COBERTURA EM AMBIENTES 3D ARBITRÁRIOS: O PROBLEMA DA GALERIA DE ARTE EM JOGOS DE TIRO

EDUARDO PENHA CASTRO FANTINI

COBERTURA EM AMBIENTES 3D ARBITRÁRIOS: O PROBLEMA DA GALERIA DE ARTE EM JOGOS DE TIRO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais - Departamento de Ciência da Computação.
como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Chaimowicz

Belo Horizonte

Janeiro de 2014

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Fantini, Eduardo Penha Castro.

F216c Cobertura em ambientes 3D arbitrários: o problema da galeria de arte em jogos de tiro / Eduardo Penha Castro Fantini. — Belo Horizonte, 2014.
xxviii, 77 f. : il. ; 29cm.

Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação.

Orientador: Luiz Chaimowicz.

1. Computação - Teses. 2. Computação gráfica - Teses. 3. Geometria computacional - Teses. 4. Jogos digitais - Teses. I. Orientador. II. Título.

519.6*83(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Cobertura em ambientes 3D arbitrários: o problema da galeria de arte em jogos de tiro

EDUARDO PENHA CASTRO FANTINI

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. GISELE LOBO PAPPA
Departamento de Ciência da Computação - UFMG

PROFA. SORAIA RAUPP MUSSE
Faculdade de Informática - PUCRS

Belo Horizonte, 20 de janeiro de 2014.

Para meu pai, Marco Antonio Cioni Fantini (in memoriam) e meu tio, Lino Wander Cioni Fantini (in memoriam)...

Agradecimentos

Agradeço, em especial, ao orientador Luiz Chaimowicz, que foi uma imensa fonte de incentivo e conselhos ao longo dos últimos anos. Por ser um grande professor, colega de trabalho e amigo, me ajudando a superar os obstáculos com a sua compreensão, tranquilidade e humanidade.

Ao amigo e psicanalista Eduardo Dias Gontijo, por me ajudar a entender e aceitar as perdas familiares que sofri ao longo do curso e a valorizar a superação de cada dia.

A minha amada família e à Laiane por me apoiarem incondicionalmente e por compreenderem a minha ausência em momentos dedicados ao curso de mestrado.

À professora Gisele Pappa, pelas aulas motivantes de Computação Natural, as quais me fizeram repensar o rumo da pesquisa e iluminar o caminho de meus objetivos por meio do conhecimento.

Aos amigos Marlos Cholodovskis, Denise Notini e Flávio Coutinho pela amizade, companhia nos simpósios e contribuição considerável de suas pesquisas para a área de jogos, ajudando no crescimento do Laboratório Multidisciplinar de Pesquisa em Jogos Digitais.

Aos amigos Erickson Nascimento, Armando Neto, Antônio Wilson, Elizabeth Duane, Rodolfo Carneiro, Cláudio Fernandes, Fabrício Carvalho e Vinicius Graciano do Laboratório de Visão Computacional e Robótica pela companhia, trocas de conhecimento e conversas divertidas.

Aos funcionários da secretaria do Programa de Pós Graduação em Ciência da Computação pelo suporte impecável a tudo o que precisei durante o curso.

Por fim, a todos que contribuíram, deforma direta ou indireta, para o desenvolvimento desse trabalho.

"A mente que se abre a uma nova idéia
jamais voltará ao seu tamanho original."
(Albert Einstein)

Resumo

O Problema da Galeria de Arte consiste em determinar o número mínimo de observadores necessários para cobrir um ambiente tal que cada ponto no espaço é coberto por, pelo menos, um observador. Este é um problema NP-Difícil bem conhecido no campo da geometria computacional. Na literatura, várias restrições são aplicadas a ambientes 2D e 3D para estudar e resolver o problema em tempo polinomial, por exemplo, usando polígonos simples, ambientes ortogonais e planares, etc.

Neste trabalho, apresentamos uma solução aproximada e polinomial com base em algoritmos genéticos meta-heurísticos que podem ser aplicados a ambientes 3D genéricos sem qualquer restrição e, portanto, aplicável em jogos de tiro e também em ambientes do mundo real. A solução utiliza técnicas de (i) computação gráfica para gerar pontos de amostragem no ambiente, (ii) teste de interseção raio-malha para gerar um gráfico de visibilidade entre as amostras e (iii) algoritmos genéticos para encontrar e otimizar o conjunto mínimo de observadores.

Os mapas do jogo Counter-Strike foram utilizados para analisar o posicionamento de pequenos grupos de observadores em ambientes complexos com obstáculos. Os motores de jogo Half-Life e Irrlicht foram usados para aplicar o teste de interseção raio-malha em ambientes 3D. Vários experimentos foram realizados, e os resultados mostram que nossa metodologia é capaz de obter uma boa cobertura de espaços complexos com um pequeno número de agentes de observação.

