Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Co-evolução de Agentes Inteligentes Antagonistas usando Algoritmos Genéticos

Jhonatan da Rosa

Murillo Titon de Souza

Monografia de Conclusão do Curso de Bacharelado em Sistemas de Informação

Orientadora: Luciana de O. Rech

Florianópolis, 15 de Maio de 2012

Co-evolução de Agentes Inteligentes Antagonistas usando Algoritmos Genéticos

Trabalho de Conclusão de Curso apresentado como parte das atividades para obtenção do título de Bacharel em Sistemas de Informação da Universidade Federal de Santa Catarina.

Prof[a] orientadora: Luciana de O. Rech

Florianópolis, 2012

RESUMO

Área de grande interesse dentro da Ciência da Computação, as pesquisas sobre Inteligência Artificial se ramificaram, e por volta da década de 50, ao se estabelecer um paralelo entre aprendizado e as teorias evolutivas oriunda da Biologia, surgia o paradigma de Computação Evolutiva. Um destaque nesse campo são os algoritmos genéticos, que buscam reproduzir os conceitos da seleção natural de Charles Darwin. Por ser focado em buscar soluções viáveis para problemas de otimização genéricos, trabalhos que usem o paradigma de algoritmos genéticos no campo de jogos são relativamente escassos. Este trabalho objetiva verificar a viabilidade do uso de algoritmos genéticos para criar adversários versáteis em jogos ao simular um simples jogo envolvendo dois ou mais tipos de agentes assimétricos antagonistas. Para tal, é proposto co-evoluir populações destes agentes, que usarão umas as outras como parte integrante de seu meio-ambiente, e analisar os comportamentos exibidos por estes ao final de sua evolução. Enquanto não se espera que seja encontrada, e nem se presume que haja, uma estratégia perfeita para qualquer dos lados, os agentes devem demonstrar ao longo do experimento uma evolução nos padrões de comportamento análogos àqueles que seriam observados durante o aprendizado de um ser inteligente.

Palavras-chave: Algoritmo Genético, Co-evolução, Jogos.

ABSTRACT

An area of great interest in Computer Science, the research on Artificial Intelligence ramified, and around the 1950s, when a parallel was established between learning and the evolution theories from Biology, the paradigm of Evolutionary Computing arose. One of the most proeminent techniques in this field are genetic algorithms, which seek to reproduce the concepts of natural selection established by Charles Darwin. Because they are focused in finding viable solutions for generical optimization problems, works that use genetic algorithms in the field of electronic games are fairly scarse. This work aims to verify the viability of using genetic algorithms to create versatile opponents in games by simulating a simple game involving two or more kinds of assymetric antagonistic agents. To achieve that, populations of these agents are to be co-evolved using each other as part of their environment and their exhibited behaviour is to be analyzed after their evolution. While it is not expected that a perfect strategy will be found, and neither is it presumed that one even exists, the agents are expected to show an evolution in their behaviour patterns analogous to those observed during the learning process of an intelligent being.

Keywords: Genetic Algorithm, Co-evolution, Games.

SUMÁRIO

1. INTRODUÇÃO 1
1.1. MOTIVAÇÃO 3
1.2. OBJETIVOS 3
1.2.1. OBJETIVOS ESPECÍFICOS 3
1.3. ORGANIZAÇÃO DO TEXTO 4
1.4. LIMITAÇÕES DA PESQUISA 4
2. FUNDAMENTAÇÃO TEÓRICA 5
2.1. ALGORITMOS GENÉTICOS 5
2.1.1. DEFINIÇÃO 6
2.1.2. ESQUEMA DE ALGORITMO GENÉTICO 6
2.1.3. REPRESENTAÇÃO CROMOSSOMIAL 8
2.1.4. DEFINIÇÃO DE *SCHEMAS* 10
2.1.5. OPERAÇÃO DE SELEÇÃO (REPRODUÇÃO) 11
2.1.6. FUNÇÃO DE AVALIAÇÃO (OU APTIDÃO) 13
2.1.7. OPERAÇAO DE *CROSSOVER* 13
2.1.8. OPERAÇÃO DE MUTAÇÃO 15
2.2. BINARY DECISION DIAGRAM (BDD) 16
2.3. CANAL DE FLUXO 17
3. TRABALHOS RELACIONADOS 19
3.1. ALGORITMOS GENÉTICOS EM ESTRATÉGIAS DE GRUPO 19
3.2. SIMULAÇÃO DE ECOSSISTEMAS USANDO CO-EVOLUÇÃO 21
3.3. OTIMIZAÇÃO DE ESTRATÉGIAS EM JOGOS 23

3.4. CO-EVOLUÇÃO DE AGENTES ANTAGONISTAS 24
3.4.1. JOGO DE CAÇA.. 24
3.4.2. CO-EVOLUÇÃO HIERÁRQUICA 26
3.5. ALGORITMOS GENÉTICOS E POPULAÇÕES COOPERANTES 27
3.6. SIMULAÇÃO DE COEVOLUÇÃO.. 28
3.7. AGENTE INTELIGENTE PARA O JOGO LEMMINGS 30
3.8. ALGORITMOS GENÉTICOS NO JOGO MASTERMIND 33
3.9. AGENTE DE PACMAN USANDO COMPUTAÇÃO EVOLUTIVA 35
3.10. COMPARAÇÕES ENTRE OS ARTIGOS 37
4. PROPOSTA 39
4.1. CENÁRIO.. 39
4.2. MODELAGEM DO CROMOSSOMO 40
4.3. IMPLEMENTAÇÃO DOS ALGORITMOS.. 40
4.4. TRABALHO DESENVOLVIDO.. 44
4.4.1. Parâmetros do Algoritmo Genético 44
4.4.2. Balanceamento do Cenário... 45
4.4.3. Interface Gráfica.. 46
4.4.4. Processo de tomada de decisão dos indivíduos.................................. 47
4.4.5. Comportamento de Ecossistema 49
5. CONSIDERAÇÕES FINAIS ....**Error! Bookmark not defined.**
6. REFERÊNCIAS BIBLIOGRÁFICAS... 51

LISTA DE FIGURAS

Figura 1. Diagrama do funcionamento de um motor de algoritmo genético 8

Figura 2. BDD para a função (x1 – y1) ^ (x2 – y2) (AKERS, 1978) 16

Figura 3. Exemplo de *n-BDD* (AKERS, 1978) 17

Figura 4. Canal de fluxo (SCHELL, 2008, p. 122) 18

Figura 5. Exemplo de n-BDDs em algoritmos genéticos. 40

Figura 6. Pseudo-código do loop de simulação 41

Figura 7. Pseudo-código da função *step* do simulador 42

Figura 8. Pseudo-código da função *step* dos herbívoros e dos carnívoros 43

Figura 9. Pseudo-código do funcionamento do algoritmo genético 44

Figura 10. Interface gráfica do simulador 47

Figura 11. Detalhamento dos sensores visuais de um indivíduo 48

Figura 12. Exemplo de árvore de decisão 48

Figura 13. Exemplo de atribuição de pesos de um indivíduo 49

Figura 14. Estado da grade após o carnívoro (azul) executar a ação de caçar 49

Figura 15. Desempenho das espécies durante simulação 50

LISTA DE TABELAS

Tabela 1. Exemplo de cromossomo para uma cadeia de aminoácidos 10

Tabela 2. Exemplo de *crossover* em ponto único entre duas *strings* de *bits* 14

Tabela 3. Ordem de verificação do algoritmo nos testes 40

Tabela 4. Relação de condição/ação dos testes iniciais 44

1. INTRODUÇÃO

O cérebro humano tem a capacidade de focar sua atenção seletivamente; um exemplo disso é demonstrado quando conseguimos manter a atenção em uma única conversa em uma festa onde várias ondas sonoras estão atingindo nossos ouvidos ao mesmo tempo. Quando algo captura a atenção e imaginação humana por um longo período, o cérebro humano entra em um estado diferenciado, denominado por psicólogos como Mihaly Csikszentmihalyi como “canal de fluxo” (SCHELL, 2008).

Manter os jogadores dentro do canal de fluxo é o que projetistas de jogos buscam ao criar um jogo, e para fazê-lo é necessário fornecer um nível de desafio condizente com a habilidade do jogador no jogo em questão. Quanto maior a discrepância entre a habilidade do jogador e a habilidade exigida pelo jogo para superar seus desafios, maior a probabilidade de que o jogador tenha uma experiência negativa e acabe desistindo de continuar: baixa dificuldade faz com que o jogador se entedie, já que não é necessário sua absoluta atenção; alta dificuldade gera ansiedade no jogador em função das seguidas frustrações (SCHELL, 2008). Logo, para que a experiência seja o mais satisfatória possível para o jogador, é necessário que a dificuldade apresentada pelo jogo seja continuamente condizente com a habilidade do jogador em resolver os problemas propostos.

Considerando que jogos são normalmente feitos para serem apreciados por uma vasta gama de pessoas, achar uma única curva de dificuldade que funcione para todos é um objetivo irreal. Por isso, nos últimos anos, algumas empresas começaram a utilizar o conceito de dificuldade adaptável: dependendo do desempenho do jogador, o jogo muda seu funcionamento para que a dificuldade seja mais condizente com a habilidade do jogador. Um exemplo notório é o jogo *Left 4 Dead*, da empresa *Valve Corporation*: nesse jogo, uma inteligência artificial denominada *AI Director* gerencia a quantidade de inimigos que surgem para enfrentar os jogadores, além da quantidade de munição e itens disponíveis no mapa para uso dos jogadores baseado na performance destes (WIKIPEDIA, 2011).

Desde o princípio da era computacional, recriar comportamentos considerados inteligentes é um objetivo constante entre os pesquisadores da área. A área de jogos se destacou desde o início dessas pesquisas como uma plataforma para teste de hipóteses e

avaliação de implementações ao permitir uma fácil asserção da qualidade das estratégias tomadas pelos algoritmos. Entretanto, enquanto essa abordagem é válida para a descoberta e teste de novas técnicas, normalmente o objetivo é obter um programa que jogue da melhor maneira possível, ou seja, que confira ao jogo a maior dificuldade possível ao adversário. Por isso, inteligências artificiais comumente se mostram adversários intransponíveis para jogadores humanos a não ser que sejam propositalmente pioradas para fins de diversão.

Ao longo das décadas de 50 e 60, vários pesquisadores da área de computação estudaram independentemente sistemas evolucionários sob a premissa de que estes poderiam ser utilizados como uma ferramenta de otimização para problemas de engenharia. Segundo (MITCHELL, 1998; GOLDBERG, 2009), John Holland definiu o conceito de algoritmos genéticos em meados da década de 60, e o desenvolveu com o auxílio de alunos e colegas ao longo das décadas de 60 e 70. Contrastando com outras técnicas de programação evolucionária (EIBEN et al., 2003), o objetivo de Holland não era obter algoritmos que resolvessem problemas específicos, mas sim estudar formalmente o fenômeno da adaptação e desenvolver maneiras de importar os mecanismos de adaptação natural para uso em sistemas computacionais.

O presente trabalho busca retomar o enfoque de Holland ao buscar não uma otimização para um problema específico mas sim observar a adaptação de agentes evoluindo pela ação de um algoritmo genético. Para tal, é proposto recriar o cenário utilizado por (MORIWAKI et al., 1997), que simula um ecossistema simples onde duas populações antagonistas e assimétricas de agentes inteligentes (“carnívoros” e “herbívoros”) são evoluidas utilizando algoritmos genéticos utilizando uma à outra como meio-ambiente. Espera-se que o avanço na qualidade das decisões tomadas pelos agentes de ambas as populações seja passível de ser considerado como o aprendizado demostrado por um ser inteligente, com a finalidade de futuramente aplicar essa técnica na inteligência artificial de oponentes de jogos eletrônicos para que estes venham a se adaptar continuamente às habilidades do jogador, mantendo-o por mais tempo entretido pelo jogo.

1.1. MOTIVAÇÃO

A motivação deste trabalho advém da afinidade com jogos e sistemas inteligentes, e do imenso potencial ainda existente em sua intersecção mesmo após anos de pesquisa envolvendo os dois temas.

Ao pesquisar sobre aplicação de algoritmos genéticos em agentes para jogos, notou-se que a grande maioria dos trabalhos realizados usam essa técnica para obter melhor custo/benefício sobre algoritmos de busca tradicionais, e os mais proeminentes costumam tratar do uso de algoritmos genéticos para ajustar parâmetros para IAs, para navegação de IAs de forma mais realista (levando em conta a elevação do terreno em adição a distância e obstáculos, como seria feito utilizando o algoritmo A*), entre outros exemplos e áreas de aplicação, como na área gráfica e de animação (WATSON et al., 2010); entretanto, houve dificuldade em encontrar trabalhos na literatura cujo objetivo fosse fornecer adaptabilidade dos agentes a estratégias avançadas desenvolvidas por jogadores humanos para prolongar sua experiência dentro do canal de fluxo. Essa lacuna, além de ajudar a direcionar o rumo desejado para o trabalho, gerou motivação para desbravar esse caminho ainda pouco percorrido.

1.2. OBJETIVOS

Neste trabalhos, pretendemos aplicar o paradigma de algoritmos genéticos na área de jogos para delinear a tomada de decisão dos oponentes dos jogadores, tornando-os menos previsíveis para jogadores experientes.

Para tal, este trabalho objetiva criar agentes que se adaptem continuamente às estratégias desenvolvidas por seus oponentes para serem posteriormente utilizados como adversários mais interessantes em jogos eletrônicos.

1.2.1. OBJETIVOS ESPECÍFICOS

- Estudar os conceitos teóricos de algoritmos genéticos e o estado da arte.
- Implementar um motor de algoritmos genéticos.
- Definir as regras para um cenário que possibilite analisar a adaptação dos agentes.
- Modelar e implementar os cromossomos dos agentes.
- Analisar as estratégias adotadas pelos agentes ao longo das simulações.

1.3. ORGANIZAÇÃO DO TEXTO

O trabalho está organizado em 5 capítulos:

- No capítulo 1, é apresentada uma breve introdução sobre o trabalho a ser desenvolvido, além da motivação, dos objetivos e limitações do trabalho
- No capítulo 2 é apresentada a base teórica necessária para a execução do trabalho proposto, incluindo: conceitos relativos ao funcionamento de algoritmos genéticos, desde a teoria de seleção natural das espécies e como esta é traduzida em um algoritmo capaz de evoluir uma população de agentes ao longo do tempo; o funcionamento de diagramas de decisão binária (n-BDDs) e como são aplicados como cromossomos de um algoritmo genético; e uma explicação superficial do canal de fluxo, um estado mental prazeroso observado em indivíduos engajados em atividades desafiadoras
- O capítulo 3 contém exemplos do estado da arte de algoritmos genéticos relacionados ao escopo deste trabalho, mostrando desde aplicações convencionais de algoritmos genéticos até aplicações em jogos e em esquemas de co-evolução; são evidenciadas também as interrelações entre os artigos apresentados neste capítulo e o trabalho a ser desenvolvido
- O capítulo 4 contém as especificações do trabalho a ser desenvolvido, desde as regras do sistema e detalhes sobre os agentes até a configuração do algoritmo genético que se planeja utilizar
- O capítulo 5 contém as considerações finais referentes ao trabalho efetuado

1.4. LIMITAÇÕES DA PESQUISA

Para este trabalho, não se objetiva analisar a eficiência de diferentes técnicas de computação evolutiva, excluindo também comparações de eficiência computacional entre uso de diferentes operações de *crossover*, mutação e seleção.

O escopo do trabalho não inclui ainda comparar os resultados do uso de computação evolutiva com outros métodos de geração de Inteligência Artificial.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são explanadas as teorias pertinentes ao desenvolvimento do trabalho. Apesar do estudo do “canal de fluxo” estar fora da área de atuação dos autores, é pertinente constar no trabalho uma visão superficial do seu funcionamento, visto que a motivação desta pesquisa é aumentar a flexibilidade da dificuldade dos jogos eletrônicos afim de fornecer maior tempo dentro do canal para os jogadores.

