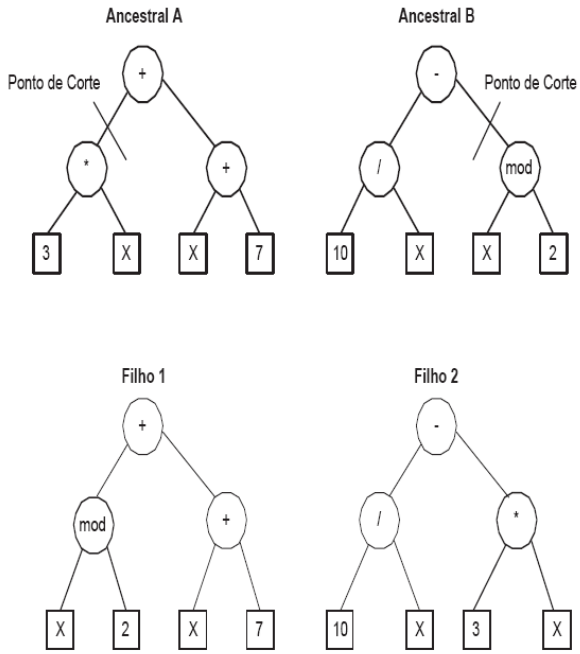


Figura 1.5. Exemplo do operador de cruzamento.

A operação de cruzamento modifica o tamanho dos programas, pois ao contrário do cruzamento em AG's onde um ponto de cruzamento é igual nos dois pais, na PG o 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 1992c].

- **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 1.6 ilustra um exemplo de aplicação do operador de mutação. O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população. A mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca possivelmente não será zero [Koza 1995].

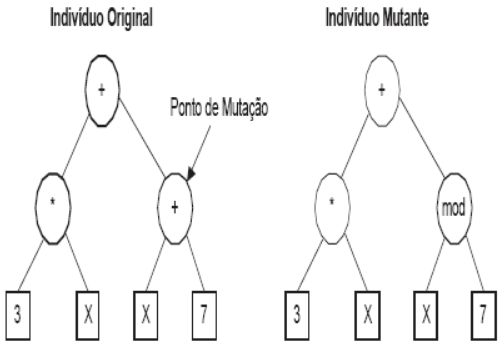


Figura 1.6. Exemplo do operador de mutação.

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 a diversidade da população [Carvalho et al. 2004].

Em PG existem outros operadores genéticos que são [Koza 1992c]: 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.

1.2.3.1. 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 and 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.

A função de avaliação é fundamental na PG e dependente do domínio do problema [Burke et al. 2004]. Usualmente para se proceder a avaliação da aptidão (*fitness*) 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.

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 avaliação. [Kramer and Zhang 2000] sugerem que o critério de término pode ser baseado no acompanhamento do processo evolutivo, ou seja, enquanto houver melhoria na média da população, o processo evolutivo prossegue.

1.2.3.2. 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 evolutivo 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 .

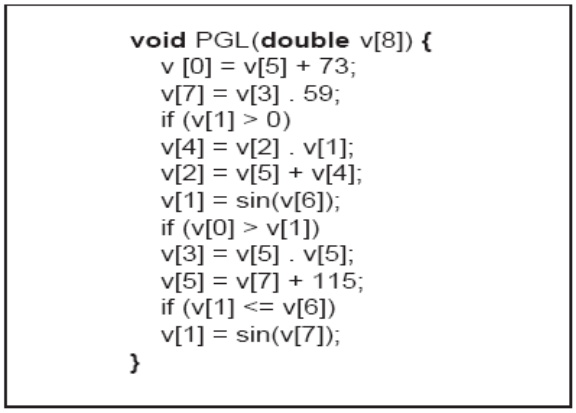
Outras extensões foram propostas para prover a Programação Genética da capacidade de evolução de funções, tais como: MA (*Module Acquisition*) [Angeline and Pollack 1993],
ADM (*Automatically Defined Macros*) [Spector 1995] e ARL (*Adaptive Representation through Learning*) [Rosca 1997].

1.2.3.3. 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 and 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 1.7 ilustra um exemplo de um programa em PGL.



```
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 1.7. Exemplo de um programa em PGL.