Palavras-chave: Problema da Galeria de Arte, Geometria Computacional, Computação Natural, Computação Gráfica, Jogos Digitais.

Abstract

The Art Gallery Problem consists in determining the minimum number of observers required to cover an environment such that each point of space is seen by at least one observer. This is a NP-Hard problem well known in the field of computational geometry. In the literature, several restrictions are applied to 2D and 3D environments to study and solve the problem in polynomial time, for example the use of simple polygons, orthogonal and planar environments, etc.

In this work we present an approximate and polynomial solution based on metaheuristic genetic algorithms that can be applied to general 3D environments without any restriction and, therefore, applicable in shooter games and also real-world environments. The solution uses the techniques of (i) computer graphics to generate sample points in the environment, (ii) ray-mesh intersection test to generate a graph of visibility between the samples and (iii) genetic algorithms to find and optimize the minimum set of observers.

The maps of the game Counter-Strike were used to analyze the placement of small groups of observers in complex environments with obstacles. The game engines Half-Life and Irrlicht were used to apply the ray-mesh intersection test in 3D environments. A series of experiments were performed and the results show that our methodology is capable of obtaining a good coverage of complex spaces with a small number of agents observing.

Keywords: Art Gallery Problem, Computational Geometry, Natural Computing, Computer Graphics, Digital Games.

Lista de Figuras

1.1 Exemplo de spotting no jogo Battlefield 3. 2

1.2 Exemplo de posicionamento estratégico utilizando o software ATC. 4

2.1 Exemplos de ambientes planares. Sendo (a) um polígono simples ortogonal, (b) um polígono não ortogonal e (c) um polígono não ortogonal com buraco. 8

2.2 Exemplo de um ambiente 3D arbitrário complexo. 8

2.3 Metodologia de Marengoni et al. [2000] para o Minimum Vertex Guard. . . 10

2.4 Metodologia de González-Banos [2001]. 11

2.5 Exemplo de três guardas monitorando três subpolígonos r-star (a). Exemplos de decomposição de acordo com a visibilidade entre duas amostras de pontos q e k: l-star, s-star e r-star da esquerda pra direita (b). 12

2.6 Um polígono ortogonal (a) e outro não ortogonal (b), ambos sem buracos e com guardas posicionados nos vértices da borda. 13

2.7 Exemplo de um polígono simples ortogonal sem buracos (a) e a triangularização correspondente, destacando-se os centroides de cada subpolígono (b). 14

2.8 Em (a) estão destacadas as reflex edges, as quais indicam o posicionamento de guardas para ambientes ortogonais. Em (b) destacam-se em pontilhado as arestas de junções dos blocos de volumes. 14

2.9 Exemplos de coberturas aproximadas por arestas utilizando reflex edges aplicadas a ambientes ortogonais não planares. 15

3.1 Passo 1a da metodologia. Amostras de pontos são geradas em um ambiente tridimensional arbitrário. 18

3.2 Passo 1b da metodologia. As amostras são convertidas em vértices para o grafo de visibilidade. 18

3.3 Passo 2 da metodologia. O grafo de visibilidade é gerado a partir dos vértices de amostra e as arestas indicam quais pontos são visíveis entre si no ambiente tridimensional. 19
3.4 Passo 3a da metodologia. Calcula-se o MVG do grafo de visibilidade, atribuindo a alguns vértices a função de guardas do ambiente (vértices azuis). 19
3.5 Passo 3b da metodologia. O MVG (vértices azuis) retorna ao ambiente original para marcar s posições dos guardas no espaço. 20
3.6 Exemplos de métodos de amostragem clássicos. Amostragem uniforme (a), amostragem uniforme aleatória (b), amostragem aleatória (c) e distribuição de Poisson (d). 21
3.7 Editor de waypoints (barras verticais) do Counter-Strike. Detalhe para as informações do waypoint. 22
3.8 Distribuição dos waypoints no mapa De_airstrip do Counter-Strike. 23
3.9 Exemplo de amostragem uniforme por volume. 23
3.10 Etapas para a validação de amostras. A partir de um ambiente complexo (a) são criadas zonas verdes e vermelhas (b) que resultam na zona de interesse. Após a amostragem uniforme (c), apenas os vértices que pertencem à zona de interesse são considerados válidos (d). 24
3.11 Exemplo do funcionamento do hit-testing. Dado um conjunto de amostras em um ambiente 3D (a), são gerados raios entre cada par de arestas. Se houver uma colisão do raio (pontilhado amarelo) com o cenário, as amostras não são intervisíveis. Caso contrário, elas são intervisíveis e uma aresta (verde) é criada (b). 26
3.12 Representação de uma solução candidata para o MVG ou MSC. 27
3.13 Exemplos de passos do Algoritmo 3.1. 30
3.14 O funcionamento do algoritmo genético. 32
3.15 Exemplo de cruzamento por ponto. Os genótipos pais são segmentados em uma posição aleatória e geram dois indivíduos novos. 34
3.16 Comportamento da função fitness no espaço de soluções candidatas. 35
3.17 Zoom do comportamento da função fitness no espaço de soluções candidatas. 35
3.18 Média da fitness média ao longo das gerações na calibração do AG. 37
3.19 Média do tamanho médio do AMVG ao longo das gerações na calibração do AG. 38
3.20 Média da melhor fitness ao longo das gerações na calibração do AG. 38
3.21 Média da média de soluções idênticas ao longo das gerações na calibração do AG. 39