Também estão explicitados o funcionamento de algoritmos genéticos, técnica que será utilizada para auferir aos agentes a adaptabilidade almejada, e de diagramas de decisão binária, que é a estruturas de dados que pretende-se utilizar para modelar os cromossomos dos agentes.

2.1. ALGORITMOS GENÉTICOS

No ano de 1859, o naturalista britânico Charles Darwin publicou o livro "The Origin of Species" onde propôs que as formas de vida existentes no planeta não foram criadas por um deus e feitas imutáveis, mas sim que evoluíram a partir de outras formas de vida mais simples em um processo que Darwin denominou “seleção natural”. Naquela época, devido a falta de evidências concretas, a ideia foi tomada por um absurdo, mas com o passar do tempo mais e mais evidências passaram a corroborar com a teoria de Darwin, sendo que em 1870 a comunidade científica e grande parte do público em geral aceitavam a evolução como um fato (WIKIPEDIA, 2011).

Entretanto, ainda haviam vários questionamentos quando à validade da teoria da evolução derivados da dificuldade em descobrir os mecanismos que regiam a evolução dos indivíduos e a passagem das características positivas aos descendentes. Esses questionamentos só foram apaziguados com a chegada da chamada síntese evolutiva moderna, uma união entre várias ideias do campo de biologia envolvendo o conceito de evoluçao. O fator determinante para essa síntese foi o estabelecimento da teoria genética, cuja descoberta se iniciou nos experimentos do monge e cientista austro-húngaro Gregor Johann Mendel sobre a herança de traços hereditários em ervilhas, realizados entre 1856 e 1863. Os estudos realizados com base nas descobertas de Mendel levaram a descoberta dos mecanismos de funcionamento da evolução, que envolvem a passagem de informações dos pais para os filhos através de genes, a troca de informações entre esses genes (chamada *crossover*) e mutações aleatórias que aumentam a variabilidade dos indivíduos (WIKIPEDIA, 2011).

Baseados nos princípios da evolução genética, John Holland criou o conceito de algoritmos genéticos: ao se representar a solução de um problema na forma de uma cadeia de *bits*, gerar um conjunto contendo diversas possíveis soluções aleatórias (denominado população) e realizar repetidas operações genéticas e de seleção sobre essa população, espera-se que os indivíduos mostrem-se cada vez mais aptos a resolver o problema. Algoritmos genéticos são, portanto, um tipo de algoritmo de busca genérico, cujo diferencial é buscar ser mais eficiente do que um algoritmo de busca exaustivo (GOLDBERG, 2009).

2.1.1. DEFINIÇÃO

Algoritmos genéticos são algoritmos de busca baseados nos mecanismos de seleção natural e genética. Eles combinam a sobrevivência do mais forte entre estruturas de *strings* e troca de informações estruturadas, porém aleatórias, para formar um algoritmo de busca dotado de um pouco da tenacidade inovativa da busca humana (GOLDBERG, 2009).

2.1.2. ESQUEMA DE ALGORITMO GENÉTICO

Para utilizar algoritmos genéticos para buscar a solução de um problema é necessário modelar uma estrutura de dados que represente uma possível solução para o problema; a essa estrutura, se dá o nome de “cromossomo”. Cromossomos devem ainda poder sofrer as operações básicas de algoritmos genéticos, sendo elas: reprodução, *crossover* e mutação (GOLDBERG, 2009). Cada cromossomo representa um indivíduo, e sua modelagem será vista com mais detalhes em 2.1.3.

Um motor de algoritmo genético é responsável por executar as operações básicas descritas acima nas populações de indivíduos. Na primeira delas, reprodução, indivíduos são selecionados para contribuirem com um ou mais filhos para a próxima geração; essa seleção é feita aplicando uma função de avaliação (também chamada de função de *fitness*) ao cromossomo do indivíduo, sendo que essa função deverá retornar alguma medida do ganho, utilidade ou qualidade que se deseja maximizar (técnicas de implementação de funções de avaliação serão discutidas em 2.1.5). Essa operação visa emular a seleção natural de Darwin nos cromossomos (GOLDBERG, 2009). Existem várias maneiras diferentes de se implementar a função de reprodução; algumas serão detalhadas em 2.1.4.

Após definir quais indivíduos contribuirão para a próxima geração de indivíduos, são efetuadas as operações de *crossover*. Inicialmente, indivíduos selecionados na etapa de reprodução são pareados (os métodos de seleção dos pares são exemplificados em 2.1.6) e, posteriormente, acontece uma troca de informações genéticas entre eles. A maneira como isso ocorre depende da modelagem do cromossomo, e será discutido mais profundamente em 2.1.7.

Por último, os indivíduos são submetidos à operação de mutação. Nessa operação, uma pequena parte das informações do indivíduos são modificadas aleatoriamente. Segundo (GOLDBERG, 2009), reprodução e *crossover* são as principais operações dos algoritmos genéticos, sendo a operação de mutação secundária, porém extremamente necessária. Outros operadores genéticos e planos reprodutivos foram abstraídos do campo da biologia. Entretanto, os três examinados acima provaram ser tanto computacionalmente simples quando efetivos em resolver um número grande de problemas de otimização (GOLDBERG, 2009).

Após executar todas as operações, a população atual é descartada e os indivíduos gerados passam a ser a nova população. O algoritmo então irá avaliar se deve continuar gerando novos indivíduos ou se o melhor indivíduo encontrado até agora será dado como resposta do algoritmo. Essa avaliação é chamada de "condição de parada" do algoritmo genético, e comumente toma a forma de um número máximo de gerações de populações.

A Figura 1 mostra um diagrama do funcionamento de um motor de algoritmo genético simples.

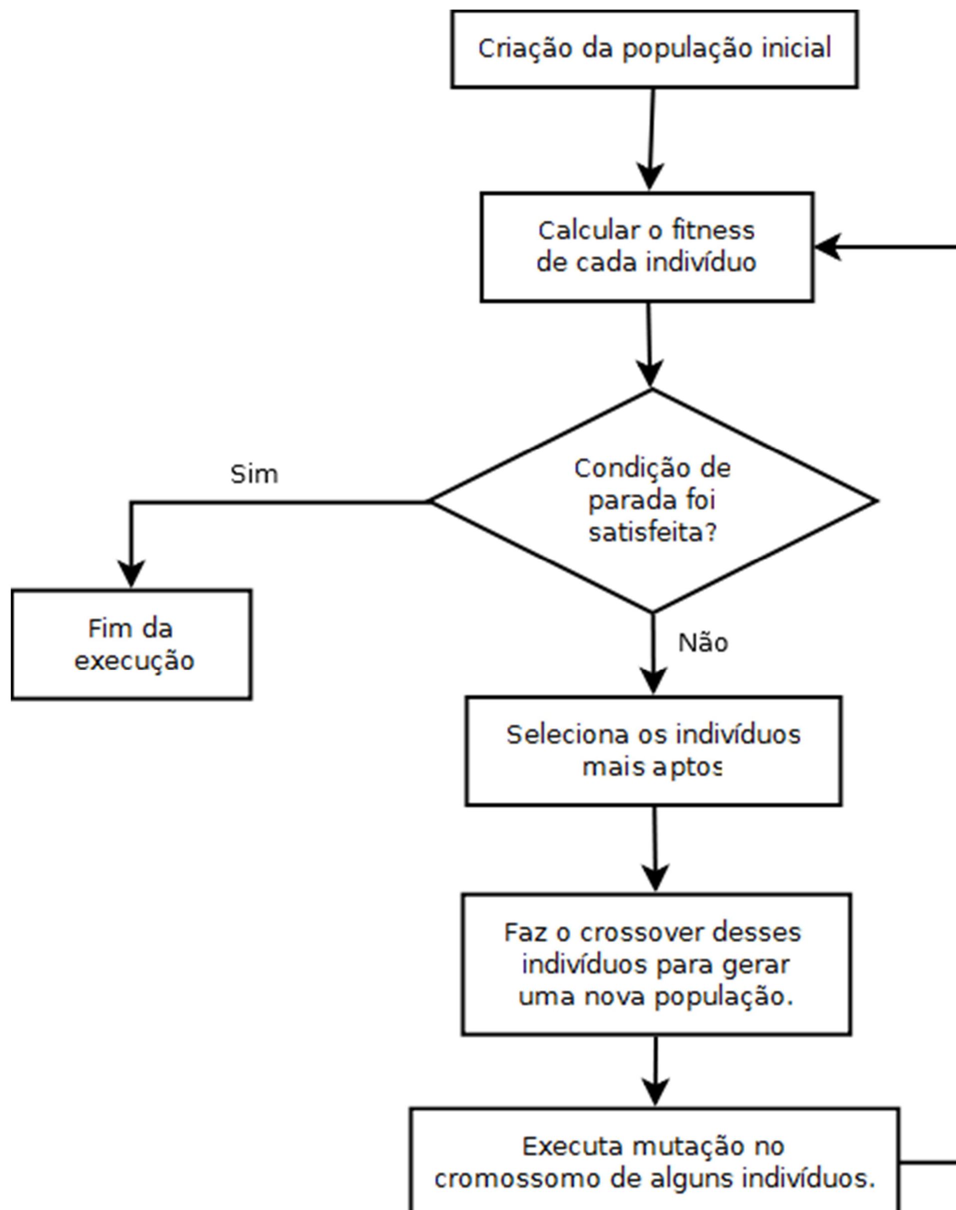

Figura 1. Diagrama do funcionamento de um motor de algoritmo genético

2.1.3. REPRESENTAÇÃO CROMOSSOMIAL

Todos os organismos vivos consistem de células, e cada célula contém o mesmo conjunto de um ou mais cromossomos; Cromossomos são cadeias de Ácido Desoxirribonucléico (DNA) que ditam como o organismo é. Um cromossomo pode ser dividido conceitualmente em genes, blocos funcionais de DNA que codificam uma proteína particular. Pode-se dizer que um gene codifica uma característica, como a cor dos olhos, e as diferentes possibilidades para uma característica são chamadas de alelos. Cada gene está localizado em um *locus*, uma posição específica, do cromossomo. (MITCHELL, 1998).

Ao trabalhar com algoritmos genéticos, o termo cromossomo se refere tipicamente a uma possível solução para um dado problema. Classicamente, cromossomos são definidos por *strings* de *bits* de tamanhos iguais (GOLDBERG, 2009), mas há exemplos de outras modelagens, como as utilizadas por (MORIWAKI et al., 1997). Na modelagem clássica, um gene é um bloco contendo um ou mais *bits* dentro da cadeia que codificam um elemento particular da solução (como, por exemplo, para o problema de otimização de uma função com vários parâmetros, os cinco primeiros *bits* poderiam representar o valor do primeiro parâmetro da função). Nesse tipo de modelagem, um *bit* é análogo a um alelo na genética.

Enquanto a implementação do motor de algoritmo genético é razoavelmente simples e bem definida, é na modelagem do cromossomo e na função de avaliação (vista em 2.1.5) que a maior parte do trabalho a ser desenvolvido está. São eles que irão definir primariamente o sucesso das soluções achadas pelo algoritmo genético. Um obstáculo comum ao codificar um problema na forma de um cromossomo é que, para problemas complexos o suficiente para requerir o uso de algoritmos genéticos, raramente se conhece *a priori* quais informações do problema serão relevantes o suficiente para entrarem na codificação do cromossomo e em quais *loci* eles serão melhor alocados (MITCHELL, 1998).

Além da tradicional cadeia de *bits*, (MITCHELL, 1998) exemplifica casos de uso de codificações utilizando cadeias cujos alfabetos são maiores (comparados aos alfabetos binários) e de codificações na forma de árvores, que foram por exemplo usadas por (KOZA, 1991) para modelar um cromossomo que representasse uma função matemática complexa.

Para ilustrar a modelagem de um cromossomo como uma cadeia de *bits*, podemos considerar como exemplo a busca por uma sequência de aminoácidos que se dobre em uma estrutura protéica desejada (MITCHEL, 1998). Considerando que existissem apenas 8 tipos de aminoácidos, e que a sequência contenha 20 deles, existiriam mais de um quinquilhão possíveis sequências diferentes de aminoácidos, o que certamente seria impraticável para uma busca exaustiva. Para modelar um cromossomo em forma de *string* de *bits*, seriam necessários 60 *bits*, onde cada 3 *bits* representariam um dos 8 tipos de aminoácido na cadeia. Considerando que 000 represente o aminoácido “A”, 001 o aminoácido “B”, 010 o aminoácido “C” e assim por diante, o cromossomo da cadeia de

aminoácidos [E, C, G, A, C, A, B, H, E, F, D, F, D, E, F, C, H, C, E, H] seria representada pela sequência de *bits* 100010110000010000001111100101011101011100101010111010100111, conforme a Tabela 1.

Tabela 1. Exemplo de cromossomo para uma cadeia de aminoácidos

E	C	G	A	C
100	010	110	000	010
A	B	H	E	F
000	001	111	100	101
D	F	D	E	F
011	101	011	100	101
C	H	C	E	H
010	111	010	100	111

A	000
B	001
C	010
D	011
E	100
F	101
G	111

2.1.4. DEFINIÇÃO DE *SCHEMAS*

Um conceito interessante que emerge da representação cromossomial são os *schemas*, definidos originalmente por John Holland como um modelo de similaridade descrevendo um subconjunto de *strings* com similaridades em certas posições da *string* (GOLDBERG, 2009). Exemplificando, em um cromossomo tradicional de tamanho 4, um *schema* possível é (0*110); Esse *schema* é definido por todos as combinações possíveis substituindo todos os "*" pelos valores do alfabeto (1 ou 0), portanto, {(00110), (01110)}.

Da noção de *schemas*, Holland formalizou a noção informal de "blocos de construção" (MITCHELL, 1998). Essa noção determina que, enquanto o algoritmo genético avalia explicitamente a aptidão das *strings* da população, ele está implicitamente avaliando a aptidão média de um número muito maior de *schemas*. Por exemplo, em uma geração aleatória, metade dos cromossomos serão parte do *schema* (1**...*) e metade será parte do schema (0**...*). Mesmo que os valores médios desses esquemas não estejam sendo explicitamente calculados e armazenados, a maneira como algoritmos genéticos funcionam permite que se diga que seu comportamento é o mesmo que seria esperado caso ele estivesse calculando essas médias (MITCHELL, 1998).

As operações de *crossover* e mutação podem criar e destruir instâncias de *schemas*; essas consequências do uso dessas operações serão exploradas mais a fundo em suas respectivas seções (2.1.7 e 2.1.8).

2.1.5. OPERAÇÃO DE SELEÇÃO (REPRODUÇÃO)

A operação de seleção (MITCHELL, 1998), também chamada de operação de reprodução (GOLDBERG, 2009), define quais indivíduos serão utilizados para gerar os indivíduos da próxima geração. Seu objetivo é enfatizar a sobrevivência das características dos indivíduos mais aptos nas futuras gerações na esperança de que sua prole tenha uma aptidão ainda maior.

A operação de seleção é feita com base no valor retornado pela função de avaliação do indivíduo (que será vista em 2.1.5) e em um método pré-definido de seleção. É possível encontrar vários métodos diferentes na literatura sobre algoritmos genéticos (MITCHELL, 1998), todos com vantagens e desvantagens próprias. Alguns exemplos são:

Seleção proporcional à aptidão: Utilizado por John Holland quando da confecção original dos algoritmos genéticos (MITCHELL, 1998), nesse método um indivíduo tem uma chance de ser selecionado para cada reprodução proporcional ao valor de sua aptidão dividido pela soma das aptidões de toda a população. Uma implementação deste método é a chamada "Roleta": aos indivíduos é alocada uma fatia em uma roda proporcional à sua aptidão, e essa roda é girada n vezes para selecionar os n indivíduos necessários. Apesar de se basear na premissa de que o número de vezes que um indivíduo será selecionado para reprodução seja proporcional à sua aptidão, em função das populações em algoritmos genéticos serem relativamente pequenas, muitas vezes esse número de vezes acaba sendo muito distante do esperado (MITCHELL, 1998).