[Brameier and Banzhaf 2001] usaram em seus trabalhos uma estrutura similar ao da Figura 1.7, entretanto foi adicionada a capacidade de remoção de instruções ineficientes, 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 and 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 and Dumitrescu 2002], GE (*Gramatical Evolution*) [C. Ryan et al. 1998], GEP (*Gene Expression Programming*) [Ferreira 2001] e IFGP (*Infix Form Genetic Programming*) [Oltean and Grosan 2003b]. [Oltean and Grosan 2003a] 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.

1.3. Robótica Evolutiva

A Robótica Evolutiva (RE) surgiu da utilização de técnicas de computação evolutiva 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 and Floreano 2001]. A Figura 1.8 ilustra a diferença entre as duas abordagens.

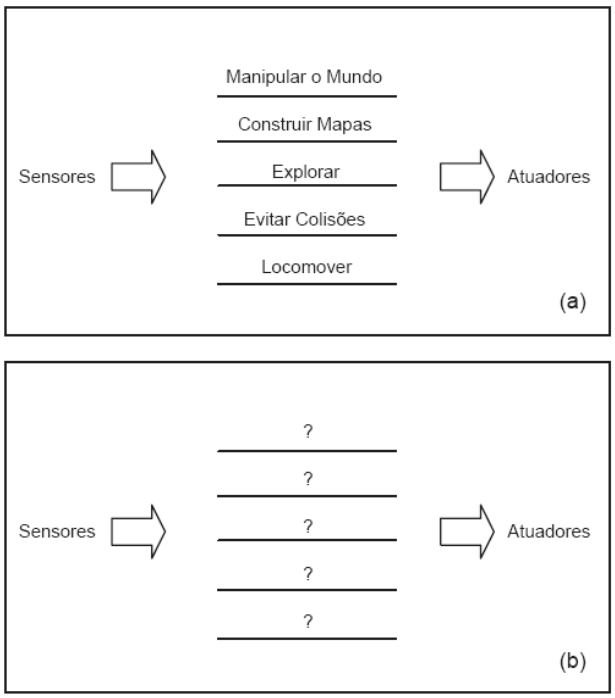


Figura 1.8. Comparação entre RBC (a) e RE (b).

Na Figura 1.8 (a), um exemplo de RBC, todos os comportamentos básicos descritos são intuitivamente projetados pelo programador ou projetista. Na RE, Figura 1.8 (b), a organização inteira do sistema evolutivo, incluindo sua organização em subcomponentes, é resultado de um processo de adaptação que usualmente envolve um grande número de interações entre o sistema e o ambiente.

A Figura 1.9 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.

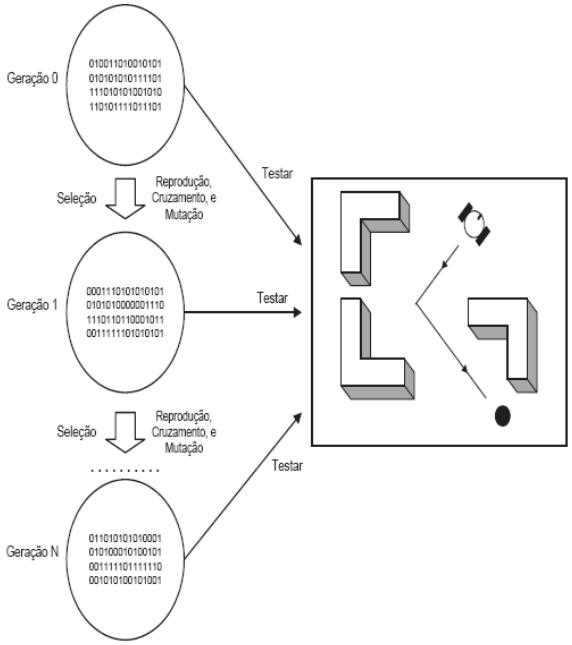


Figura 1.9. Visão geral da robótica evolutiva.

A computação evolutiva oferece meios para automatizar a geração de novas competências em agentes autônomos em um determinado ambiente. As principais questões abordas pela computação evolutiva são [Simões 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 evolutivas 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 e adaptação contínuas de um agente (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].

1.3.1. Função de Avaliação

Num processo de evolução artificial a função de aptidão ou avaliação (*fitness*) é usada para avaliar o desempenho dos indivíduos e selecionar os mais adaptados. O resultado

do processo de evolução depende muito do formato da função de avaliação [Floreano and Urzelai 2000].

Algumas abordagens definem a função de avaliação 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.