3.22 Média da média de filhos melhores que os pais ao longo das gerações na calibração do AG. 39

4.1 Exemplo de um grafo esparso com 4 guardas. 42
4.2 Evolução da média da fitness média no grafo esparso de 500 vértices. . . . 43
4.3 Evolução da média do tamanho médio do AMVG no grafo esparso de 500 vértices. 44
4.4 Evolução da melhor fitness no grafo esparso de 500 vértices. 44
4.5 Comparação do tempo de execução dos algoritmos exato e genético para um grafo esparso de 500 vértices. 45
4.6 Evolução da média da fitness média em alguns mapas do Counter-Strike. . 48
4.7 Evolução da média do tamanho médio do AMVG em alguns mapas do Counter-Strike. 48
4.8 Evolução da melhor fitness em alguns mapas do Counter-Strike. 48
4.9 Cobertura por camp points (a) e pelo AMVG (b) no mapa De_dust. . . . 49
4.10 Avaliação da função fitness no espaço de soluções para cobertura de 60%. . 50
4.11 Evolução da média da fitness média para taxas de cobertura distintas no mapa De_cbble. 51
4.12 Evolução da média do tamanho médio do AMVG para taxas de cobertura distintas no mapa De_cbble. 51
4.13 Evolução da melhor fitness crobrindo 70% do mapa De_cbble. 51
4.14 AMVGs para coberturas de (a) 60%, (b) 70%, (c) 80%, (d) 90% e (e) 100% das amostras do mapa Cs_italy. 53
4.15 Coberturas completas no mapa De_survivor. As Figuras (a) e (b) foram os resultados do MVG de tamanho 3 e a Figura (c) foi o resultado do AMVG de tamanho 4. 55
4.16 Volume de validação criado para otimizar as amostras da distribuição uniforme no cenário Cs_assault. 56
4.17 Amostragem uniforme no cenário Cs_assault. 58
4.18 Amostras validadas no cenário Cs_assault a partir do volume de interesse. 58
4.19 Evolução da média da fitness média do AG sobre as amostras uniformes no cenário Cs_assault. 59
4.20 Evolução da média do tamanho médio dos AMVGs do AG sobre as amostras uniformes no cenário Cs_assault. 59
4.21 Evolução da melhor fitness do AG sobre as amostras uniformes no cenário Cs_assault. 59

4.22 Grafos de visibilidade da amostragem uniforme (a) e da amostragem por waypoints (b) do mapa Cs_assault. 60
4.23 AMVGs da amostragem uniforme (a) e da amostragem por waypoint (b) do mapa Cs_assault. 61

Lista de Tabelas

2.1 Aplicabilidade das principais metodologias presentes no estado da arte. . . 16

3.1 Resultados dos testes de calibração do AG. 37

4.1 Comparativo entre a aplicação dos algoritmos exato e genético em grafos esparsos. 45
4.2 Algoritmo genético em waypoints dos mapas do Counter-Strike. 47
4.3 Tradeoff dos tamanhos do AMVG para diferentes taxas de cobertura das amostras. 52
4.4 Resultado das MFMs para diferentes taxas de cobertura das amostras. . . 52
4.5 Resultados comparativos entre os algoritmos exato e genético para mapas com AMVGs reduzidos. 54
4.6 Resultados do algoritmo genético em diferentes métodos de amostragem. . 57

Lista de Algoritmos

3.1 Algoritmo exato para calcular o MSC. 28
3.2 Verificação de cobertura total de amostras. Função checkFullCoverage(). 29
3.3 Algoritmo genético. 33