***Stochastic Universal Sampling*:** Proposto em 1987 por James Baker (MITCHELL, 1998), esse método busca minimizar o efeito negativo observado no uso do método da Roleta. Ao invés de girar a roleta n vezes para encontrar n indivíduos, são colocados sobre ela n pontos fixos igualmente espaçados radialmente. A roleta é rolada então apenas uma vez, e todos os indivíduos sob os pontos fixos são selecionados.

Esses dois métodos apresentam um grande problema comum: Em função da alta variação entre os indivíduos mais aptos e os menos aptos durante as populações iniciais, os mais aptos e seus descendentes tendem a rapidamente dominar a população, prevenindo o algoritmo genético de explorar mais a fundo outras possibilidades (MITCHELL, 1998). Esse problema é conhecido como "convergência prematura".

Escalonamento *Sigma*: Esse método busca manter a pressão de seleção (o grau a que indivíduos altamente aptos são selecionados para reprodução) constante ao longo da execução do algoritmo genético. Nesse método, o número de vezes esperado que um indivíduo seja selecionado é uma função direta de sua aptidão e inversa da média da população e do desvio-padrão da população. Isso garante que no início, quando o desvio-padrão é maior, os indivíduos menos aptos terão chances de continuar por tempo suficiente na população até que o desvio-padrão diminua, quando será permitido que a pressão de seleção cresça (MITCHELL, 1998).

Elitismo: É uma adição para vários métodos de seleção que força a permanência de um número fixo dos elementos mais aptos da população. Alguns pesquisadores afirmam que o uso de elitismo melhora significativamente a performance da busca (MITCHELL, 1998).

Seleção por *Ranking*: Nesse método, o conceito de aptidão absoluta é descartado e é usado ao invés disso a aptidão relativa dos indivíduos. Estes são ordenados de 1 até n (o tamanho da população) de acordo com sua aptidão e lhes é conferido de acordo com essa ordem um número esperado de vezes para serem selecionados, que evolue linearmente do melhor até o pior colocado. Como a pressão de seleção se mantém constante, esse método pode se mostrar lento para encontrar indivíduos altamente aptos (MITCHELL, 1998).

Seleção por Torneio: Em todos os métodos descritos acima, são necessárias pelo menos duas iterações por toda a população (uma para computar a média das aptidões e uma para computar o número de seleções esperado). Além disso, na seleção por *ranking*, é necessário ordenar os indivíduos. Essas operações podem ser custosas demais, e diminuir a eficiência do algoritmo. No método de seleção por torneio, a pressão de seleção é semelhante à da seleção por *ranking* com a vantagem de ser computacionalmente mais eficiente e mais passível de paralelismo (MITCHELL, 1998). Nela, dois indivíduos aleatórios são escolhidos da população. Dado um número aleatório k contido entre 0 e 1, caso esse número seja maior que um valor arbitrário (tal qual 0,75) é escolhido o indivíduo de maior aptidão; caso contrário, o de menor aptidão. Ambos podem ser escolhidos novamente no futuro.

Seleções em *Steady-State*: A maior parte dos algoritmos genéticos na literatura são orientados a gerações; em cada geração a população consiste de indivíduos novos (apesar de poder haver alguns que sejam iguais a um de seus ancestrais). Nesse paradigma de seleção, apenas uma pequena parte dos indivíduos são trocados a cada

geração. Entre outras aplicações, esse método se mostrou efetivo quando há a necessidade de aprendizado incremental e onde membros da população devem agir coletivamente (MITCHELL, 1998).

2.1.6. FUNÇÃO DE AVALIAÇÃO (OU APTIDÃO)

Conforme visto em 2.1.3, cada cromossomo representa uma possível solução para o problema especificado. Para definir quais indivíduos serão utilizados para criar a próxima geração, é necessário atribuir uma nota a cada um de alguma maneira. A função que atribui essa nota é chamada função de avaliação, ou de aptidão.

A aptidão de um cromossomo deve refletir o mais fielmente possível a qualidade da solução apresentada pelo cromossomo. Em certos casos, é impraticável aplicar a solução representada para todos os indivíduos, devendo portanto a função de avaliação realizar uma aproximação. Consideremos o exemplo utilizado em 2.1.3, onde se deseja encontrar uma sequência de aminoácidos que se dobre em uma estrutura protéica desejada. Um método possível para definir a aptidão de uma dada solução seria medir a energia potencial da sequência de aminoácidos representada em relação à estrutura desejada: quanto menor esse valor, maior a aptidão da solução (MITCHELL, 1998). Entretanto, seria impraticável (senão impossível) testar fisicamente todas as sequências em laboratório e medir a resistência demonstrada. Contudo, dada uma sequência e a estrutura desejada, é possível através da aplicação de conhecimentos sobre biofísica estimar computacionalmente essa energia potencial calculando algumas das forças agindo sobre cada aminoácido.

2.1.7. OPERAÇAO DE *CROSSOVER*

Uma operação de *crossover* visa misturar características de dois indivíduos para gerar dois novos indivíduos ao mimetizar as operações de recombinação biológica entre dois organismos. Existem algumas maneiras de realizar *crossover* (ponto único, ponto duplo, parametrizada uniforme, etc.), sendo sua execução condicionada à modelagem cromossomial.

Na modelagem clássica, utilizando *strings* de *bits*, uma operação de *crossover* em ponto simples consiste em definir um inteiro k cujo valor seja maior que 0 e menor ou igual ao tamanho da *string*. Duas novas *strings* são então criadas pela troca de todos os caracteres entre a posição $k+1$ e o tamanho da *string* entre os dois cromossomos pais. Esse processo é exemplificado na Tabela 2.

Tabela 2. Exemplo de *crossover* em ponto único entre duas *strings* de *bits*

Antes do *crossover*	Valor de *k*	Após *crossover*
1001100 1110001	1	1110001 1001100
1001100 1110001	2	1010001 1101100
1001100 1110001	3	1000001 1111100
1001100 1110001	4	1001001 1110100
1001100 1110001	5	1001101 1110000
1001100 1110001	6	1001101 1110000

A aplicação de *crossover* está fortemente ligada à noção de *schemas*. Considerando dois blocos (**110) e (01***) que, para dado problema, sejam superiores a todos os outros, uma operação de *crossover* entre os relativamente fracos indivíduos (10110) e (01001) no ponto 3 geraria dois indivíduos, um deles com ótima aptidão, (01110). Essa recombinação de blocos é o objetivo principal da aplicação deste operador (MITCHELL, 1998).

Contudo, conforme dito na seção 2.1.4, operações de *crossover* podem criar ou destruir instâncias de *schemas*. Por exemplo, se um indivíduo (101), pertencente ao *schema* (1*1), sofre *crossover* em ponto único com o indivíduo (010) na posição 1, os novos indivíduos serão (110) e (001), nenhum dos dois pertencentes ao *schema* supra-citado. Entretando, novos *schemas* foram criados nessa operação, e os benefícios e prejuízos para a aptidão média da população dependerão do problema ao qual está sendo aplicado o algoritmo genético.

As diferentes técnicas de *crossover*, assim como os métodos de seleção, tem vantagens e desvantagens próprias. Utilizar *crossover* em ponto único impossibilita, por exemplo, a combinação de *schemas* como (11*****1) e (****11**) para formar o *schema* (11**11*1). Da mesma maneira, *schemas* longos tem alta probabilidade de serem quebrados durante um *crossover* em ponto único. Ainda que usar *crossover* de ponto único seja válido para *schemas* curtos e de baixa ordem, normalmente não é conhecido *a priori* quais combinações e ordens de *bits* vão agrupar funcionalidades relacionadas (MITCHELL, 1998). Além disso, esse método fomenta a propagação de "caroneiros", *bits* que não fazem parte de um *schema* mas que, em função de seu posicionamento, se propagam junto a um *schema* positivo, e vários autores notaram

também que certos *loci* recebem preferência com o uso desse método (MITCHELL, 1998).

Para minimizar esses problemas, costuma-se utilizar também *crossover* em ponto duplo, onde além de ser definida a posição inicial da cadeia de *bits* a ser trocada entre os indivíduos é definida a posição final. Esse método reduz o favorecimento de *loci* e aumenta a gama de *schemas* que podem ser trocados, ainda que não englobe todas as possibilidades (MITCHELL, 1998). Métodos utilizando mais pontos foram experimentados, e há desenvolvedores que acreditam que o uso do método chamado de parametrização uniforme, onde cada *bit* tem uma chance (normalmente entre 0.5 e 0.8) de ser trocado independentemente é superior (MITCHELL, 1998). Apesar de esse método poder realizar recombinações entre quaisquer *schemas*, a chance de que ele quebre *schemas* é muito maior, e a falta de agregação atribuída aos alelos podem inibir a formação de alelos coadaptados.

Assim como na escolha do método de seleção, não há uma resposta simples para a escolha do método de *crossover*; o sucesso ou falha de um método particular depende de detalhes próprios de cada problema.

2.1.8. OPERAÇÃO DE MUTAÇÃO

Há na comunidade de algoritmos genéticos uma visão de que a operação de crossover é o instrumento principal de variação e inovação nos algoritmos genéticos, com a mutação evitando uma fixação permanente da população em qualquer *locus* particular, que difere da posição de outros métodos computacionais evolucionários que utilizam mutações como a única fonte de variação (MITCHELL, 1998). Ainda segundo (MITCHELL, 1998), apesar de os estudos comparativos entre *crossover* e mutação serem inconclusivos, a verdadeira questão não é a escolha entre um e outro mas sim o balanço entre o uso dos dois.

(GOLDBERG, 2009) diz que a frequência de mutação para obter resultados satisfatórios deve ser baixa: para modelagem de cromossomos em *strings,* é recomendado que seja da ordem de uma mutação por mil *bits*. Ainda nessa modelagem, executar o operador de mutação em um indivíduo implica simplesmente em escolher um dos *bits* de sua cadeia de maneira aleatória e inverter seu valor. Para diferentes modelos de cromossomos, técnicas específicas devem ser aplicadas para promover a aleatoriedade das alterações.

2.2. BINARY DECISION DIAGRAM (BDD)

Um Diagrama de Decisão Binário (BDD) é um diagrama utilizado para representar uma função *booleana*, sendo que a partir de cada nó, para uma determinada entrada, existe um e somente um caminho para um valor de saída de 0 ou 1, e os nós podem ser ativados somente uma vez (AKERS, 1978). Esse diagrama pode ser utilizado como estrutura de dados para representar uma função *booleana* em um algoritmo.

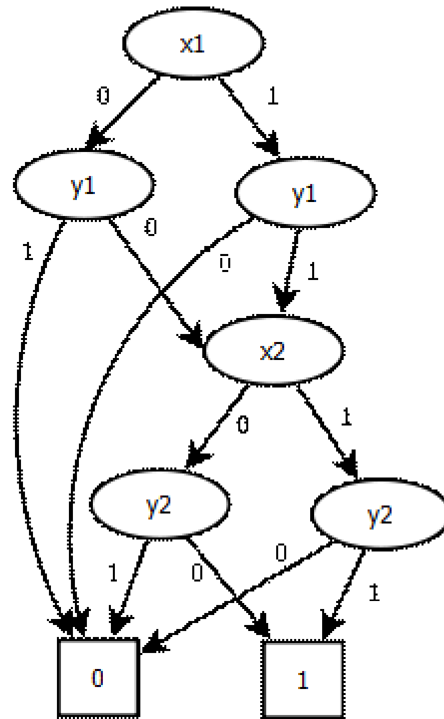

Figura 2. BDD para a função (x1 – y1) ^ (x2 – y2) (AKERS, 1978)

Um *n-BDD* difere de um *BDD* na quantidade de possíveis valores de saída. Um *BDD* tradicional, como o da Figura 2, possui apenas 2 valores de saída, 0 ou 1. Já em um *n-BDD* é possível ter *n* valores de saída, conforme mostrado na Figura 3. Por isso, um *n-BDD* não está restrito somente a funções lógicas. Pode-se utilizar *n-BDD*s em diversas áreas, possuindo eles portanto uma ampla aplicabilidade (MORIWAKI et al., 1997).

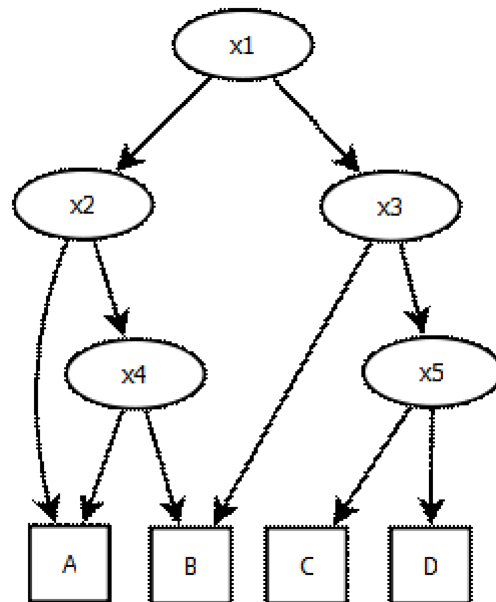

Figura 3. Exemplo de *n-BDD* (AKERS, 1978)

2.3. CANAL DE FLUXO

O que o cérebro humano foca em um dado momento é determinado através de uma combinação de desejos inconscientes e da vontade consciente. Quando se cria jogos, o objetivo é criar uma experiência interessante o suficiente para que prenda o foco do jogador por mais tempo, e o mais intensamente possível.

Quando algo captura a completa atenção e imaginação de uma pessoa por um longo período ela entra em um estado mental interessante e prazeroso, chamado por alguns especialistas de "canal de fluxo" e definido por estes como "um sentimento de foco completo em uma atividade, que provê um grande nível de entretenimento e satisfação" (SCHELL, 2008). Há algumas prerrogativas chave para criar uma atividade que coloque um ser humano no estado de fluxo. São elas:

- **Objetivos claros:** Quanto mais claro o objetivo, mais fácil se manter focado na tarefa em mãos
- **Sem distrações:** Distrações quebram o foco na tarefa, e sem foco, não há fluxo
- ***Feedback* direto:** Respostas imediatas ajudam a manter o foco
- **Desafio constante:** Desafios exigem concentração

A parte mais problemática é manter o nível do desafio condizente com a habilidade do jogador. Se uma pessoa começa a pensar que não pode vencer um desafio, ela se sentirá frustrada e sua mente começará a buscar uma atividade que ofereça recompensas

a menor custo. Por outro lado, se o desafio é muito fácil, ela se sentirá entediada, e buscará então atividades com maior ganho.