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

Visando auxiliar o projetista a definir uma boa função de avaliação, [Floreano and Urzelai 2000] propuseram uma ferramenta gráfica em três dimensões, que descreve e compara possíveis funções de avaliação para sistemas de controle para robôs autônomos. Uma função de avaliação representa um ponto neste espaço 3D. A Figura 1.10 ilustra esta representação tridimensional.

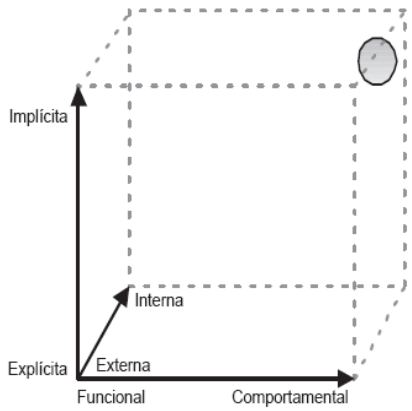


Figura 1.10. Função de avaliação num espaço tridimensional.

- **Dimensão Funcional-Comportamental**: uma função de avaliação puramente funcional é baseada na média ponderada entre as medidas de cada função do sistema de controle. Uma função de avaliação comportamental mede somente comportamentos individuais. A posição da função de avaliação ao longo da dimensão funcional 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 avaliação será computada usando informações internas ou externas ao sistema de controle do robô. Uma função de avaliação externa, ao contrário de uma função de avaliação interna, não pode ser medida diretamente pelo robô. Usualmente funções de avaliação 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 avaliação é definida depende dos objetivos do sistema. Se o objetivo for a otimização de parâmetros para um problema bem definido, a função de avaliação 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 avaliação deve estar no quadrante CII (Comportamental, Implícita, Interna) [Nolfi and Floreano 2001].

1.3.2. Abordagens da RE

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).

1.3.2.1. 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].

Num ambiente simulado é quase impossível representar virtualmente o ambiente real, onde eventualmente podem acontecer fatos que não foram previstos no simulador. De acordo com [Brooks 1992] é muito difícil simular o dinamismo do mundo real, 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 e 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 evolutivo é transferido para robôs reais. A dificuldade no uso da simulação mínima é determinar os conjuntos de características básicas e complementares.

1.3.2.2. Evolução em Robôs Reais (embarcada)

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. Esse tipo de ambiente, geralmente, é difícil de ser representado em simulador.

Quando o processo de evolução está embarcado 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 and Mondada 1994].

Um outra abordagem, chamada de Evolução Embarcada (EE), 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 de evolução acontece em um único robô [Watson et al. 2002]. De acordo com [Simões 2000], um sistema evolutivo embarcado é aquele em que o processo evolutivo 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 utiliza 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 and 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 num 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 and 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].

1.4. Estudos de Caso

Nesta seção são descritos alguns estudos de caso relacionados ao controle de robôs móveis com o uso de computação evolutiva. Os trabalhos relatados foram separados em dois grupos: Robótica Evolutiva com AGs (Seção 1.4.1) e Robótica Evolutiva com PG (Seção 1.4.2).

1.4.1. Robótica Evolutiva com AGs

[Ficici et al. 1999], inspirado no trabalho de [Mataric and 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 Seção 1.2.1.6).

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 et al. 1999] 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 1.11 ilustra a arquitetura neural utilizada por Ficici.

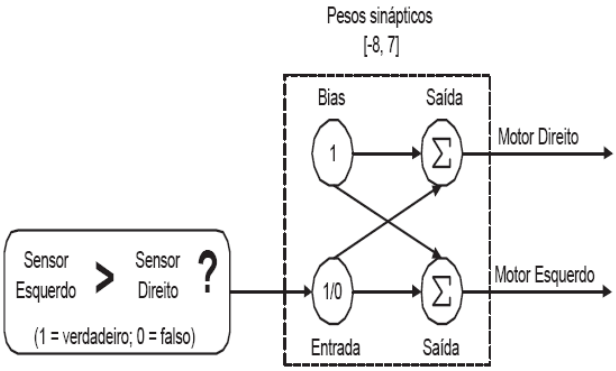


Figura 1.11. Arquitetura de controle utilizada nos experimentos de [Ficici et al. 1999].

Com seu trabalho [Ficici et al. 1999] 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 evolutivo 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 *fitness* do emissor era maior que o *fitness* 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 em todos os experimentos foram satisfatórios. O tempo médio para a convergência da população para uma determinada competência (tarefa) não levou mais 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*), S (*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 algoritmo genético 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 evolutivo 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) 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 avaliação 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 evolutivo 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 artigo [Simões and Dimond 2001] relata 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, e também a morfologia de cada robô. O experimento foi realizado com seis robôs tanto em simulador quanto com robôs reais. Diferente taxas de mutação foram testadas com o objetivo de avaliar a diversidade da população.