B.1 Algoritmo para extração de dados no Counter-Strike. 76

Lista de Abreviaturas e Siglas

AG Algoritmo Genético

AIIDE *Association for the Advancement of Artificial Intelligence*

AMVG *Approximated Minimum Vertex Guard*

API *Application Programming Interface*

ATC *Advanced Tactical Center*

AVP *Atomic Visibility Polygons*

bots Agentes Virtuais

CGAMES *International Conference on Computer Games*

DEM *Digital Elevation Map*

DLL *Dynamic-link library*

MEG *Minimum Edge Guard*

MFM Média da Fitness Média

MFP Média do Número Médio de Filhos Melhores que os Pais

MMC Método de Monte Carlo

MMF Média da Melhor Fitness

mods *Game Modifications*

MSC *Minimum Set Cover*

MSI Média da Quantidade Média de Soluções Idênticas

MTM Média do Tamanho Médio dos AMVGs

MVG *Minimum Vertex Guard*

PGA Problema da Galeria de Arte

SBGAMES Simpósio Brasileiro de Jogos e Entretenimento Digital

SCP *Set Covering Problem*

SDK *Software Development Kit*

Sumário

Agradecimentos **ix**

Resumo **xiii**

Abstract **xv**

Lista de Figuras **xvii**

Lista de Tabelas **xxi**

Lista de Algoritmos **xxiii**

Lista de Abreviaturas e Siglas **xxv**

1 Introdução **1**

1.1 Motivações 3

1.2 Objetivo 3

1.3 Contribuições 4

1.4 Organização do Trabalho 5

2 Trabalhos relacionados **7**

2.1 Preliminares 7

2.2 Trabalhos relevantes 9

3 Metodologia **17**

3.1 Primeiro passo: Amostragem de pontos 20

3.1.1 Distribuição de Poisson (waypoints) 22

3.1.2 Amostragem uniforme por volume 23

3.2 Segundo passo: Criação do grafo de visibilidade 25

3.2.1 Teste de interseção raio-malha 25

3.3 Terceiro passo: Busca pelo MVG 27
3.3.1 Algoritmo exato 27
3.3.2 Algoritmo genético 31

4 Experimentos e resultados **41**
4.1 Algoritmos exato e genético em grafos esparsos 41
4.1.1 Algoritmo genético 42
4.1.2 Algoritmo exato 44
4.2 Algoritmo genético em waypoints do Counter-Strike 46
4.3 Tradeoff entre cobertura e número de guardas 49
4.4 Algoritmo exato aplicado no Counter-Strike 54
4.5 Algoritmo genético em um ambiente arbitrário 55

5 Conclusão e trabalhos futuros **63**
5.1 Conclusão 63
5.2 Trabalhos futuros 64

Referências Bibliográficas **67**

Apêndice A Jogos aplicáveis em experimentos **71**
A.1 Tipos de plataformas 71
A.2 Lista de plataformas por gênero 72
A.2.1 Jogos de ação 72
A.2.2 Jogos de aventura 72
A.2.3 Jogos de plataforma 72
A.2.4 Jogos de interpretação 73
A.2.5 Jogos de simulação 73
A.2.6 Jogos de esporte 73
A.2.7 Jogos de estratégia 73

Apêndice B Utilizando o SDK do Half-Life I **75**
B.1 Instalação do E[POD]-Bot 75
B.2 Extração de dados com o E[POD]-Bot 76

Capítulo 1

Introdução

Na área de jogos digitais, mais especificamente no gênero de jogos *first-person shooter* (FPS) ou simplesmente jogos de tiro, há uma constante evolução das simulações de combate. A partir do lançamento do título *Doom* em 1993, os jogos FPS apresentam a funcionalidade multijogadores, exigindo também uma crescente colaboração em equipe.

Entre as diversas táticas de colaboração, o posicionamento estratégico é uma das mais importantes. É essencial que cada jogador da equipe informe o posicionamento dos oponentes avistados, uma ação conhecida como *spotting*. Essa informação pode ser passada por meio de voz, texto ou elementos interativos do próprio jogo. Dessa forma é possível abater os oponentes de forma mais eficiente [Choe, 2011].

Na Figura 1.1 é demonstrada uma situação de *spotting* na qual um inimigo é marcado ao ser avistado. Essa marcação fica visível para toda a equipe no mini mapa do jogo e também no cenário, destacados pelos círculos vermelhos. Ainda nessa figura podemos observar um aliado, destacado pelo círculo verde, agindo de forma colaborativa para abater o oponente marcado.

Como o número de jogadores por equipe é limitado, variando de quatro a trinta e dois dependendo do jogo, determinar o número mínimo de pontos de visão ou *viewpoints*, capaz de monitorar o ambiente ao máximo é uma tarefa desafiadora e crucial para o sucesso de uma equipe. Esse é um problema clássico de visibilidade formulado no campo da geometria computacional e denominado de Problema da Galeria de Arte (PGA) [Berg et al., 2008].