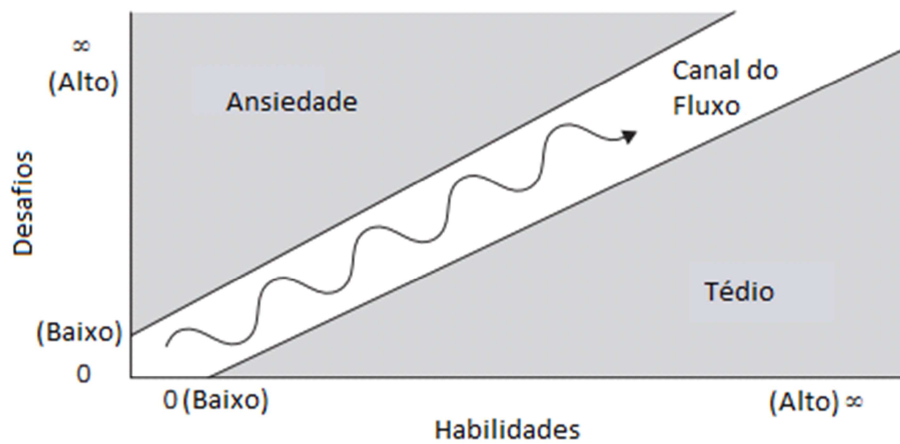

Figura 4. Canal de fluxo (SCHELL, 2008, p. 122)

Por isso, projetistas de jogos visam calibrar a dificuldade dos desafios para manter o jogador dentro do canal de fluxo pelo máximo de tempo possível; conforme o jogador progride no jogo, espera-se que suas habilidades aumentem, logo a dificuldade deve ser ajustada para acompanhar o jogador.

3. TRABALHOS RELACIONADOS

Neste capítulo serão mostradas algumas aplicações de algoritmos genéticos relacionadas a este trabalho. Estas aplicações incluem aplicações na área de jogos, co-evolução de agentes, otimização de performance de algoritmos genéticos e comparação de estruturas de cromossomos.

3.1. ALGORITMOS GENÉTICOS EM ESTRATÉGIAS DE GRUPO

No trabalho realizado por (FERNÁNDEZ et al., 2008), o objetivo dos autores foi demonstrar o uso de algoritmos evolucionários em jogos eletrônicos e seu uso em jogos comerciais, não só como uma forma de manter a atenção do jogador por mais tempo mas também proporcionando maior entretenimento ao fornecer estratégias menos previsíveis. Para executar as simulações, foi utilizado o *framework* Robocup (ROBOCUP, 2011), que trata toda a simulação de uma partida de futebol virtual. A meta do trabalho é gerar controladores para governar o comportamento dos 11 jogadores do time, sendo que o *framework* fornece informações aos agentes em tempo real via três sensores: *aural*, *visual* e *corporal*.

Um agente deve possuir um conjunto de regras para controlar suas reações às mudanças em seus sensores. A estratégia do time é, portanto, a soma das estratégias de cada agente. Entretanto, definir estratégias específicas para cada agente seria custoso e complexo, além de requerer alto conhecimento do comportamento necessário de cada jogador. Para simplificar, todos os agentes (exceto o goleiro, que foi implementado manualmente) são gerenciados pelo mesmo controlador. Isso não significa que todos os jogadores executam a mesma ação, já que seus sensores terão valores individuais.

Para reduzir o grau de complexidade, foram selecionados quatro parâmetros oriundos dos sensores: Estado de vantagem (0 se há mais jogadores do próprio time do que do adversário por perto ou 1 caso contrário), possibilidade de chutar a bola (1 se puder, 0 se não), posição do agente (0 se estiver perto do seu gol, 1 se estiver perto do gol adversário e 3 para não definido), posse de bola (0 se a bola estiver consigo, 1 se estiver com um jogador de seu time, 2 para um adversário e 3 se não souber). Assim, um indivíduo possui 48 genes, representando todas as combinações possíveis desses parâmetros. Foram também realizados testes utilizado um quinto parâmetro, mas devido ao aumento de complexidade este acabou por ser descartado.

Cada um desses genes aponta para uma entre as nove ações definidas: Voltar (para sua posição inicial), procurar a bola, interceptar a bola, virar 15º, chutar (em direção ao gol adversário), passar para alguém próximo, passar para alguém distante, chutar para longe (da sua área), levar a bola (em direção ao gol adversário). Isso gera um espaço de busca de $9^{48} = 3^{96}$, visivelmente imprático para um algoritmo de busca convencional.

Para ser calculada a aptidão (*fitness*) da estratégia utilizada pelo time, ou seja, o genoma dos seus indivíduos, são necessários dados pós simulação fornecidos pelo *framework*. Cinco dados foram utilizados, para reduzir a necessidade de poder computacional: Distância da bola (distância média entre jogadores e bola), posse de bola (tempo médio que a bola passou no campo adversário), bola na área (tempo médio que a bola passou na área adversária), gols pró e gols contra. A função de aptidão foi dada por $fitness(pop[i]) = f1(c, p, a) + f2(s, r)$, onde:

- c é o valor de "distância da bola";
- p é o valor de "posse de bola";
- a é o valor de "bola na área";
- $f1$ é igual à (0,01 * c + 0,1 * p + 1 * a)
- s é o valor de "gols pró";
- r é o valor de "gols contra";
- $f2$ é igual à 0, se s for 0; (100 * (s - r + 1)), se $s \geq r$; e (99 / (s - r)) caso contrário.

A função de avaliação é, portanto, dividida em uma avaliação dos fundamentos do futebol (visto que, a priori, os agentes não possuem nenhum conhecimento sobre como jogar futebol) e uma avaliação do resultado da partida.

O algoritmo utilizado empregou *crossover* em um ponto único, torneio binário para seleção de pais e elitismo. Mutações são efetuadas alterando a ação de um gene por outra aleatóriamente. Em todas as simulações, foram utilizadas populações de 30 indivíduos e probabilidade de *crossover* igual à 1.0. Vários valores de mutação, número de gerações e de filhos foram utilizados, e as populações foram sempre inicializadas com valores aleatórios. Inicialmente, as populações foram testadas contra um oponente implementado manualmente; posteriormente, foram usados como oponente as melhores estratégias desenvolvidas neste primeiro estágio.

Os primeiros testes obtiveram sucesso: Como esperado, inicialmente, os agentes não sabem jogar, e portanto obtém péssimos resultados; entretanto, em poucas gerações eles começaram a evoluir e obter resultados satisfatórios. Ao tentar usar os melhores candidatos como oponente, a complexidade do algoritmo impediu que mais de dois oponentes fossem utilizados. Ainda assim, foi concluído que, para os objetivos traçados (gerar comportamento adaptativo e de baixa previsibilidade contra jogadores humanos), o experimento obteve sucesso.

3.2. SIMULAÇÃO DE ECOSSISTEMAS USANDO CO-EVOLUÇÃO

Os autores (MORIWAKI et al., 1997) buscaram demonstrar a eficácia do uso de genes expressos por n-BDD (*Binary Decision Diagram*) em algoritmos genéticos evoluindo paralelamente duas populações antagonistas de indivíduos. Para tal, é criado um modelo de cadeia alimentar povoado por indivíduos de três tipos: carnívoros, herbívoros (ambos ativos) e plantas (passivas).

Um n-BDD é composto por vários nós de decisão, todos com um ponto de entrada e dois de saída, e alguns nós de saída, que possuem zero ou mais pontos de entrada. Para analisar a eficiência do uso de n-BDD, foram também realizados testes usando Autômatos de Estado Finito e Sistemas Classificadores.

Todos os tipos de genes modelados usam como entrada uma cadeia de *bits*, que contém informações sobre o estado do animal e sua percepção visual. Essa cadeia é composta por:

- Bit 1: Indica se o animal está com fome.
- Bit 2: Indica se o animal está de estômago cheio.
- Bit 3: Indica se há um carnívoro longe.
- Bit 4: Indica se há um carnívoro perto.
- Bit 5: Indica se há um herbívoro longe.
- Bit 6: Indica se há um herbívoro perto.
- Bit 7: Indica se há uma planta longe.
- Bit 8: Indica se há uma planta perto.

Dada uma entrada composta por estes 8 bits, os genes fornecem uma das seguintes ações como saída:

- Andar (em uma direção aleatória);
- Comer (andar na direção da comida mais próxima e comê-la, se alcançada, ou Andar se não houver comida no campo de visão);
- Ficar Parado;
- Aproximar-se (do animal mais próximo do mesmo tipo que o seu, ou Andar se não houver nenhum no campo de visão).

Os herbívoros possuem, ainda, uma quinta ação:

- Fugir (mover-se na direção oposta ao carnívoro mais próximo, ou Andar se não houver carnívoros no campo de visão);

Para testar a modelagem, foi criado um simulador com dois modos: o modelo pequeno, onde um carnívoro de genoma fixo e um herbívoro passível de evolução são simulados em um campo de tamanho 20x20; e o modelo grande, que busca simular um modelo de cadeia alimentar e visualizar o fenômeno da co-evolução.

No modelo pequeno, várias plantas são inicialmente criadas, e uma nova planta sempre nasce quando outra é comida por herbívoros. A simulação acaba quando o herbívoro morre de fome ou é comido, e a aptidão é igual ao tempo que o herbívoro sobreviveu. Cada geração é composta de 30 indivíduos, sendo que os 5 melhores são inclusos na próxima geração, os 5 piores são descartados e outros 25 são criados através de operações genéticas entre os demais 20 e os 5 melhores. Foi observada uma evolução mais rápida ao utilizar n-BDD, seguido pelo uso de Sistemas Classificadores e por último Autômatos de Estado Finito.

No modelo grande, são criados vários carnívoros e herbívoros, todos passíveis de evolução, e a simulação se dá em um espaço de tamanho 150x150. Ao comer, um animal adquire energia; quando esta energia atinge um valor determinado, o animal cria um filho e sua energia é dividida igualmente entre os dois (no artigo, não é especificado como os autores definiam o gene dos filhos ou como e quando eram efetuadas as operações genéticas sobre as populações no modelo grande). A energia cai a cada passo da simulação até atingir zero, quando o animal morre de fome e plantas nascem ao redor de seu corpo. Após efetuar a simulação, observaram-se picos intercalados de população de carnívoros e herbívoros, caracterizando uma cadeia alimentar.

Foi concluído que, para o caso proposto, o uso de n-BDD mostrou-se mais eficiente do que os outros dois modelos de gene testados.

3.3. OTIMIZAÇÃO DE ESTRATÉGIAS EM JOGOS

O objetivo de (WATSON et al., 2010) foi utilizar algoritmos genéticos para otimizar o desenvolvimento de cidades no jogo FreeCiv, um jogo *open-source* de estratégia em turnos, baseado no famoso Sid Meier's Civilization. No FreeCiv, o jogador deve escolher uma civilização e evoluí-la, criando novas cidades, gerenciando recursos, fazendo as cidades crescerem, afim de se tornar a maior civilização de todas. Existem três maneiras de alcançar esse objetivo: Erradicar todas as outras civilizações no mapa; alcançar a era espacial e ser a primeira civilização a alcançar Alpha Centauri; ou ter a maior pontuação em um limite configurado.

Com relação a implementação do algorítimo genético nesse projeto, foram consideradas duas opções. A primeira seria uma implementação *offline*, onde várias partidas de FreeCiv seriam simuladas para treinar o algoritmo genético até ser gerada uma hipótese otimizada para gerenciar uma civilização. Entretanto, esta abordagem requeriria a simulação de praticamente todas as possibilidades de cenários que podem se desenvolver ao longo de uma partida, o que se mostrou imprático. A segunda opção foi de uma implementação *online*, que seria executada paralelamente enquanto um jogador humano jogasse uma partida. O algoritmo genético seria utilizado então para desenvolver uma estratégia de desenvolvimento para o estado atual de cada turno do jogo. Apesar de requerer uma quantidade menor de simulações, esta abordagem traz o ponto negativo de forçar um jogador humano a aguardar o processamento do algoritmo genético durante sua partida. Mesmo assim, foi escolhida a alternativa *online*.

O *framework* de algoritmo genético possui três níveis: no mais baixo, estão os genes: vetores de inteiros, cada um representando um fator relevante ao desenvolvimento de uma cidade. Intuitivamente, uma cidade (segundo nível) consiste de uma coleção de genes. O último nível, o genoma, é composto de várias cidades.

A estrutura de genes tem três atributos: identificador, comprimento e alcance. O identificador indica qual o tipo de informação que o gene representa ("felicidade" ou "comida", por exemplo); O comprimento indica qual o tamanho do vetor daquele gene, e o alcance, os valores possíveis. A estrutura das cidades contem um identificador e um vetor de genes. O genoma foi composto por um vetor de todas as cidades do jogador e

seu tamanho, e um vetor contendo todos os genes de todas as cidades. O genoma é, portanto, a estrutura usada para representar hipóteses para lidar com o desenvolvimento da civilização. Também nele é que serão executadas as operações de *crossover* e mutação.

3.4. CO-EVOLUÇÃO DE AGENTES ANTAGONISTAS

(KOZA, 1991) cita vários exemplos de aplicações práticas onde a aplicação de algoritmos genéticos obteve sucesso. Entre elas, são notáveis a emergência de planejamento, reconhecimento de padrões, descobertas empíricas (como a redescoberta da Terceira Lei de Kepler), entre outras aplicações.

Constam no artigo três aplicações práticas com resultados detalhados; Na primeira, uma 'formiga artificial' deve encontrar uma trilha de 89 pedras contidas em uma grade de 32x32. No segundo exemplo, dois agentes são posicionados em um plano, e um deles deve alcançar o outro. No terceiro, é criado um pseudo-jogo de soma zero, onde dois jogadores assimétricos buscam obter o maior ganho possível.

Enquanto a primeira aplicação é um exemplo comum de aplicação de algoritmos genéticos, as duas seguintes tratam da co-evolução de agentes; Ou seja, enquanto a 'formiga' evolue em um cenário estático, os agentes co-evolucionário deverão frequentemente buscar se readaptar às novas estratégias de seus oponentes. Por esse motivo, este resumo conterá apenas detalhes sobre os dois últimos exemplos.

Utilizar co-evolução força os indivíduos a melhorar constantemente: citando um exemplo oriundo da natureza, em um ambiente onde haja uma planta predada por insetos, essa planta poderia evoluir uma camada externa dura para evitar ser comida. Em contrapartida, os insetos poderiam então evoluir uma mandíbula mais potente, que pudesse quebrar a camada. A planta, novamente ameaçada, poderia então desenvolver uma toxina para afugentar os insetos, que poderiam posteriormente desenvolver enzimas digestivas que anulassem os efeitos do veneno (KOZA, 1991).

3.4.1. JOGO DE CAÇA

Este jogo é caracterizado por dois oponentes que se movem simultaneamente: O jogador perseguidor P deve capturar o jogador escapante E, sendo que P se movimenta mais rapidamente do que E. A cada turno, os jogadores devem escolher um ângulo para se moverem; P possui velocidade 1, enquanto E possui velocidade 0.67. Os jogadores

escolhem simultaneamente o ângulo no qual vão se mover e sabem as coordenadas do oponente no plano. Para fins de simplificação do algoritmo, a posição de P será sempre considerada a origem (0,0), e a posição de E será a posição relativa à P.

Para desenvolver as estratégias, um conjunto de posições iniciais de tamanho 10 foi criado, onde os valores iniciais de X e Y de E estão contidos entre -5.0 e 5.0. Foi considerado que P capturara E quando a distância entre os dois for menor do que 0.5. O ganho dos jogadores é medido em tempo para a captura: Enquanto P deve minimizar o tempo de captura, E deve sobreviver o maior tempo possível, sendo que após 100 turnos considera-se que P não conseguiu capturar E, sendo 100 portanto o maior ganho possível para E. As coordenadas que determinam a posição de cada agente é sempre conhecida por ambos, e cada um deve fornecer o ângulo no qual desejam se deslocar.

Neste exemplo, a melhor estratégia para P é seguir sempre uma linha reta na direção em que E esteja, enquanto para E a melhor estratégia é correr para longe de P sobre a mesma linha. Os genes dos agentes são compostos pelas operações matemáticas de adição, subtração, multiplicação, divisão (sendo que divisões por 0 retornam 0) e exponenciação. Segundo Koza, o paradigma de algoritmos genéticos é apropriado para esse tipo de problema já que a solução tem a forma de uma expressão matemática cujo tamanho e forma podem não ser previamente conhecidos.