1.4.2. Robótica Evolutiva com PG

[Kofod-Petersen 2002], em sua dissertação de mestrado, 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 and Banzhaf 1997a], [Nordin and 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 and 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, *fitness*, identificação, etc; e 2) um programa em C. A Figura 1.12 ilustra um exemplo de representação linear dos indivíduos.

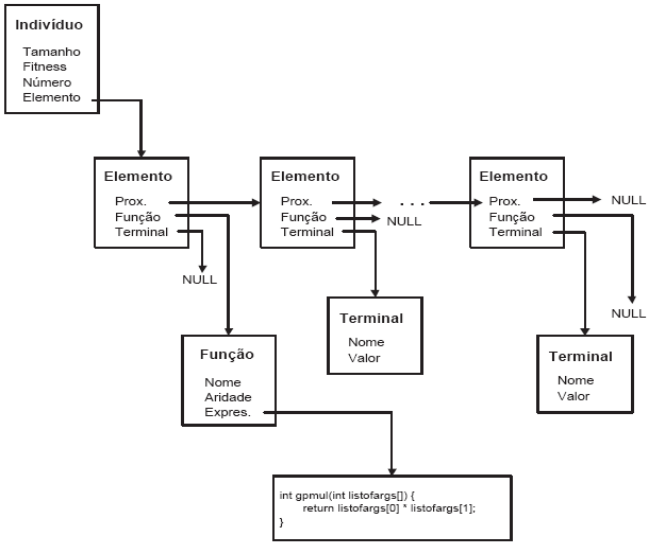


Figura 1.12. Representação linear dos indivíduos da população.

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 (cinquenta) indivíduos.

A função de avaliação utilizada nos experimentos de Petersen é representada na Equação 7.

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

onde α e β 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.

[Sim et al. 2002] utilizou 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 subárvores 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 numa forma de representação de hardware. Após a execução individual de cada subprocesso, os resultados parciais foram combinados para produzir o resultado final.

O experimento demonstrou que é possível combinar PG com hardware do tipo FPGA e obter bons resultados. Em apenas cinquenta gerações foi possível encontrar uma solução satisfatório 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.

[Perez et al. 2008] desenvolveu uma extensão do algoritmo tradicional da PG chamada Programação Genética Distribuída (PGD). A PGD é baseada no Microbial GA (ver seção 1.2.1.6), 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 PGD 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 *Ptotal* , 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 evolutivo ocorre sempre considerando a população total, ou seja, partes (sub-árvores) de um indivíduo local de um determinado robô podem ser consideradas no processo evolutivo da população local de outro robô.

Ao contrário de outras abordagens em Evolução Embarcada (EE), na PGD o processo evolutivo é assíncrono, isto é, não é necessário que dois robôs se sincronizem

para se reproduzirem. Partes de um indivíduo mais adaptado são enviados para todos os outros robôs. A seqüência de passos do algoritmo da PGD 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 (*fitness*);

(b) Receber uma mensagem *M* de um indivíduo remoto enviadas por outro robô;

(c) Selecionar os *t* melhores indivíduos da população local usando o método de seleção por torneio;

(d) Selecionar aleatoriamente uma parte do melhor indivíduo local (mais adaptado) e enviar uma mensagem *M*, contendo a parte selecionada mais o valor do *fitness*, em *broadcast* (difusão) para os outros robôs;

(e) Comparar se o *fitness* do pior indivíduo selecionado localmente é menor que o *fitness* do indivíduo remoto. Se sim, executa o operador de mutação 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 PGD, 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 8:

$$P_{local}(I) = \left\{ \begin{array}{ll} OMR(I) & \text{if } Fitness(I_{remoto}) > Fitness(I_{local}) \\ I & \text{if } Fitness(I_{remoto}) \leq Fitness(I_{local}) \end{array} \right\} \quad \textbf{(8)}$$

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 do *fitness* do indivíduo remoto ($Fitness(I_{remoto})$) for maior que o valor do *fitness* 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 PGD 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 and Parmee 1997].

As mensagens trocadas entre os robôs devem conter o *fitness* 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 PGD garante a continuidade do processo evolutivo do sistema de controle, mesmo quando houver algum problema 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 evolutivo.