Este problema de visibilidade e de algumas de suas variações são problemas reais com várias aplicações práticas além de jogos digitais. A colocação de antenas para as companhias de telefones celulares, em que o número de antenas deve ser minimizado, é um deles. Um problema similar é o de calcular a cobertura de um novo conjunto de antenas colocadas em algumas posições desejadas. O posicionamento de câmeras para

Figura 1.1. Exemplo de spotting no jogo Battlefield 3.

fins de segurança em bancos, supermercados ou lojas de departamento é outra. Em cenários militares, os comandantes precisam colocar olheiros para cobrir uma determinada região, ou, de modo alternativo, decidir aonde esconder seus recursos analisando áreas de menor visibilidade [Marengoni et al., 2000]. Outra aplicação bastante estudada é o posicionamento e movimentação eficientes de pequenas quantidades de robôs equipados com *scanners* com o intuito de digitalizar ambientes fechados [Nüchter et al., 2003].

O PGA foi introduzido em 1973 por Victor Klee e consiste em determinar a quantidade mínima de guardas estacionários necessários para monitorar o interior de uma galeria com n paredes [Honsberger, 1976]. Em 1975, Chvátal estabeleceu um teorema que ficou conhecido como Chvátal's Art Gallery Theorem: $n/3$ guardas ocasionalmente são necessários e sempre suficientes para cobrir um polígono com n vértices. A partir daí diversos teoremas para instâncias específicas do problema foram desenvolvidos e agrupados em livros dedicados ao tema [O'Rourke, 1987; Urrutia, 2000; Michael, 2009].

Lee & Lin [1986] estudaram a complexidade computacional do problema galeria de arte e suas variações. Provou-se que o problema de determinar o número mínimo de guardas para cobrir uma galeria de arte conectada é NP-Difícil. A prova foi modificada para mostrar que o problema de determinar o número mínimo de guardas para uma região poligonal, sendo esses guardas representados por vértices ou arestas, é também NP-difícil. Como subproduto, o problema da decomposição de um polígono simples para um número mínimo de polígonos em forma de estrela (*star-shaped polygons*), tais que a sua união é o polígono original, também foi provado ser NP-Difícil.

No estado da arte são conhecidos algoritmos de aproximação com relações de aproximação logarítmicas [Efrat & Har-peled, 2006; Ghosh, 1987; González-Banos, 2001]

para as versões restritas do problema, por exemplo, exigindo guardas para serem posicionados nos vértices ou nos pontos de um grid que discretiza o ambiente. Aproximações de fator constante são conhecidas por guardar terrenos 1.5D e polígonos monótonos[1] [Ben-Moshe et al., 2005; King, 2006; Nilsson, 2005], e métodos exatos são conhecidos para o caso especial de visibilidade *r-star* em polígonos ortogonais bidimensionais [Worman & Keil, 2007]. Outros trabalhos focaram na decomposição eficiente de polígonos planares buscando coberturas de área completa [Amit et al., 2007; Couto et al., 2008]. Para ambientes tridimensionais destacam-se os trabalhos de Marengoni et al. [2000] aplicável em terrenos planares e o de Viglietta [2012], que em sua tese de doutorado calcula a cobertura por arestas em volumes ortogonais não planares. Apesar de existir uma vasta pesquisa nessa área, uma versão sem restrições desse problema persistia em aberto.

1.1 Motivações

Além da motivação criada pela possibilidade de preencher uma lacuna no estado da arte, existe uma demanda pela análise mais eficiente dos mapas em jogos de tiro com o intuito de aprimorar o posicionamento de jogadores.

Existem diversas competições de jogos FPS e, antes das partidas, as equipes estudam os cenários para definir o posicionamento estratégico em diferentes situações de jogo. Em equipes avançadas, essas informações são comumente compartilhadas por meio de um software chamado *Advanced Tactical Center* (ATC), aplicável a diversos jogos de tiro.

A Figura 1.2 exemplifica as instruções de posicionamento de quatro jogadores para um mapa do jogo Battlefield 2. Essas estratégias são montadas a partir de experiências de jogo, sem o auxílio de qualquer análise computacional.

Encontrar uma solução aproximada para o PGA em ambientes 3D arbitrários, ou seja, sem padrões predefinidos, seria útil para diversas áreas, tais como jogos digitais, segurança, projetos de iluminação e finalidades militares.

1.2 Objetivo

O objetivo principal desse trabalho é encontrar soluções aproximadas para o Problema da Galeria de Arte em ambientes tridimensionais não planares e não ortogonais, ou

[1]Na geometria, um polígono P no plano é denominado monótono em relação a uma reta L se cada linha ortogonal a L intercepta P, no máximo, duas vezes.

Figura 1.2. Exemplo de posicionamento estratégico utilizando o software ATC.

seja, ambientes arbitrários complexos como os presentes em jogos digitais e no mundo real. Dessa forma, buscamos identificar, de modo aproximado, a quantidade mínima e a posição de guardas necessários para monitorar um ambiente complexo sem restrições.