Tipicamente, após cada geração, os perseguidores melhoram seu desempenho. Após algumas gerações, os melhores conseguem capturar o escapante em uma fração (2, 3 ou 4) das posições iniciais. Em um período de tempo ainda menor, passam a ser capazes de alcançá-lo em uma fração maior (4, 5, ou 6) das 10 posições iniciais. É comum que esses perseguidores sejam eficazes apenas em alguma fração do plano ou valor de distância.

Na 17a. geração, um indíviduo que capturava o escapante em todos os 10 casos iniciais emergiu. A solução deste indivíduo foi (% (- (% (+ (* 2.0 Y) -0.066) -0.365) (% Y -0.124)) (+ (EXP X) Y -0.579)), a qual, simplificada, é uma aproximação excelente da função arco tangente, e, testada em um conjunto maior de posições iniciais (1000 amostras) obteve 100% de sucesso. Após a evolução de um perseguidor otimizado, um escapante otimizado evoluiu similarmente.

3.4.2. CO-EVOLUÇÃO HIERÁRQUICA

O objetivo deste experimento é achar estratégias de 'minimax' para dois jogadores simultaneamente em um simples jogo discreto de soma zero. O jogo consiste de uma árvore arbitrária (descrita na figura 2 do artigo), onde o primeiro jogador (denominado 'X') deve fazer três movimentos e o segundo (chamado 'O') tem apenas dois, sendo estes movimentos intercalados começando por X. Em cada movimento, o jogador deve fazer uma escolha binária ('L' ou 'R'), e ao final das escolhas O entrega à X um ganho arbitrário especificado pela árvore. O jogo é de informação completa, ou seja, todos os jogadores tem acesso completo aos movimentos anteriores.

Em uma aplicação comum de algoritmo genético, cada população seria testada contra um algoritmo minimax específica para o jogo em questão (uma medida de aptidão absoluta); Neste experimento, entretando, a ideia é utilizar as populações de X como 'ambiente' para as populações de 'O', e vice-versa (aptidão relativa), onde as duas populações irão evoluir simultaneamente, precisando se adaptar conforme seus adversários melhoram suas estratégias. A aptidão relativa é calculado como a média dos ganhos obtidos pelo indivíduo ao jogar contra toda a população oponente. Para o jogo-exemplo usado no artigo, se X e O fossem algoritmos minimax, o ganho de X seria 12, sendo este portanto o valor do jogo.

Foram geradas duas populações aleatórias de 300 indivíduos; Os melhores valores de aptidão observados foram de 10,08 para X e 7,57 para O, sendo que nenhum dos dois possui aptidão máxima absoluta. Na geração 1, o melhor jogador X obteve aptidão relativa de 11.28, mas ainda não atingiu aptidão absoluta; Entretanto, o melhor indivíduo de O, com aptidão relativa de 7.18, caracterizou uma estratégia *minimax*. É relevante lembrar que o algoritmo não possui conhecimento de que tal estratégia possua valor máximo de aptidão absoluta.

Nas gerações entre 2 e 14, o número de indivíduos de O que chegaram à estratégia de aptidão máxima absoluta foi, respectivamente, 2, 7, 17, 28, 35, 40, 50, 64, 73, 83, 93, 98 e 107. Ou seja, a estratégia minimax passou a dominar a população O.

Na geração 14, um indivíduo da população X atingiu a máxima aptidão absoluta, obtendo aptidão relativa de 18,11. Entre as gerações 15 e 29, o número de indivíduos atingindo aptidão absoluta máximo foi de, respectivamente, 3, 4, 8, 11, 10, 9, 13, 21, 24, 29, 43, 32, 52, 48 e 50. Na população 38, o número era de 74 (a população O tinha 188

indivíduos com aptidão absoluta então); assim como aconteceu com a população O, a estratégia minimax passou a dominar a população X. É interessante notar que, como as populações ainda não eram totalmente compostas por algoritmos de aptidão absoluta, a aptidão relativa destes indivíduos era 19,11 para X e 10,47 para O.

Resumindo, o resultado do experimento foi que as duas populações chegaram a uma estratégia otimizada sem que fosse necessário nenhum conhecimento do jogo *a priori* para nenhum dos lados.

3.5. ALGORITMOS GENÉTICOS E POPULAÇÕES COOPERANTES

(DALLE MOLE, 2002) buscou aplicar os conceitos de computação paralela em algoritmos genéticos; tal algoritmo está modelado segundo o conceito de populações cooperantes, coordenadas por um nó central.

Nesse modelo, cada nó escravo é responsável por processar uma população com N indivíduos, sendo que a troca de informações entre populações se dá através da migração de indivíduos. Os indivíduos foram modelados como unidades autônomas com a responsabilidade da aplicação dos operadores de *crossover* e mutação. Cada indivíduo é também responsável por determinar sua aptidão. Ao nó central, cabe criar as populações (uma vez que uma população é criada, ela é repassada para um nó escravo, onde operações genéticas de *crossover* e mutação serão executadas em ciclos) e servir de mediador para migrações de indivíduos entre os nós escravos. Este enfoque torna simples a exploração do paralelismo encontrado nos sistemas multiprocessados.

Para testar a implementação, foi escolhido o problema do caixeiro viajante: trata-se de um agente que, dado um grafo não-direcional com pesos representando várias cidades, deve viajar por todos os nós do grafo uma vez com o menor custo total possível (WIKIPEDIA, 2011). Várias configurações de grafo foram utilizadas, com complexidade crescente.

Um indivíduo é composto de uma lista circular contendo todas as cidades do grafo em questão, sendo o espaço de busca igual a (N-1)!/2, onde N é o número de cidades. A função de aptidão é dada por 1/(C), onde C é a soma dos custos de todas as viagens feitas pelo agente. As cidades são representadas por pontos em um plano, e o custo de viagem entre duas cidades é igual a distância entre seus pontos. A seleção dos pares para operações de *crossover* se deu pelo método de dardos.

Para haver troca de informações entre as populações, cada população efetuou, a intervalos variáveis, envio e aceite de indivíduos migrantes. Esses indivíduos são clones do melhor indivíduo de cada população no momento de envio. Esse mecanismo de migração busca acelerar a convergência das populações ao mesmo tempo em que tenta evitar convergência para um mínimo local.

Além das migrações, ao se detectar uma estagnação (caracterizando um possível mínimo local), a população é submetida a uma redução (aonde apenas o indivíduo de maior aptidão tem sua sobrevivência garantida) e posterior crescimento por mutação e seleção de volta até o número máximo.

Para testar a hipótese, foi desenvolvido um sistema *multi-thread* em Java, onde cada população e cada indivíduo (além do nó central) eram executados em uma *thread* diferente. Ao testar o algoritmo nos diversos casos teste propostos, foram obtidas aproximações razoáveis das melhores soluções conhecidas na maioria deles. Verificou-se convergência em tempo similar ao testar poucas (ou apenas uma) populações numerosas e ao testar várias populações menores paralelamente, demonstrando que, mesmo sem o uso de um *grid* computacional com alto poder de paralelismo, o algoritmo trabalhou com eficiência similar ao de um algoritmo não-paralelo. Além disso, ao evoluir várias populações paralelamente, observou-se menor número de mínimos locais.

3.6. SIMULAÇÃO DE COEVOLUÇÃO

Em (EBNER et al., 2010), os autores buscaram analisar quais são as condições propícias para uma corrida armamentista entre várias populações coevoluindo em um mesmo ambiente, tentando isolar fatores importantes que tenham um impacto maior nas dinâmicas de coevolução. Para tal, foi criado um modelo simplificado de simulação de evolução, onde as técnicas evolutivas são abstraídas e representadas pelo simples movimento das populações em um ambiente bidimensional. Para cada posição nesse plano é atribuído um valor de *fitness*, gerando uma paisagem tridimensional; quando as espécies se movem ao longo de seus dois eixos possíveis de movimento, eles deformam os valores de *fitness*, fazendo com que o ambiente esteja em constante mudança, o que ilustra um ambiente de coevolução onde diversas espécies continuamente se adaptam aos novos comportamentos das outras.

Foram propostos três diferentes modelos para como as populações se movem (ou seja, como simulam sua evolução). No primeiro, é computada a direção para a qual o

gradiente de *fitness* do ambiente aponta, e a população se move naquela direção em 1 unidade (velocidade constante). No segundo modelo, a população se move como na primeira, mas proporcionalmente à força do gradiente, gerando uma resposta mais condizente com os modelos clássicos onde a taxa de mudança da população é proporcional à variância do *fitness*. O último modelo, após computar o mesmo gradiente, integra esse gradiente ao longo do tempo; segundo os autores, o efeito da posição anterior da população pode ser interpretado como a manutenção de alguns indivíduos entre as gerações.

Ao posicionar diversas populações de espécies sobre a paisagem, o *fitness* das áreas ao redor destas são deformadas negativamente usando uma função Gaussiana. Essa deformação é cumulativa; várias populações ocupando uma mesma área irão somar suas defromações, e, portanto, áreas pouco visitadas vão aos poucos virando montanhas enquanto áreas sempre habitadas viram vales.

Para melhor ilustrar o fenômeno da coevolução, foi implementado também o conceito de latência: a deformação causada pelas espécies, em um dado passo da simulação, se dá na posição em que a espécie estava posicionada um número de passos atrás. Com essa adição, é possível visualizar um o chamado da "Corrida da Rainha Vermelha": uma espécie localizada em uma dada posição se move na direção de um pico local, mas a deformação persegue a população, dando a impressão de que a população não está melhorando ao longo do tempo.

Dados estes parâmetros, foram realizadas diversas simulações utilizando diferentes combinações, sempre com 40 espécies. Cada combinação gerava um comportamento diferente nas espécies; Utilizar, por exemplo, o primeiro tipo de movimento sem latência em um ambiente de *fitness* inicialmente plano fez com que as espécies inicialmente se espalhassem uniformemente ao longo do plano, e posteriormente se movessem em conjunto na mesma direção, fazendo com que o *fitness* fosse praticamente constante ao longo do ambiente. A simples adição de uma latência de 50 passos, entretanto, fez com que as espécies não conseguissem se distribuir uniformemente; após mais de 16000 passos, elas formaram vários grupos e, então, passaram a se mover em conjunto como no primeiro caso. Entretanto, a formação de grupos fez com que os valores de *fitness* variassem significativamente ao longo do plano, gerando vales e montanhas. Segundo (EBNER et al., 2010), essa simulação

caracterizou uma corrida armamentista entre as espécies, mesmo que ao final o comportamento das espécies tenha se homogeneizado.

Outras combinações de parâmetros geraram resultados parecidos, como comportamentos cíclicos, levando a um estado de utilização ótima do ambiente (ou seja, gerando uma paisagem de *fitness* plana). Comumente, entretanto, a adição de latência (e, em grau menor, a deformação do plano de *fitness* inicial) faziam com que o comportamento das espécies caracterizassem corridas armamentistas. De acordo com (EBNER et al., 2010), a ausência de latência faz com que, ao longo do tempo, as espécies se movam em direção a picos locais e permaneçam estacionárias. Entretanto, com uma latência suficientemente grande, quando uma espécie se move na direção de um pico esse pico continua sendo atrativo para outras espécies por um tempo. Enquanto a influência combinada das espécies não for suficiente para reduzir o pico, mais espécies irão se juntar no local, até que essa influência force o surgimento de novos picos.

3.7. AGENTE INTELIGENTE PARA O JOGO LEMMINGS

Segundo (KENDALL et.al, 2004), o jogo *Lemmings* tem potencial para servir como campo de estudos sobre inteligência artificial da mesma maneira que o jogo de xadrez o fez nas últimas décadas. Entretanto, mesmo havendo outros pesquisadores que pensem de maneira semelhante, não foram encontrados outros trabalhos utilizando esse jogo como plataforma de análise, servindo este trabalho como forma de testar sua hipótese sobre o dito potencial.

Na versão original, *Lemmings* é um jogo para um jogador que consiste de várias entidades humanóides chamadas *lemmings* que se movem em um ambiente bidimensional. Esse ambiente contém uma entrada por onde os *lemmings* são introduzidos no ambiente e uma saída por onde eles podem escapar, sendo que o objetivo de cada fase é resgatar uma quantidade mínima de *lemmings*. O ambiente é composto também de plataformas e paredes e vários perigos às vidas dos *lemmings* na forma de fogo, água, lâminas rotatórias e quedas.

Os *lemmings* têm um entre nove comportamentos no jogo original, mas apenas oito estão presentes no modelo proposto. Inicialmente, os lemmings são do tipo *walker*: caminham em uma plataforma (inicialmente para a direita) até encontrar um obstáculo,

quando invertem seu sentido de movimento. As ações do jogador consistem em transformar um *lemming* em um dos sete tipos abaixo:

- O *basher* cava buracos horizontais em paredes, voltando a ser um walker quando chega ao outro lado
- Um *blocker* fica parado e bloqueia a passagem de outros *lemmings* como uma parede, fazendo-os mudar de direção; no jogo original, cavar o chão sob os pés de um *blocker* o faz virar novamente um *walker*, mas essa funcionalidade não existe no modelo proposto
- Um *bomber* explode em cinco segundos, destruindo o ambiente ao seu redor e morrendo no processo
- Um *builder* constrói uma escada ascendente na direção para a qual estava se movendo, parando após um dado número de degraus ou ao ser bloqueado por uma parede, quando volta a ser um *walker*
- Um *climber* move-se da mesma maneira que um *walker*, mas passa a escalar paredes até o fim da fase; *climbers* ainda podem virar outros tipos especiais, mas nunca perdem a habilidade de escalar
- Um *digger* faz um buraco vertical no chão e volta a ser um *walker* quando acaba de atravessar (similar ao *basher*)
- Um *floater* se move da mesma maneira que o *walker* mas sobrevive a quedas de qualquer altura; da mesma maneira que o *climber*, *floaters* mantém sua habilidade até o fim da fase e podem ser transformados nos outros tipos (um *lemming* com ambas as habilidades de *climber* e *floater* é chamado de *athlete*)

Os comandos possíveis ao jogador são transformar os *lemmings* nos tipos supra-citados e a ação de *nuke*, que transforma todos os *lemmings* ainda vivos e que não tenham escapado em *bombers*. Essa ação é utilizada para eliminar *lemmings* que estejam presos (como os *blockers*), pois uma fase só termina quando todos os *lemmings* conseguiram fugir ou morreram.

Para testar a hipótese de criar um agente inteligente para o jogo, os autores desenvolveram uma versão simplificada do jogo original. Enquanto o jogo original fazia detecção de colisões em nível de pixels, a versão utilizada nos testes realiza movimentos e detecção de colisão dentro de uma grade, onde cada célula é um quadrado de lado 10 pixels. Além disso, enquanto o jogo original funcionava como uma simulação com

cerca de 30 passos por segundo, no modelo proposto a simulação foi realizada com 1 passo por segundo.

Foram criadas sete fases utilizando um editor de mapas criado pelos próprios autores, todas com um limite de tempo máximo de 300 segundos. O primeiro mapa requer o uso de apenas um tipo especial de *lemming* (*basher*), enquanto cada mapa subsequente adiciona a necessidade de um tipo novo.

Foram utilizados algoritmos genéticos para buscar a solução de cada mapa, utilizando seleção por roleta russa com taxa de *crossover* de 0.6 e de mutação de 0.001. Os genes dos indivíduos são formados por uma série de 300 pares de valores inteiros representando o *id* único de um *lemming* e o *id* do papel que deve ser auferido àquele *lemming* (exceto para o comando *nuke*, que se aplica a todos os *lemmings* vivos e tem o distinto *id* 256). Ao longo da simulação, a ação representada por cada um desses pares é executada no segundo correspondente. Para um mapa com um limite de 5 segundos (e portanto 5 passos), um exemplo de gene poderia ser (1,2)(4,1)(2,5)(2,1)(3,1), sendo que no primeiro passo o *lemming* de *id* 1 seria transformado em um *bomber* (ação de *id* 2), no passo dois o *lemming* de *id* 4 seria transformado em um *blocker* (*id* 1), e assim por diante. A avaliação de um indivíduo se da através de uma fórmula que leva em conta o tempo que os *lemmings* levam para escapar, a quantidade de *lemmings* que escaparam, o tempo de vida dos *lemmings* em geral e quantas células do mapa foram exploradas.