Para avaliar o algoritmo da PGD foi realizado um experimento com um conjunto de robôs que devem navegar por um ambiente a procura de comida. Esse problema é conhecido como forageamento [Perez et al. 2008]. A Tabela 1.1 lista o conjunto de funções e terminais usados nesse exemplo.

Tabela 1.1. Conjunto de funções e terminais para o problema do forageamento.

Funções			
Nome	**Aridade**	**Id.**	**Definição**
ComidaFrente	2	2	Se encontrou comida, executa nodo da esquerda; senão, executa nodo 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).

Na Tabela 1.1 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 1.13 ilustra o mecanismo de mutação remota da PGD. Nela são representados dois robôs e um indivíduo da população local de cada um.

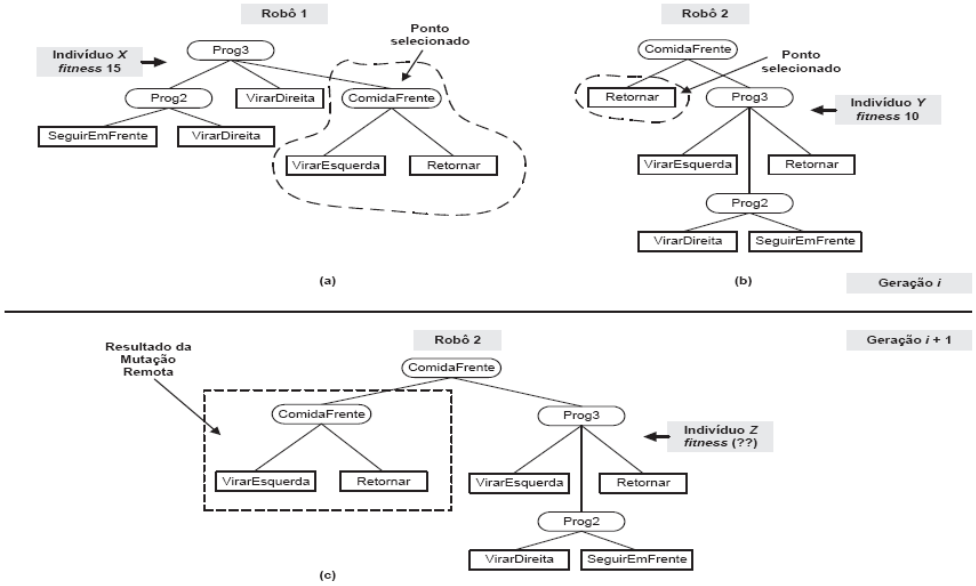


Figura 1.13. Exemplo de funcionamento da função de mutação remota da PGD.

Na Figura 1.13 (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 1.1, 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: M= {15,2,3,7}. Isto é, o valor do *fitness* 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 1.13 (b) irá comparar o valor do *fitness* da mensagem com o valor do *fitness* do indivíduo local (*Y*). Se a condição imposta pela Equação 8 for satisfeita, isto é, se o valor do *fitness* da mensagem, ou seja, do indivíduo remoto, for maior que o valor do *fitness* do indivíduo local, a operação de mutação remota é executada. Nesse caso, uma 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 PGD 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.

Para a execução da PGD, em cada robô, é necessário a existência de um sistema que dê suporte a execução e ao gerenciamento de todo o processo evolutivo, que ocorre de forma embarcada. O sistema desenvolvido com esse propósito chama-se SEGS (Sistema de Execução, Gerenciamento e Supervisão).

O SEGS é responsável pela execução, gerência e supervisão de todo o processo evolutivo em cada robô de um SMR [Perez et al. 2007a]. A Figura 1.14 ilustra, genericamente, o funcionamento do SEGS.