Para isso foi utilizada uma abordagem dividida em três etapas. A primeira utiliza técnicas de amostragem de pontos em volumes ou superfícies, a segunda gera um grafo de visibilidade entre as amostras e a terceira e última etapa busca, por meio de uma abordagem genética, um conjunto mínimo de vértices capaz de cobrir todas as amostras. Assim, a metodologia se aplica a qualquer instância do problema.

1.3 Contribuições

Utilizando cenários complexos de jogos digitais 3D, mais especificamente mapas do jogo *Counter-Strike*, diversos experimentos foram realizados e demonstrou-se que a abordagem genética evolui para soluções com cobertura total de amostras do espaço e conjuntos reduzidos de guardas observadores.

A metodologia proposta não depende de restrições no ambiente, portanto, consiste em uma abordagem genérica aplicável tanto em ambientes simples quanto em ambientes arbitrários complexos que até então não haviam sido explorados.

Os resultados desse estudo geraram uma publicação no Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGAMES) [Fantini & Chaimowicz, 2013]. Além disso, um levantamento sobre plataformas de diversos gêneros de jogos aplicáveis em pesquisas (Apêndice A) foi publicado na *International Conference on Computer Games* (CGAMES) [Machado et al., 2011a] e um tutorial contendo uma lista estendida de

plataformas foi apresentado no SBGAMES [Machado et al., 2011b].

1.4 Organização do Trabalho

A dissertação é organizada da seguinte forma: no Capítulo 2 são apresentadas as metodologias mais significativas do estado da arte para tratar instâncias variadas do Problema da Galeria de Arte. Já no Capítulo 3 a nova metodologia proposta é descrita em detalhes. Os experimentos e a análise dos resultados são mostrados no Capítulo 4 e, por fim, o Capítulo 5 traz as conclusões da pesquisa e sugestões para trabalhos futuros.

Capítulo 2

Trabalhos relacionados

Ao longo da pesquisa foi feito um levantamento dos principais trabalhos relacionados ao Problema da Galeria de Arte. Existe uma variedade grande de metodologias aplicadas na busca por soluções em instâncias distintas do problema, porém foi identificada a lacuna de uma metodologia generalista. Ou seja, até então não era possível aplicar nenhuma das metodologias do estado da arte em um ambiente tridimensional arbitrário não planar e não ortogonal. Para propor uma nova abordagem, foram estudadas as técnicas utilizadas em diversas publicações, sendo as mais relevantes apresentadas nesse capítulo.

2.1 Preliminares

Com o intuito de auxiliar o entendimento das abordagens do PGA, são apresentados nessa seção os principais conceitos presentes no escopo do problema. No campo da geometria computacional, um polígono ou volume ortogonal é aquele em que todos os ângulos da geometria possuem noventa graus. Já um ambiente planar é aquele que pode ser projetado em um plano sem que ocorram sobreposições de superfícies, ou seja, um ambiente com túneis subterrâneos não é planar. Os buracos são espaços vazios gerados em uma área ou volume a partir de algum obstáculo [Michael, 2009]. Na Figura 2.1 é possível observar exemplos de polígonos planares com características variadas.

De acordo com [O'Rourke, 1987; Urrutia, 2000; Michael, 2009], as características citadas acima classificam os ambientes no escopo do PGA, podendo torná-los extremamente complexos ou simples. Entre os ambientes mais simples destacam-se as áreas bidimensionais, representadas por polígonos simples ortogonais e sem buracos (Figura 2.1(a)). Já entre os mais complexos estão os ambientes tridimensionais não

planares, representados por volumes arbitrários não ortogonais, com buracos formados por obstáculos (Figura 2.2).

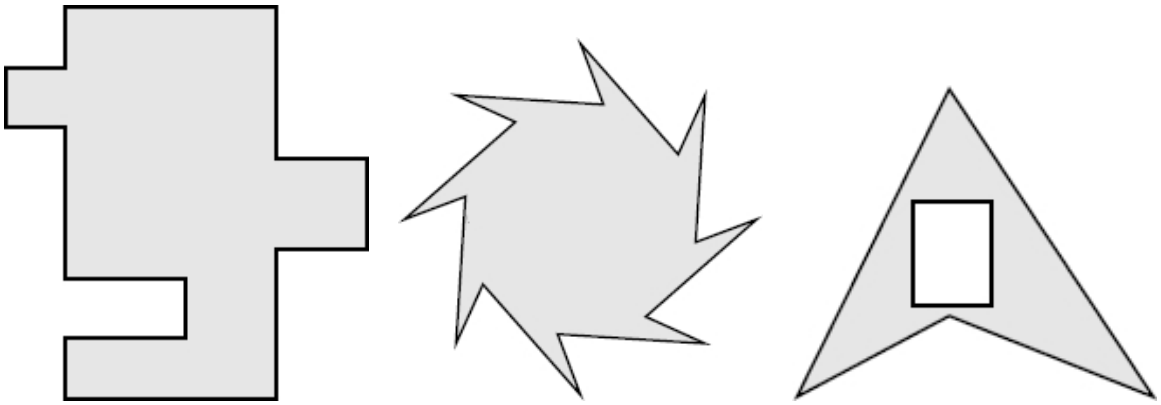


Figura 2.1. Exemplos de ambientes planares. Sendo (a) um polígono simples ortogonal, (b) um polígono não ortogonal e (c) um polígono não ortogonal com buraco.