Foram utilizadas quatro formas de inicialização para os genes:

- Inicialização TYPEA: Todos os cromossomos são inicializados como pares nulos (-1, -1), ou seja, que não executam nenhuma ação
- Inicialização TYPEB: Todos os cromossomos são inicializados com uma lista de pares aleatórios, como $(8,-1)$ $(4,\ 256)$ $(-1,\ 4)$...
- Inicialização TYPEC e TYPED: Para a primeira fase, são inicializados da mesma maneira que TYPEA e TYPEB, respectivamente. Porém, para as fases subsequentes, são inicializados com a melhor estratégia do estágio anterior

Com todos os modos de inicialização, todos os sete mapas foram solucionados. É interessante, contudo, notar algumas pecurialidades decorrentes das diferentes populações iniciais. Com relação à velocidade de convergência, os genes TYPEC e TYPED tipicamente evoluiam para uma estratégia de sucesso mais rapidamente,

enquanto os TYPEB eram visivelmente mais demorados. Isso se deve a dois motivos: o primeiro, que os genes TYPEC e TYPED mantinham alguma memória das experiências das fases anteriores, que eram possivelmente reutilizadas nos outros mapas. Quanto à ineficiência dos genes TYPEB, (KENDALL et.al, 2004) propõe que, como em *Lemmings* o jogador, durante a maior parte do tempo, deve não realizar ações, a inicialização com valores aleatórios põe o gene em desvantagem. De fato, para resolver a sétima fase criada pelos autores é necessário efetuar apenas 18 comandos, sendo que nos demais 282 passos da simulação o jogador não precisa (e para a maior parte dos comandos possíveis nem deve) efetuar ações.

Já quanto à qualidade das soluções, os autores notaram que os cromossomos TYPEC e TYPED apresentavam soluções de qualidade aceitável, enquanto os TYPEA oscilavam em comparação com os demais. Para a quarta fase, os TYPEA atingiram uma pontuação menor que os demais; entretanto, nas fases 5 e 6 sua pontuação foi a maior, conseguindo inclusive uma solução melhor para a quinta fase do que a proposta pelos autores. A explicação para este efeito, segundo os autores, é que os genes TYPEC e TYPED herdam tanto genes bons quanto genes ruins. Como os mapas tem pequenas diferenças, é possível que uma parte da solução encontrada em um mapa anterior se mostre sub-ótima em um mapa posterior.

3.8. ALGORITMOS GENÉTICOS NO JOGO MASTERMIND

Mastermind (conhecido no Brasil como Senha) é um jogo de tabuleiro para dois jogadores inventado em 1970. O jogo começa com um dos jogadores criando um código secreto, uma sequência de P cores em um conjunto de N cores possíveis (permitindo repetições), sendo que no artigo as N cores são referenciadas como números inteiros crescentes (1, 2, 3..). No jogo original, P tem valor 4 e N, 6. O outro jogador deverá, então, tentar adivinhar a senha em um dado número de tentativas. Após cada tentativa, o primeiro jogador irá revelar duas informações: quantas cores aparecem na posição correta (no artigo representado por X), e quantas aparecem na posição errada (no artigo representado por Y). O objetivo de (BERGHMAN et al., 2009) foi criar um algoritmo genético que desempenhe o papel do segundo jogador.

Os autores dividiram as abordagens existentes na literatura para o jogo Mastermind em três categorias: enumerações completas, regras-de-ouro e meta-heurísticas. As abordagens de enumeração completa investigam todas as senhas dentro do conjunto de

senhas possíveis e escolhem aquela com a maior pontuação em uma função de avaliação. É váido notar que, para os valores padrão de P e N, existem apenas 1296 senhas possíveis, mas que esses algoritmos rapidamente passam a ser inviáveis em casos mais complexos. Entretanto, nessas condições, (BERGHMAN et al., 2009) cita algoritmos que conseguem encontrar a solução em uma média de 4,34 tentativas, analisando todos os casos possíveis. Outros algoritmos conseguiram médias melhores (por volta de 3,835), mas utilizaram apenas 500 jogos possíveis, por questões de tempo computacional.

Nas abordagens de regras-de-ouro, a meta muda para alcançar bons resultados em tempo computacional diminuto. Para isso, a estratégia básica é gerar senhas aleatórias e checar se podem ou não ser a senha do primeiro jogador, utilizando-as então como tentativa. Vários algoritmos fazem isso de maneiras diferentes, com o melhor desempenho sendo de uma média de 4,64 tentativas, avaliando apenas 41,2 combinações em média para os valores padrão do jogo (menos de 4% do total de combinações).

Finalmente, nas abordagens de meta-heurísticas, propostas de uso de algoritmos genéticos e computação evolutiva são feitas para resolver versões mais complexas do jogo. Nesses trabalhos, observa-se um desempenho comparável aos anteriore, como 4,75 e 4,13 para os valores padrão do jogo, e uma escalabilidade aceitável tanto de custo computacional como de taxa de acerto (um dos trabalhos obteve média de 6,169 tentativas para P=5 e N=8 e de 7,079 para P=6 e N=9).

(BERGHMAN et al., 2009) propõe, então, um novo algoritmo para desempenhar o papel do segundo jogador: Para dado P e N, o algoritmo faz uma tentativa inicial fixa. Enquanto a senha não for adivinhada, o algoritmo inicializa um conjunto vazio de senhas válidas (isto é, que, dadas as tentativas anteriores, podem ser a senha verdadeira) e uma população de senhas distintas de tamanho 150. Então, para um número dado de gerações máximas ou senhas válidas máximas, várias novas população são gerada a partir da anterior e as senhas válidas nessas populações são adicionadas ao conjunto. Uma senha é então escolhida de acordo com uma heurística entre as senhas criadas.

A primeira população é aleatória, e as subsequentes são geradas com crossover de 1 ou 2 pontos (0,5 de chance para cada) com dois pais aleatórios da geração anterior, seguido de uma possível mutação de uma das cores (com chance de 0,03), uma possível

permutação de duas cores da senha (chance também de 0,03) e finalmente uma possível inversão (com chance de 0,02). Caso a senha resultante já exista na população, uma senha aleatória é criada em seu lugar. A probabilidade de que uma senha seja escolhida como pai é proporcional ao seu *fitness*, que é calculado comparando-a com todos as tentativas feitas anteriormente: a diferença dos valores de X e Y caso a senha fosse a tentativa e dos valores de X e Y, respectivamente, da tentativa para a senha real é uma indicação da qualidade da senha. Além disso, se, para todas as tentativas anteriores, as diferenças forem iguais a zero, a senha gerada é válida.

Os resultados obtidos, em comparação com as abordagens mencionadas anteriormente, mostraram que o algoritmo proposto por (BERGHMAN et al., 2009) obtem médias próximas aos observados em outras implementações. Entretanto, claramente a proposta dos autores se mostra mais viável tanto em tempo computacional (principalmente contra os algoritmos de enumeração completa) quanto em média de tentativas para valores maiores de P e N.

3.9. AGENTE DE PACMAN USANDO COMPUTAÇÃO EVOLUTIVA

Lançado no início da década de 1980, Ms. Pac-Man é um jogo eletrônico que consiste em guiar um personagem por um labirinto coletando objetos para obter pontos. O desafio do jogo advém da presença de quatro fantasmas, cada um com um comportamento distinto e não-determinístico, que matam o personagem caso façam contato; o jogador tem uma quantidade inicial de vidas, e caso perca todas o jogo acaba. Além dessas regras, existem no labirinto objetos coletáveis especiais denominados *power pills*; quando Ms. Pac-Man "come" uma *power pill*, durante um pequeno período de tempo são os fantasmas que são mortos quando é feito contato com o personagem, feito que confere ao jogador pontos extras. Quando todos os objetos são coletados, um novo labirinto é criado, os fantasmas passam a se mover mais rapidamente e o tempo de duração da *power pill* é reduzido.

Apesar de, nos últimos anos, diversos pesquisadores terem procurado criar agentes para o jogo, o desempenho destes ainda se mostra pequeno frente ao de seres humanos . Os quatro melhores programas desenvolvidos até 2010 obtiveram pontuações máximas de 30.010, 15.640, 9.000 e 8.740, enquanto o recorde mundial para um jogador humano é de 921.360 pontos (GALVÁN-LÓPEZ et al., 2010).

Em seu trabalho, (GALVÁN-LÓPEZ et al., 2010) buscou desenvolver uma série de regras na forma “se <condição> então faça <ação>” para manobrar a personagem através de um labirinto em uma versão simplificada do jogo (com apenas uma fase e uma vida) visando a maior pontuação possível. Para tal, foi criado um algoritmo evolutivo para gerar populações de agentes inteligentes utilizando uma técnica denominada Evolução Gramatical, além de outros quatro agentes: um que agia de maneira completamente aleatória; um que agia de maneira aleatória mas não permitia que o personagem revertesse sua direção; um agente que segue sempre na direção do coletável mais próximo; e um agente criado a mão pelos autores utilizando as mesmas funções disponíveis ao agente inteligente. Foram criados também três “times” de fantasmas; o primeiro time (denominado *Random*) é composto de fantasmas que se movem aleatoriamente, mas não revertem seu sentido de movimento; o segundo (*Legacy*) é composto de três agentes que usam diferentes métricas) de distância (Manhattan, Euclideana e caminho-mais-curto) e tentam sempre ficar o mais próximo possível de Ms. Pac-Man, e um agente igual ao do primeiro time; e um time que busca trancar Ms. Pac-Man entre as junções dos caminhos do labirinto (*Pincer*).

Conforme esperado, os agentes completamente aleatórios tiveram as piores performances do grupo (pontuações máximas entre 200 e 810). É interessante notar, entretanto, que o simples fato de não permitir a inversão de movimento melhorou consideravelmente a performance do agente, que obteve 5.310 pontos contra o time *Legacy*. O agente que buscava os colecionáveis mais próximos apresentou melhor performance em relação ao segundo contra os times *Random* e *Pincer*, mas ainda assim sua maior pontuação, 5.380 contra o *Legacy*, não foi muito diferente da anterior; contudo, as médias das suas pontuações foram bem melhores.

Ao analisar a pontuação dos outros dois agentes, uma melhoria significante é notada. O agente criado a mão pelos autores obteve pontuações máximas de 11.220, 11,740 e 12.820 contra os times *Random*, *Legacy* e *Pincer*, respectivamente. Além disso, as médias de suas pontuações foram de 3 a 4 vezes melhores do que as do agente anterior. Entretanto, mesmo que marginalmente, a melhor performance no quesito de melhor pontuação foi obtida pelo agente evoluído pelo algoritmo: 11.640, 12.350 e 13.830 contra os times *Random*, *Legacy* e *Pincer*, respectivamente. Apesar de sua média ter sido pior contra os times *Random* e *Legacy*, (GALVÁN-LÓPEZ et al., 2010) considerou

que o agente obteve sucesso ao conseguir delinear uma estratégia que atingisse uma melhor pontuação máxima.

3.10. COMPARAÇÕES ENTRE OS ARTIGOS

Os trabalhos supra-citados mostram desde casos de aplicação clássica de algoritmos genéticos, como a busca de soluções otimizadas para o problema do Caixeiro-Viajante ou para o jogo Mastermind (ambientes estáticos) até a simulação de um ecossistema ou a emergência de estratégias para perseguição e fuga (ambientes adaptativos). Em boa parte dos trabalhos, observa-se uma constante na dificuldade para gerenciar o aumento de complexidade ao evoluir as populações; preocupação essa que se mostra ainda mais importante ao trabalhar com adaptabilidade, pois, ao buscar soluções aceitáveis para problemas em ambientes estáticos, esperar uma quantia de tempo grande para alcançar essa solução é uma possibilidade viável, visto que só será necessário executar essa busca uma vez. Em um ambiente não-estático, como por exemplo onde haja um oponente que também evolua suas estratégias ao longo do tempo, uma boa estratégia pode se tornar de baixa qualidade quando o ambiente mudar, conforme ilustrado em (EBNER et al., 2010), requerindo que novas buscas de soluções com o algoritmo genético sejam efetuadas constantemente.

Esse empecilho é de alta relevância para a aplicação de algoritmos genéticos como adversários em jogos: para chegar ao objetivo de fornecer ao jogador um adversário que se adapte continuamente a suas novas estratégias, é necessário que o algoritmo esteja constantemente sendo executado, analogamente ao que foi observado durante a implementação *offline* do gerenciador de cidades para FreeCiv (WATSON et al., 2010).

O trabalho efetuado por (DALLE MOLE, 2002), descrito resumidamente em 3.5, traz uma alternativa para gerenciar o aumento de complexidade: de acordo com seus experimentos, mesmo executando em apenas uma máquina utilizando *threads* para simular paralelismo, o algoritmo obteve eficiência próxima à execução convencional, sendo possível portanto extrapolar que haveria muito ganho de performance ao utilizar tal técnica em um *grid* computacional. Entretanto, é difícil prever quando o poder de processamento paralelo disponível para uso doméstico será suficiente para que essa alternativa seja viável para jogos eletrônicos, ou até mesmo se algum dia será. Isso posto, considerando as novas tecnologias de processadores multi-núcleo, é possível que

em breve o poder de processamento não seja mais tão restritivo quando agora à aplicação deste paradigma.

Nos experimentos de (KOZA, 1991), é possível traçar paralelos fortes com os objetivos do presente trabalho: ambos os experimentos citados em 3.4.1 e 3.4.2 usam evolução em ambientes mutáveis ao utilizar duas populações antagonistas que evoluem simultaneamente. E, apesar de não ser o foco do trabalho, as ideias trabalhadas por (FERNÁNDEZ et al., 2008) e (MORIWAKI et al., 1997), citadas em 3.1 e 3.2 respectivamente, mostram-se ótimos cenários para observar a emergência de comportamentos e estratégias que possam ser consideradas críveis enquanto oriundas de um agente inteligente.

O trabalhos de (KENDALL et. al, 2004) e (GALVÁN-LÓPEZ et al., 2010) demonstram a possibilidade de criar agentes para tarefas complexas mesmo com pouca necessidade de conhecimento sobre o domínio de ação, sugerindo que há viabilidade em automatizar boa parte do processo de criação de oponentes artificiais.

4. PROPOSTA

Para observar o fenômeno da co-evolução através da aplicação de algoritmos genéticos em populações assimétricas, foi proposto criar um ambiente baseado no trabalho de (MORIWAKI et al., 1997). Entretanto, diferente desse experimento, o foco desta implementação será analisar a evolução dos agentes das populações de carnívoros e herbívoros, e não as vantagens e desvantagens entre as diferentes possibilidades de implementação de genes.

4.1. CENÁRIO

O cenário para a implementação foi inspirado em (MORIWAKI et al., 1997). Em um campo bidimensional, duas populações antagonistas (uma população de herbívoros que se alimentam de plantas e uma população de carnívoros que se alimentam de herbívoros) são coevoluídas por um algoritmo genético. Existe também uma população de plantas, cujo comportamento é completamente passivo, sendo seu único propósito servir de alimento para os herbívoros.