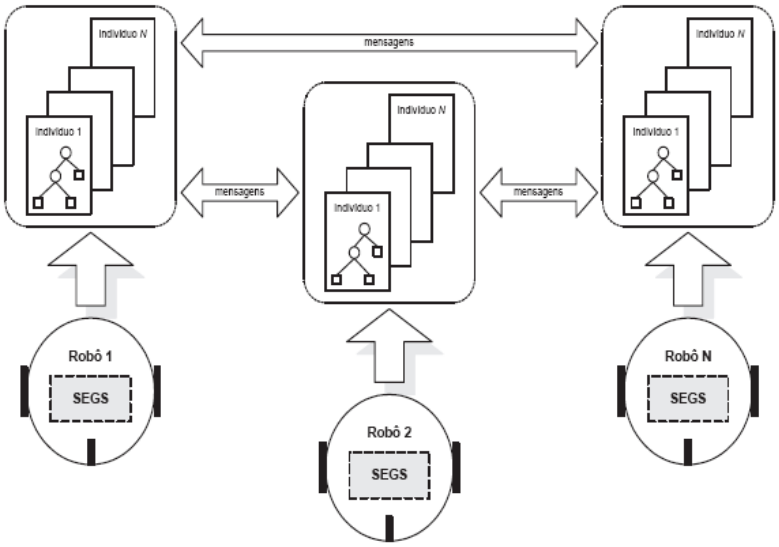


Figura 1.14. Representação esquemática do SEGS.

O SEGS é executado em cada robô que faz parte de um grupo de robôs, onde existe uma população local 6 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 de um grupo de robôs.

O SEGS é estruturado em módulos que são ilustrados na Figura 1.15. Cada módulo é responsável por uma etapa do processo evolutivo.

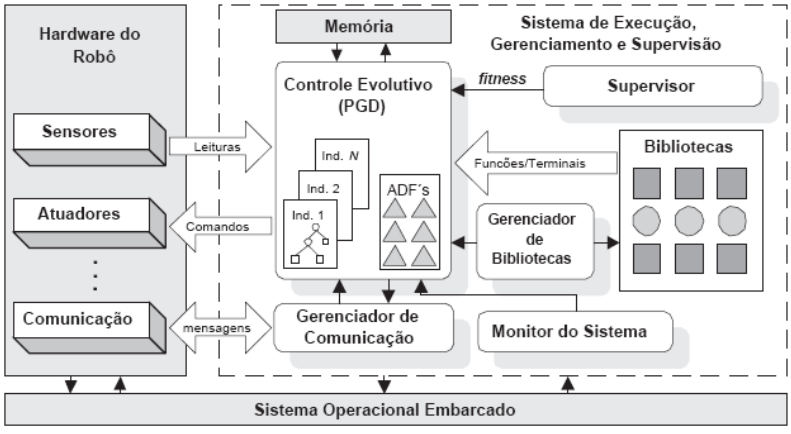


Figura 1.15. Estrutura modular do SEGS.

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 Evolutivo (CTE)**: é o componente principal do SEGS, é nele que está implementado a PGD. 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 adaptados em cada geração. Por exemplo, considerando o problema do forageamento, o melhor indivíduo do Robô 1, de acordo com a coluna *Id.* da Tabela 1.1, 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, como por exemplo, desviar de obstáculos e deslocar um objeto de um lugar para outro. 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.

- **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 do *fitness* 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 poderem ser enviadas para outros robôs pelo GC. Ter as bibliotecas separadas do sistema evolutivo 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 *fitness* 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 do *fitness* 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 evolutivo 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 and 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.

Outros trabalhos em robótica móvel envolvendo programação genética podem ser encontrados em: [Koza 1992a], [Koza and Rice 1992], [Lee and Zhang 2000] e [Lazarus and Hu 2001].

1.5. Conclusão

A Robótica Evolutiva 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 evolutiva, sintetizar automaticamente controladores 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 evolutivo 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 evolutivo. Entretanto, com o uso de um simulador, é quase impossível representar virtualmente o ambiente real, onde eventualmente pode acontecer fatos que não foram previstos no simulador.

Na RE com robôs reais o processo evolutivo acontece internamente, ou seja, embarcado no hardware do robô. Também é possível evoluir um sistema de controle entre vários robôs, nesse caso o processo evolutivo 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 evolutivo pode acontecer totalmente em um simulador e o resultado ser transferido para o robô real a fim de ser avaliado.

1.6. Referências

[Angeline and Pollack 1993] Angeline, P. J. and Pollack, J. (1993). Evolutionary module acquisition. In *of the 2nd Annual Conference on Evolutionary Programming*, pages 154–163.

[Arkin 1998] Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press. [Bekey 2005] Bekey, G. A. (2005). *Autonomous Robots. From Biological Inspiration to Implementation and Control*. MIT Press. ISBN: 0-262-02578-7.

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

[Blickle and Thiele 1996] Blickle, T. and Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394.

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

[Brameier and Banzhaf 2001] Brameier, M. and Banzhaf, W. (2001). A comparation of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionarz Computation*, 5(1):17–26.