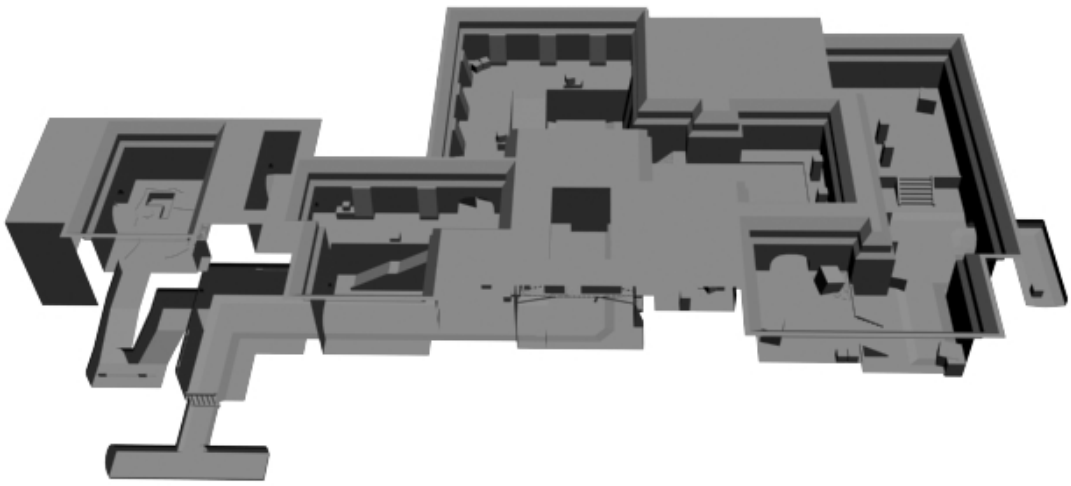


Figura 2.2. Exemplo de um ambiente 3D arbitrário complexo.

Uma solução para o PGA consiste em um conjunto mínimo de observadores/vértices com posições definidas capaz de monitorar um ambiente por completo. O conjunto de solução exata é denominado *Minimum Vertex Guard* (MVG), enquanto para o aproximado criamos o conceito *Approximated Minimum Vertex Guard* (AMVG). Em um modelo computacional, o conjunto de observadores também pode ser representado por arestas ao invés de vértices e recebe o nome de *Minimum Edge Guard* (MEG).

Ao modelar o ambiente na forma de um grafo ou na forma discreta, é comum utilizar o termo cobertura para tratar de monitoramento. Ou seja, dizer que um MVG cobre um ambiente significa que esse conjunto mínimo de guardas é capaz de monitorar a representação do ambiente. Na área de geometria computacional, um grafo de visibilidade é aquele formado por um conjunto de vértices pertencentes ao ambiente sendo que a visibilidade entre os mesmos é representada por uma aresta em comum. Ou seja, cada aresta indica que seus vértices adjacentes são visíveis entre si no espaço euclidiano.

2.2 Trabalhos relevantes

Existem diversos trabalhos relacionados ao PGA e suas instâncias. Nessa seção é apresentado o resumo de diversas metodologias já propostas, as quais apresentam técnicas e escopos de aplicação variados.