As ações possíveis para um indivíduo são semelhantes às utilizadas por (MORIWAKI et al., 1997): cada indivíduo pode Andar (mover-se aleatoriamente em qualquer direção) e Aproximar-se (mover-se em direção ao indivíduo do mesmo tipo do seu mais próximo). Os herbívoros tem ainda a ação Fugir (mover-se em direção oposta ao carnívoro mais próximo), enquanto os carnívoros tem a ação de Caçar (mover-se em direção ao herbívoro mais próximo). Quando, durante a simulação, um indivíduo se encontrar na mesma posição que outro que seja sua presa (herbívoros são presas para carnívoros, e plantas são presas para herbívoros), o indivíduo come a presa.

Cada indivíduo possue um atributo denominado “energia” que é decrementado a cada ação tomada pelo agente; quando esse atributo atinge o valor 0, o indivíduo morre de fome e é removido da simulação. Adicionalmente, caso esse indivíduo seja um herbívoro, três novas plantas são criadas próximo ao local onde o indivíduo morreu. Para recuperar sua energia, os indivíduos devem se alimentar: quando um herbívoro come uma planta, sua energia é incrementada em 30; e quando um carnívoro come um herbívoro, ele ganha a energia restante do herbívoro que foi comido. Se isso fizer com que a energia do indivíduo ultrapasse o valor 100, o indivíduo se reproduz, gerando um novo indivíduo (que tem o mesmo genoma do indivíduo original) e dividindo igualmente sua energia entre pai e filho.

As decisões de um indivíduo são baseadas nas entradas dos seus sensores (exceto para as plantas, que não realizam ações). Cada indivíduo possue um campo de visão de três espaços em todas as direções, totalizando uma área de 7x7 posições; as entradas de seus sensores denotam a presença de carnívoros, herbívoros e plantas nesse campo de visão.

4.2. MODELAGEM DO CROMOSSOMO

Para a modelagem do gene, inicialmente foi proposto utilizar uma *string* de *bits* simbolizando os sensores; entretanto, após os testes iniciais, foi verificado que essa arquitetura limitava o potencial de evolução dos agentes, pois os sensores eram processados sempre na mesma ordem arbitrária especificada na Tabela 3.

Tabela 3. Ordem de verificação do algoritmo nos testes

Carnívoros	Herbívoros
Verifica se tem herbívoro perto	Verifica se tem carnívoro perto
Verifica se tem planta perto	Verifica se tem planta perto
Verifica se tem carnívoro perto	Verifica se tem herbívoro perto
Nada visível	Nada visível

Assim, foi decidido remodelar os genes utilizando n-BDDs, já que o uso deste paradigma possibilitaria alterações na ordem de verificação dos sensores, e até mesmo descarte de sensores. Usando n-BDDs, essa alteração de ordem pode ser feita através de operações genéticas. A Figura 5 ilustra como esse processo gera maior flexibilidade nas ações dos agentes.

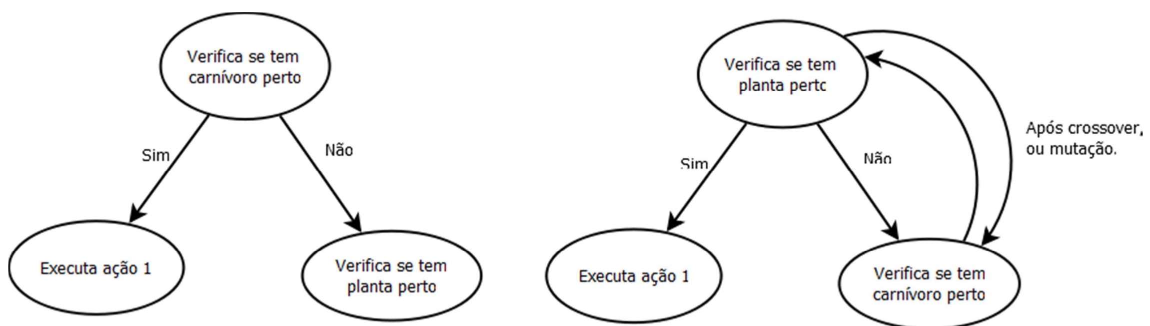

Figura 5. Exemplo de n-BDDs em algoritmos genéticos.

4.3. IMPLEMENTAÇÃO DOS ALGORITMOS

O algoritmo do cenário é baseado em *steps* (passos): em cada *step* cada agente pode executar uma ação, definida por seu cromossomo, e a ação executada consome um

pouco de energia. Após a ação ser executada, é verificado se existe algum *overlap*, ou seja, se há dois indivíduos ocupando a mesma posição. Caso exista, dependendo da combinação de indivíduos, resolvem-se as ações “comer” de carnívoros e herbívoros que estejam em *overlap* com herbívoros e plantas, respectivamente. Nas outras combinações possíveis, nada acontece. Quando um número pré-determinado de *steps* é alcançado, a simulação é interrompida. É gerada a próxima população de indivíduos, passando pelos passos de seleção, *crossover* e mutação. Após a execução do algoritmo genético, a nova população gerada é utilizada para reiniciar a simulação.

A Figura 6 apresenta um pseudo-código do algoritmo responsável por controlar o avanço da simulação. na linha 1, a função *init* irá inicializar as populações de carnívoros e herbívoros em uma posição aleatória e com cromossomos aleatórios, além de inicializar as plantas em posições aleatórias. Na linha 3, a função *step* faz com que as populações de carnívoros e herbívoros executem a próxima ação. Na linha 4, é verificado se a simulação já alcançou o número máximo de *steps*: caso sim, a próxima geração de indivíduos (linhas 5 e 6) é criada, e a simulação reiniciada com essa nova geração (linhas 7 e 8).

```
1. init();
2. while (true) {
3.    step(currentStep);
4.    if (currentStep == maxSteps) {
5.          herbivores = nextGeneration(herbivores);
6.          carnivorous = nextGeneration(carnivorous);
7.          generation++;
8.          currentStep = 0;
9.    } else {
10.         currentStep++;
11.   }
12.}
```

Figura 6. Pseudo-código do loop de simulação

A função *step* chamada na linha 3 é responsável por iterar por todos os indivíduos de ambas as populações e chamar a função *step* de cada um deles. O pseudo-código desta função está representado na Figura 7.

```
1. foreach herb in herbivores {
2.    herb.step(currentStep);
3. }
4. foreach carn in carnivorous {
5.    carn.step(currentStep);
6. }
```

Figura 7. Pseudo-código da função *step* do simulador

Na figura 8 é detalhada a função *step* dos agentes, responsável por fazer com que o agente em questão realize uma ação. Carnívoros e herbívoros são uma subclasse da classe Indivíduo, e utilizam a mesma função *step*. O primeiro passo a ser tomado na função é verificar se o indivíduo está morto (linha 1): caso esteja, o processamento para aquele indivíduo acaba. Caso ele esteja vivo, um array com as condições dos sensores (Carnívoro Visível, Herbívoro Visível, Planta Visível) será criado (linhas 3 à 6), para poder então buscar a ação adequada de acordo com os sensores (linha 7). Após a ação ser processada (linha 8), é decrementada a energia do indivíduo (linha 9). Em seguida, é verificado se existe algum indivíduo na mesma posição em que a posição do indivíduo atual (linha 10 e 11), e caso exista, é verificado se um indivíduo pode “comer” o outro. Dependendo do resultado, ou o indivíduo atual é comido e morre, ou ele come o outro indivíduo, e soma a energia dele na sua (linhas 12 à 16). Caso o indivíduo atual seja um herbívoro (linha 18) e ele esteja morto (linha 19), serão então criadas 3 plantas ao redor do indivíduo atual (linhas 20 à 23). Feito isso, é verificado se a energia do indivíduo é maior que 100 (linha 26), caso seja o indivíduo se reproduz, ou seja, sua energia é dividida por 2, e um novo indivíduo do mesmo tipo, com o mesmo gene, é criado em uma posição próxima (linhas 27 à 29). Por fim é incrementada a variável que diz quantos *steps* ele sobreviveu (linha 31), que será usada como função de aptidão do indivíduo.

```
1. if (dead) return;
2. int energyUsed = 0;
3. boolean[] conditions;
4. conditions[CARNIVOROUS_VISIBLE] = isCarnivorousVisible();
5. conditions[HERBIVORE_VISIBLE] = isHerbivoreVisible();
6. conditions[PLANT_VISIBLE] = isPlantVisible();
7. Action action = getAction(conditions);
8. energyUsed = action.process();
9. myEnergy = myEnergy – energyUsed;
10. Individual ind = checkOverlap();
11. if (ind != null) {
12.   if (canEat(ind)) {
13.         myEnergy = myEnergy + ind.getEnergy();
14.   } else (ind.canEat(this)) {
15.         dead = true;
16.   }
17. }
18. if (isHerbivore()) {
19.   if (dead) {
20.      for i in 0..3 {
21.          Point position = getRandomNearPosition();
22.          grid.getPlants.add(new Plant(position));
23.      }
24.   }
25. }
26. if (myEnergy > 100) {
27.    myEnergy = myEnergy / 2;
28.    Point position = getRandomNearPosition();
29.    grid.getIndividuals().add(new Individual(myEnergy, gene,
position));
30. }
31. stepsSurvived++;
```

Figura 8. Pseudo-código da função *step* dos herbívoros e dos carnívoros

Para finalizar esta breve visão do algoritmo desenvolvido, na Figura 9 está demonstrada a função *nextGeneration*, que cria a próxima população de indivíduos. Na linha 1, são selecionados os indivíduos que serão usados na reprodução. Os método de seleção, *crossover* e mutação utilizados são configurados antes da simulação ser executada. Depois é feita a reprodução dos indivíduos (linha 2) e com os indivíduos gerados a partir da reprodução, é aplicada a mutação. Após a mutação ser executada, a nova população é retornada (linha 4) para que a simulação desta nova população possa ser iniciada.

```
1. List nextGen = selectionMethod.select(individuals);
2. nextGen = crossoverMethod.crossover(nextGen);
3. nextGen = mutationMethod.mutate(nextGen);
4. return nextGen;
```

Figura 9. Pseudo-código do funcionamento do algoritmo genético

Nos testes iniciais, ainda com o cromossomo modelado em uma *string* de *bits*, observou-se que, ao longo das gerações, os cromossomos foram se padronizando, convergindo para as relações condição/ação representada na Tabela 4. Essas relações foram consideradas adequadas.

Tabela 4. Relação de condição/ação dos testes iniciais

	Carnívoros	Herbívoros
Carnívoro Visível	Aproximar	Fugir
Herbívoro Visível	Perseguir/Comer	Aproximar
Planta Visível	Aproximar	Perseguir/Comer
Nada Visível	Procurar/Mover Aleatoriamente	Procurar/Mover Aleatoriamente

Ainda durante os testes do protótipo inicial, era possível presenciar, em algumas condições, o curioso comportamento de herbívoros que ficavam indecisos sobre como agir quando dada a opção entre comer uma planta ou fugir de um carnívoro. Além disso, alguns carnívoros passaram a circular próximos das plantas, aguardando a chegada de herbívoros famintos.

4.4. TRABALHO DESENVOLVIDO

O sistema criado para testar a proposta foi implementado utilizando a linguagem Java.

4.4.1. Parâmetros do Algoritmo Genético

Com o intuito de buscar melhores resultados, foram efetuados diversos testes com métodos de seleção variados para carnívoros e herbívoros:

- Roleta

- Elitismo com Roleta (n=2, ou seja, 10% da população)
- Elitismo com Torneio(n=2, k=4)
- Torneio com Roleta (k=4, ou seja, 20% da população)
- Roleta Rankeada
- Roleta Truncada, (50% melhores da população)

Ao utilizar métodos diferentes para as populações observou-se que os métodos mais agressivos levaram a melhores resultados, fazendo uma população se sobressair em relação a outra. Por exemplo, com uma população A utilizando um método agressivo como a Roleta Truncada, e uma população B utilizando um método menos agressivo, como a Roleta Rankeada, a população A tendia a se sobressair em relação a população B.

Essa diferença na velocidade de convergência pode se mostrar útil para o balanceamento da curva de dificuldade do jogo: trocar os métodos de seleção em tempo de execução pode tornar o *gameplay* mais desafiante para o jogador, caso ele esteja se saindo bem, ou mais fácil. Dessa forma, podemos fazer com que o jogador permança mais tempo no canal de fluxo, não se sentindo nem entendiado, nem desafiado demais.

4.4.2. Balanceamento do Cenário

Durante os testes, na tentativa de buscar melhores resultados, fizemos alterações no cenário proposto com o intuito de melhorar o balanceamento da dificuldade de sobrevivência para os indivíduos.

Inicialmente, tanto carnívoros quanto herbívoros geravam novas plantas quando morriam de fome. Neste caso, os herbívoros tinham ótimo desempenho, mas os carnívoros não conseguiam sobreviver por muito tempo, independente do método de seleção escolhido. Pudemos perceber que havia grande vantagem para os herbívoros em função da abundância de plantas.

Testamos então fazer com que os herbívoros não gerassem mais plantas ao morrer de fome. Apesar de ter diminuído a vantagem (já que não havia mais tanta comida), os herbívoros ainda assim facilmente fugiam dos carnívoros até que grande parte deles morresse, quando então tinham facilidade de sobreviver já que haviam poucos predadores e muita comida.

Para tentar aumentar a oferta de comida para os carnívoros, foi habilitado o comportamento de canibalismo entre carnívoros. Se um carnívoro A estivesse próximo a outro carnívoro B com a vida menor ou igual a um determinado valor, o carnívoro A tentaria comer o carnívoro B, e a energia do carnívoro B se somaria a energia do carnívoro A. Neste caso de teste, considerou-se que os carnívoros passaram a competir demais entre si. Como o objetivo do trabalho é observar a evolução de populações antagonistas competindo entre si, essa mudança foi desconsiderada.

Finalmente, chegou-se a um balanço considerado adequado alterando o valor de energia provido pelas plantas, o valor gasto pelo indivíduo para viver e fazendo apenas os herbívoros gerarem novas plantas.

4.4.3. Interface Gráfica

A Figura 10 mostra a interface desenvolvida para visualizar a simulação, através da qual se pode observar o surgimento de comportamentos dos agentes. A grade na parte esquerda representa o campo por onde os agentes podem se mover, onde cada célula representa uma possível posição que os indivíduos podem ocupar. Os herbívoroes são representados por círculos verdes, os carnívoros por círculos vermelhos e as plantas pelos círculos de cor marrom.

Ainda na Figura 10 é possível visualizar gráficos na parte direita que mostram a média da função de avaliação (linha preta) e o máximo da mesma (linha vermelha) para as 15 últimas gerações de herbívoros (gráfico na parte superior) e carnívoros (na parte inferior). A interface contém também controles para acelerar a simulação (controlando o máximo de *steps* a serem executados por segundo) e um botão para pausar a simulação. Finalmente, na parte inferior são informadas a geração atual e qual o *step* atual da simulação.