[Broodier and Banzhaf 2006] Broodier,M. and Banzhaf,W. (2006). *Linear Genetic Programming*. Springer.

[Brooks 1992] Brooks, R. (1992). Artificial life and real robots. In *European Conference on Artificial Life*, pages 3–10.

[Burke et al. 2004] Burke, E., Gustafson, S., and Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62.

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

[Cao et al. 1997] Cao, Y. U., Fukunaga, A. S., and Kahng, A. B. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4,:7–27.

[Carvalho et al. 2004] Carvalho, A. C. P. L. F., Delbem, A. C. B., Romero, R. A. F., Simões, E. V., and Telles, G. P. (2004). Computação bioinspirada. In *XXIV Congresso da Sociedade Brasileira de Computação*, volume 2 of *XXIII JAI*, pages 145–191, Salvador, BA.

[Chellapilla 1997] Chellapilla, K. (1997). Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216.

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

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

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

[Elfwing 2007] Elfwing, S. (2007). *Embodied Evolution of Learning Ability*. PhD thesis, KTH School of Computer Science and Communication, Stockholm, Sweden. [Ferreira 2001] Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13:87–129.

[Ficici et al. 1999] Ficici, S., Watson, R., and Pollack, J. (1999). Embodied evolution: A response to challenges in evolutionary robotics. pages 14–22.

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

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

[Fogel 2000] Fogel, D. (2000). What is evolutionary computation? *IEEE Spectrum*, 37(2):28–32.

[Fogel 1992] Fogel, D. B. (1992). *Evolving Artificial Intelligence*. PhD thesis, The University of California, San Diego, CA, USA.

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

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

[Harvey 1992] Harvey, I. (1992). Species adaptation genetic algorithms: a basis for a continuing SAGA. In Varela, F. J. and Bourgine, P., editors, *of the First European*

Conference on Artificial Life. Toward a Practice of Autonomous Systems, pages 346–354, Paris, France. MIT Press, Cambridge, MA.

[Harvey 1996] Harvey, I. (1996). The microbial genetic algorithm.

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

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

[Hornby et al. 2000] Hornby, G., Takamura, S., Hanagata, O., Fujita, M., and Pollack, J. B. (2000). Evolution of controllers from a high-level simulator to a high DOF robot. In *ICES*, pages 80–89.

[Jakobi 1998a] Jakobi, N. (1998a). Evolutionary robotics and the radical envelope-ofnoise hypothesis. *Adaptive Behavior*, 6(2):325–368.

[Jakobi 1998b] Jakobi, N. (1998b). *Minimal Simulations for Evolutionary Robotics*. PhD thesis, University of Sussex, England.

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

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

[Kofod-Petersen 2002] Kofod-Petersen, A. (2002). Adaptive behavior based robotics using on-board genetic programming. Master's thesis, Norwegian University of Science and Technology.

[Koza 1992a] Koza, J. R. (1992a). Evolution of subsumption using genetic programming. In *of the First European Conference on Artificial Life. Towards a Practice of Autonomous Systems*, pages 110–119, Barcelona.

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

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

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

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

[Koza 1997] Koza, J. R. (1997). Genetic programming. In *Encyclopedia of Computer Science and Technology*, pages 1–26.

[Koza and Rice 1992] Koza, J. R. and Rice, J. P. (1992). Automatic programming of robots using genetic programming. In *of Tenth National Conference on Artificial Intelligence*, pages 194–201. AAAI Press/MIT Press.

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

[Lazarus and Hu 2001] Lazarus, C. and Hu, H. (2001). Using genetic programming to evolve robot behaviours. In *of the 3 rd BritishWorkshop on Towards IntelligentMobile Robots (TIMR) '01, Manchester*.

[Lee and Zhang 2000] Lee, K.-J. and Zhang, B.-T. (2000). Learning robot behaviors by evolving programs. In *26th Annual Conference of the IEEE Industrial Electronics Society*, volume 4, pages 2867–2872, Nagoya, Japan.

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

[Luke 2000] Luke, S. (2000). Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283.

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

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

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

[Martin 2003] Martin, P. N. (2003). *Genetic Programming in Hardware*. PhD thesis, University of Essex, UK.

[Mataric and Cliff 1996] Mataric, M. and Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems - Special Issue on Evolutional Robotics*, 19(1):67–83.

[Michalewicz 1994] Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

[Miglino et al. 1995] Miglino, O., Lund, H. H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434.

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

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

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

[Nelson et al. 2003] Nelson, A., Grant, E., Barlow, G., and White, M. (2003). Evolution of complex autonomous robot behaviors using competitive fitness. In *IEEE International Conference on Integration of Knowledge IntensiveMulti-Agent Systems*, Boston, MA.

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

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

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

[Nolfi and Floreano 2002] Nolfi, S. and Floreano, D. (2002). Synthesis of autonomous robots through evolution. *Trends in Cognitive Science*, 6(1):31–36.

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

[Nordin and Banzhaf 1995] Nordin, P. and Banzhaf, W. (1995). Evolving turingcomplete programs for a register machine with self-modifying code. In Eshelman, L., editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 318–325, Pittsburgh, PA, USA. Morgan Kaufmann.

[Nordin and Banzhaf 1997a] Nordin, P. and Banzhaf, W. (1997a). An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour*, 5(2):107–140.

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

[Oltean and Dumitrescu 2002] Oltean, M. and Dumitrescu, D. (2002). Multi expression programming. Technical Report UBB-01-2002, Babes-Bolyai University, Romania.

[Oltean and Grosan 2003a] Oltean,M. and Grosan, C. (2003a). A comparation of several linear genetic programming techniques. *Complex Systems*, 14(4):285–313.

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

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

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

[Perez et al. 2008] Perez, A. L. F., Bittencourt, G., and Roisenberg, M. (2008). A new approach to control a population of mobile robots using genetic programming. In *SAC 2008 - Symposium on Applied Computing*, Fortaleza, Ceará, Brazil. ACM.

[Perkis 1994] Perkis, T. (1994). Stack-based genetic programming. In *of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA. IEEE Press.

[Poli and McPhee 2001] Poli, R. and McPhee, N. F. (2001). Exact GP schema theory for headless chicken crossover and subtree mutation. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1062–1069, COEX,World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.

[Pollack et al. 2000] Pollack, J. B., Lipson, H., Ficci, S., Funes, P., Hornby, G., and Watson, R. A. (2000). Evolutionary techniques in physical robotics. In *ICES*, pages 175– 186.

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

[Rodrigues 2002] Rodrigues, E. L. M. (2002). Evolução de funções em programação genética orientada a gramáticas. Master's thesis, Universidade Federal do Paraná, Curitiba, PR. Programa de Pós-Graduação em Informática.

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

[Rosca 1997] Rosca, J. P. (1997). *Hierarquical learning with procedural abstraction mechanisms*. PhD thesis, University of Rochester, Rochester, NY. [Silveira and Barone 2003] Silveira, S. R. and Barone, D. A. C. (2003). Modelando comportamento inteligente com algoritmos genéticos. In Barone, D. e. a., editor, *Sociedades Artificiais - A Nova Fronteira da Inteligência nas Máquinas*, pages 61–73. Bookman.

[Sim et al. 2002] Sim, K.-B., Lee, D. W., and 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*, 2:20–25.

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

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

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

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

[Spears et al. 1993] Spears, W. M., Jong, K. A. D., Bäck, T., Fogel, D. B., and de Garis, H. (1993). An overview of evolutionary computation. In Brazdil, P. B., editor, *of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria. Springer Verlag.

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

[Thakoor et al. 2004] Thakoor, S., Morookian, J. M., Chahl, J., Hine, B., and Zormetzer, S. (2004). Bees: Exploring mars with bioinspired technologies. *Computer*, 37(9):38– 47.
[Tomassini 1995] Tomassini,M. (1995). A survey of genetic algorithms. *Annual Reviews of Computational Physics*, 3:87–118.

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

[Wahde 2004] Wahde, M. (2004). Evolutionary robotics: the use of artificial evolution in robotics. Technical Report TR-BBR-2004-001, Chalmers University of Technology.

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

[Watson et al. 2002] Watson, R., Ficici, S., and Pollack, J. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18.

[Whitley 2001] Whitley, D. (2001). An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831.
[Whitley 2002] Whitley, D. (2002). Genetic algorithms and evolutionary computing. In *Van Nostrand's Scientific Encyclopedia*.

[Whitley et al. 1996] Whitley, D., Rana, S. B., Dzubera, J., and Mathias, K. E. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276. [Yu 1999] Yu, T. (1999). Structure abstraction and genetic programming. In Press, I., editor, *of the 1999 Congress on Evolutionary Computation*, pages 652–659.