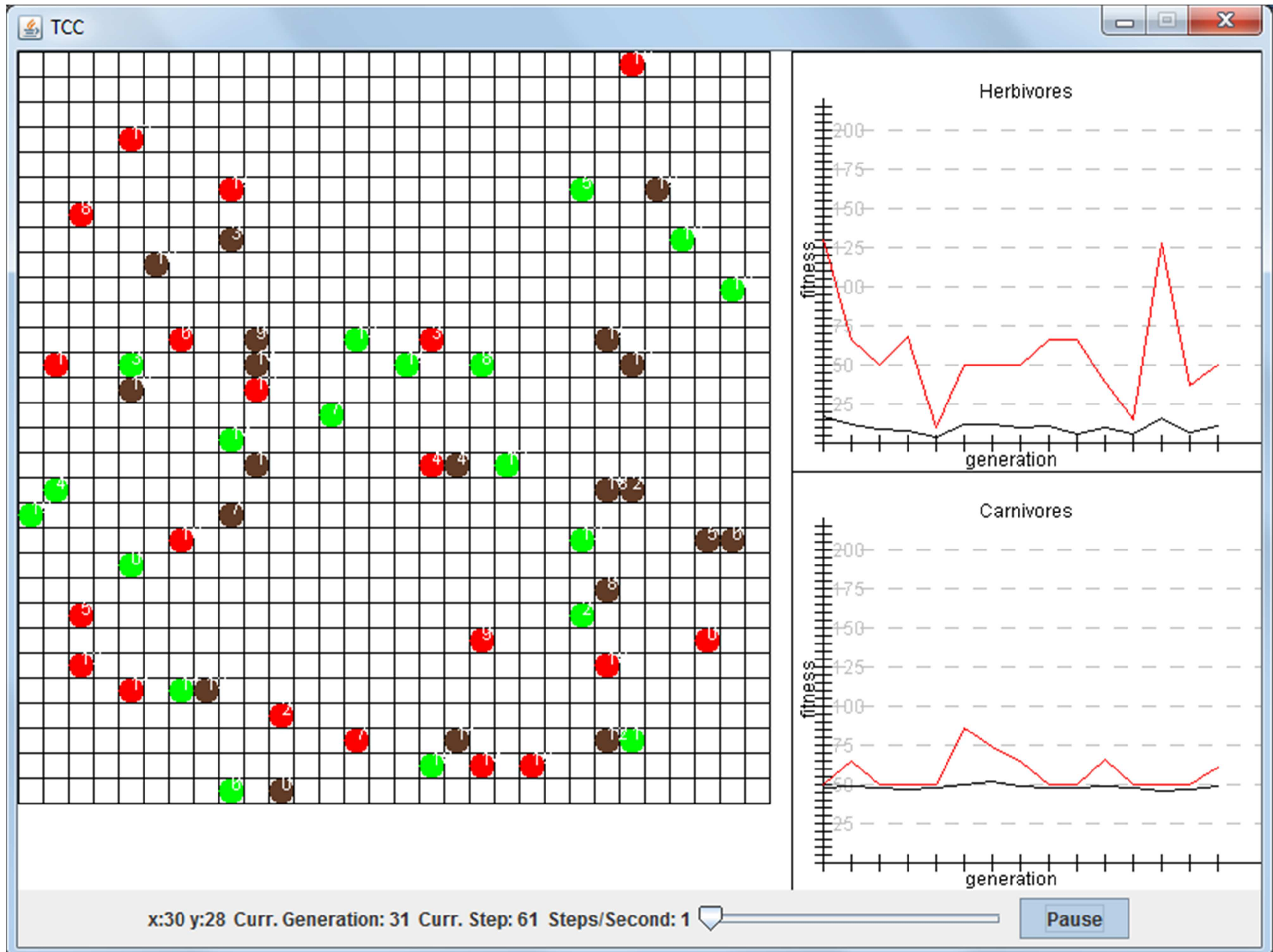

Figura 10. Interface gráfica do simulador

4.4.4. Processo de tomada de decisão dos indivíduos

Um indivíduo determina suas ações a partir de sua árvore de decisão, cujas entradas são provenientes de seus sensores visuais. Considerando o carnívoro representado no centro da Figura 11 (colorido em cor azul para facilitar sua identificação), as entradas sensoriais indicariam a presença de carnívoros, herbívoros e plantas em sua proximidade. A Figura 12 mostra uma possível árvore de decisão para um carnívoro, indicando na cor vermelha o caminho que seria percorrido através da árvore caso esta árvore fosse a do carnívoro azul da Figura 11.

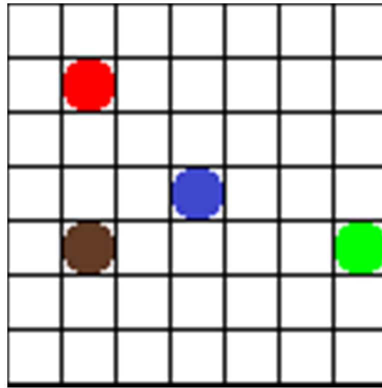

Figura 11. Detalhamento dos sensores visuais de um indivíduo

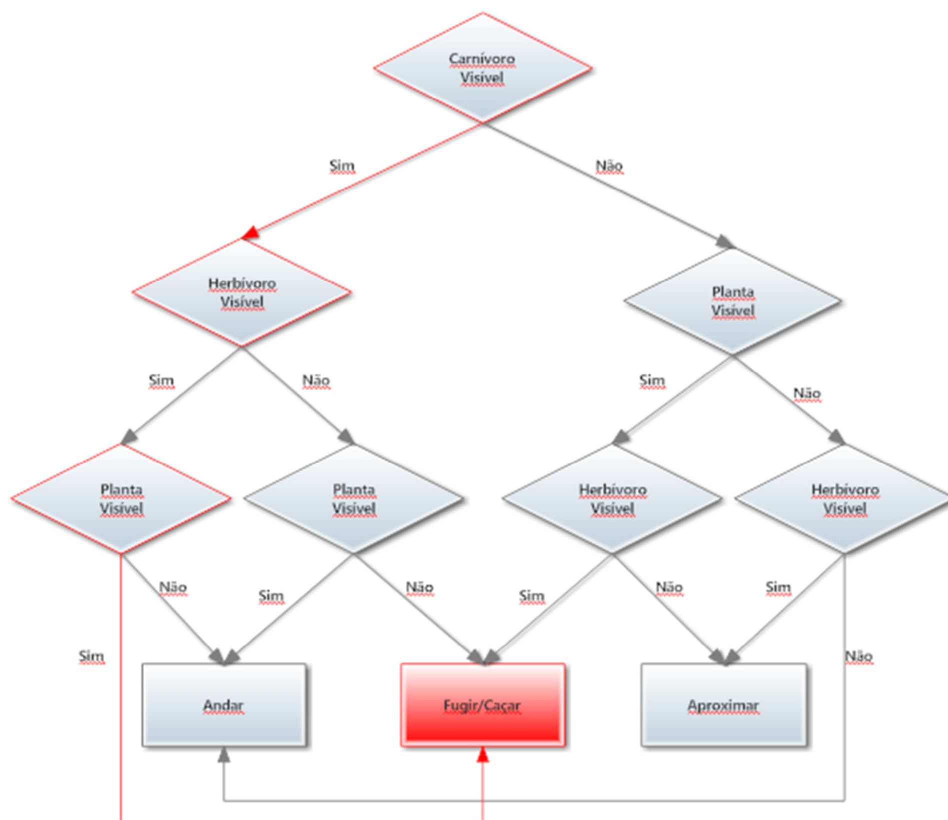

Figura 12. Exemplo de árvore de decisão

Após encontrar a ação a ser executada de acordo com seus sensores, essa ação será processada de acordo com pesos atribuídos aos indivíduos visíveis, baseado em sua distância em relação ao indivíduo e na ação sendo executada. A Figura 13 mostra os pesos atribuídos para cada indivíduo no campo de visão do carnívoro azul da Figura 11.

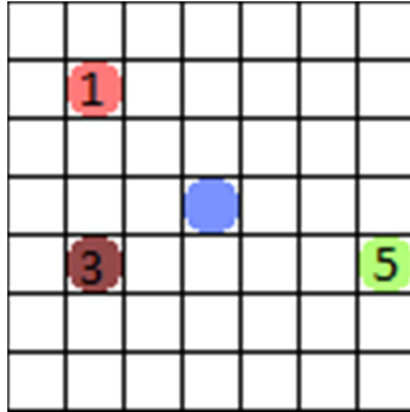

Figura 13. Exemplo de atribuição de pesos de um indivíduo

Feito isso, o indivíduo buscará o ponto com o maior peso. Caso este ponto seja um dos pontos ao redor do indivíduo ele passará a ocupar essa posição; caso não seja, se moverá para o ponto que o deixará mais próximo do ponto alvo. No exemplo, o processamento da ação resultará no movimento do indivíduo sendo analisado para o ponto ao redor dele que seja o mais próximo possível do herbívoro de peso 5. O estado da grade após o movimento do carnívoro está representado na Figura 14.

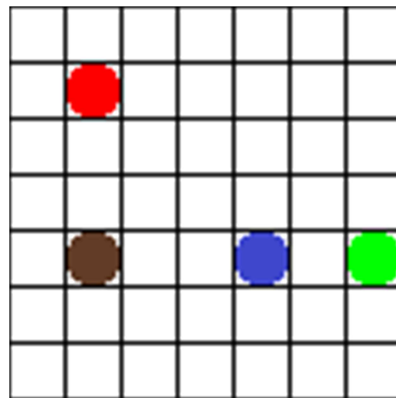

Figura 14. Estado da grade após o carnívoro (azul) executar a ação de caçar

4.4.5. Comportamento de Ecossistema

Assim como foi observado por (MORIWAKI et al., 1997), observou-se picos alternados entre o desempenho das populações, caracterizando uma cadeia alimentar.

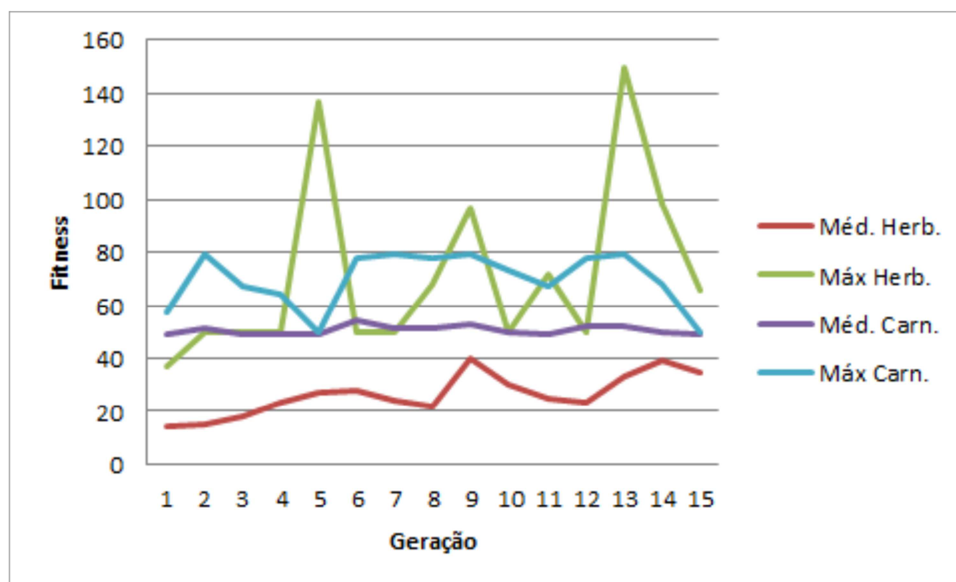

Figura 15. Desempenho das espécies durante simulação

5. REFERÊNCIAS BIBLIOGRÁFICAS

AKERS, S. B. Binary Decision Diagrams. **IEEE Transactions on Computers**, Washington, DC, EUA, v. C-27, n. 6, p. 509-516, jun. 1978. Disponível em <http://web.cecs.pdx.edu/~shahd/BDD_Akers.pdf> Acesso em: 29/06/2011

BERGHMAN, L.; GOOSENS, D.; LEUS, R. Efficient Solutions for Mastermind using Genetic Algorithms. **Computer and Operations Research**, v. 36, n. 6, 2009.

DALLE MOLE, V. L. **Algoritmos Genéticos - Uma Abordagem Paralela Baseada em Populações Cooperantes**. 2002. 97 f. Dissertação (Mestre em Ciência da Computação) - Universidade Federal de Santa Catarina. Florianópolis, SC. 2002.

EBNER, M.; WATSON, R. A.; ALEXANDER, J. Coevolutionary Dynamics of Evolutionary Species. In: EvoStar, 2010, Istanbul Technical University, Istanbul, Turquia. **Proceedings...** Istanbul Technical University, Istanbul, Turquia, 2010. Disponível em <http://www.bibsonomy.org/bibtex/1101e2600db45c4975a23a1aa9b28c5a7/dblp> Acesso em: 13/05/2012

EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. 1.ed. New York, New York, EUA: Springer, 2003.

FERNÁNDEZ, A. J.; COTTA, C.; CEBALLOS, R. C. Generating Emergent Team Strategies in Footbal Simulation Videogames via Genetic Algorithms. In: GAMEON'08, 9., 2008, Universidad Politècnica de València, Valência, Espanha. **Proceedings...** Universidad Politècnica de València, Valência, Espanha, 2008. Disponível em <http://www.lcc.uma.es/~afdez/Papers/Fernandez_Game_ON2008.pdf> Acesso em: 28/06/2011

GALVÁN-LOPEZ, E.; SWAFFORD, J. M.; O'NEILL, M.; BRABAZON, A. Evolving a Ms. PacMan Controller using Grammatical Evolution. In: EvoStar, 2010, Istanbul Technical University, Istanbul, Turquia. **Proceedings...** Istanbul Technical University, Istanbul, Turquia, 2010. Disponível em < http://irserver.ucd.ie/dspace/bitstream/10197/2593/1/evolvingMsPacmanControllerEvoGames2010.pdf> Acesso em: 13/05/2012

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 29.ed. Crawfordsville, Indiana, EUA: RR Donnelley Crawfordsville, 2009.

KENDALL, G.; SPOERER, K. Scripting the Game of Lemmings with a Genetic Algorithm. In: IEEE Congress on Evolutionary Computation, 2004, Portland, Oregon, EUA. **Proceedings...** IEEE Congress on Evolutionary Computation, 2004, Portland, Oregon, EUA, 2004. Disponível em < http://www.fcet.staffs.ac.uk/rgh1/cai/foster/GA%201.pdf> Acesso em: 13/05/2012

KOZA, J. R. Evolution and Co-Evolution of Computer Programs to Control Independently-Acting Agents. In: International Conference on Simulation of Adaptive Behavior, 1., 1991, Paris, França. **Proceedings...** Cambridge, Massachussets, EUA: The MIT Press, 1991. Disponível em <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.140.7387> Acesso em: 28/06/2011

MITCHELL, M. **An Introduction to Genetic Algorithms**. 1.ed. Cambridge, Massachussets, EUA: The MIT Press, 1998.

MORIWAKI, K.; INUZUKA, N.; YAMADA, M.; SEKI, H.; ITOH, H. A Genetic Method for Evolutionary Agents in a Competitive Environment. In: On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2), 2., 1997, Internet. **Proceedings...** Abingdon, Oxfordshire, Inglaterra: Springer, 1998. Disponível em <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.3897> Acesso em: 28/06/2011

ROBOCUP. **RoboCup**. Disponível em <http://www.robocup.org/> Acesso em: 28/06/2011

SCHELL, J. **The Art of Game Design**. 1.ed. Burlington, Massachussets, EUA: Morgan Kaufmann, 2008.

WATSON, I.; AZHAR, D.; CHUYANG, Y.; PAN, W.; CHEN, G. **Optimization in Strategy Games: Using Genetic Algorithms to Optimize City Development in FreeCiv**. Relatório parcial. In: ____, 2010. Disponível em <http://www.cs.auckland.ac.nz/research/gameai/projects/GA%20in%20FreeCiv.pdf> Acesso em: 28/06/2011

WIKIPEDIA. **Charles Darwin – Wikipedia, the free encyclopedia**. Disponível em <http://en.wikipedia.org/wiki/Charles_Darwin> Acesso em: 28/06/2011

WIKIPEDIA. **Gregor Mendel – Wikipedia, the free encyclopedia**. Disponível em <http://en.wikipedia.org/wiki/Gregor_Mendel> Acesso em: 28/06/2011

WIKIPEDIA. **Left 4 Dead – Wikipedia, the free encyclopedia**. Disponível em <http://en.wikipedia.org/wiki/Left_4_dead> Acesso em: 30/06/2011

WIKIPEDIA. **Traveling Salesman Problem – Wikipedia, the free encyclopedia**. Disponível em <http://en.wikipedia.org/wiki/Travelling_salesman_problem> Acesso em: 27/06/2011