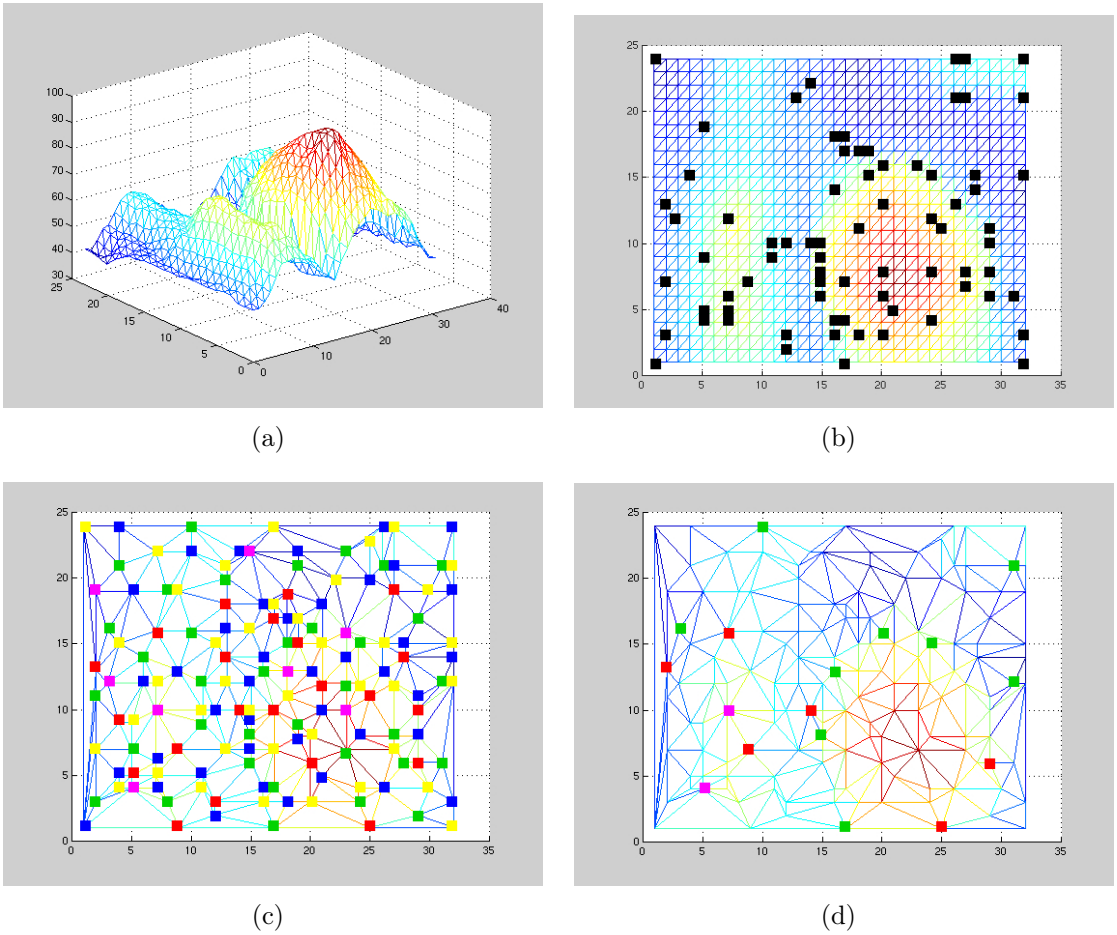
Em 1987, Ghosh propôs algoritmos de aproximação para os problemas de MVG e MEG aplicados em polígonos simples bidimensionais com e sem buracos. A técnica principal de sua metodologia foi a transformação do PGA no problema *Set Covering Problem* (SCP), no qual busca-se um conjunto de vértices de um grafo que cobre os demais. Nessa transformação foram gerados grafos de visibilidade de polígonos simples, um tipo especial de grafo de visibilidade que deve conter, obrigatoriamente, um ciclo hamiltoniano entre os vértices que constituem o polígono. Nessa abordagem os guardas são posicionados apenas nos vértices das bordas do polígono.

Por algumas décadas essa foi a única metodologia disponível para o PGA aplicável em polígonos bidimensionais com e sem buracos.

Marengoni et al. [2000] propuseram uma metodologia na qual um terreno topográfico é discretizado em um modelo digital denominado *Digital Elevation Map* (DEM). A unidade elementar dessa discretização é um poliedro, sendo que os guardas podem se posicionar sobre os vértices desses polígonos. Um terreno topográfico é considerado um ambiente tridimensional, porém a metodologia em questão é aplicável apenas em superfícies que podem ser projetadas em um plano bidimensional sem sobreposição, ou seja, não se aplica a ambientes tridimensionais não planares. Como a superfície é discretizada em poliedros, a metodologia permite a representação de formas não ortogonais e também buracos.

A metodologia de Marengoni et al. [2000] divide-se em três etapas. Na primeira etapa, a DEM é transformada em uma hierarquia de terreno (Figura 2.3(a)). Combinando dados de resoluções diferentes resulta em um mapa (Figura 2.3(b)) com um

nível de detalhe variável, que podem melhorar a qualidade do modelo de terreno em áreas designadas. Na segunda etapa, apenas a visibilidade local é considerada e um algoritmo 5-coloração é usado para posicionar um conjunto de observadores no terreno (Figura 2.3(c)). A terceira etapa remove a restrição de visibilidade local e calcula mapas globais de visibilidade para um subconjunto de observadores selecionados a partir do segundo passo. O autor então usa mapas visibilidade global e uma abordagem gulosa para o problema do conjunto de cobertura para reduzir o número de observadores no terreno (Figura 2.3(d)).



(a) (b)

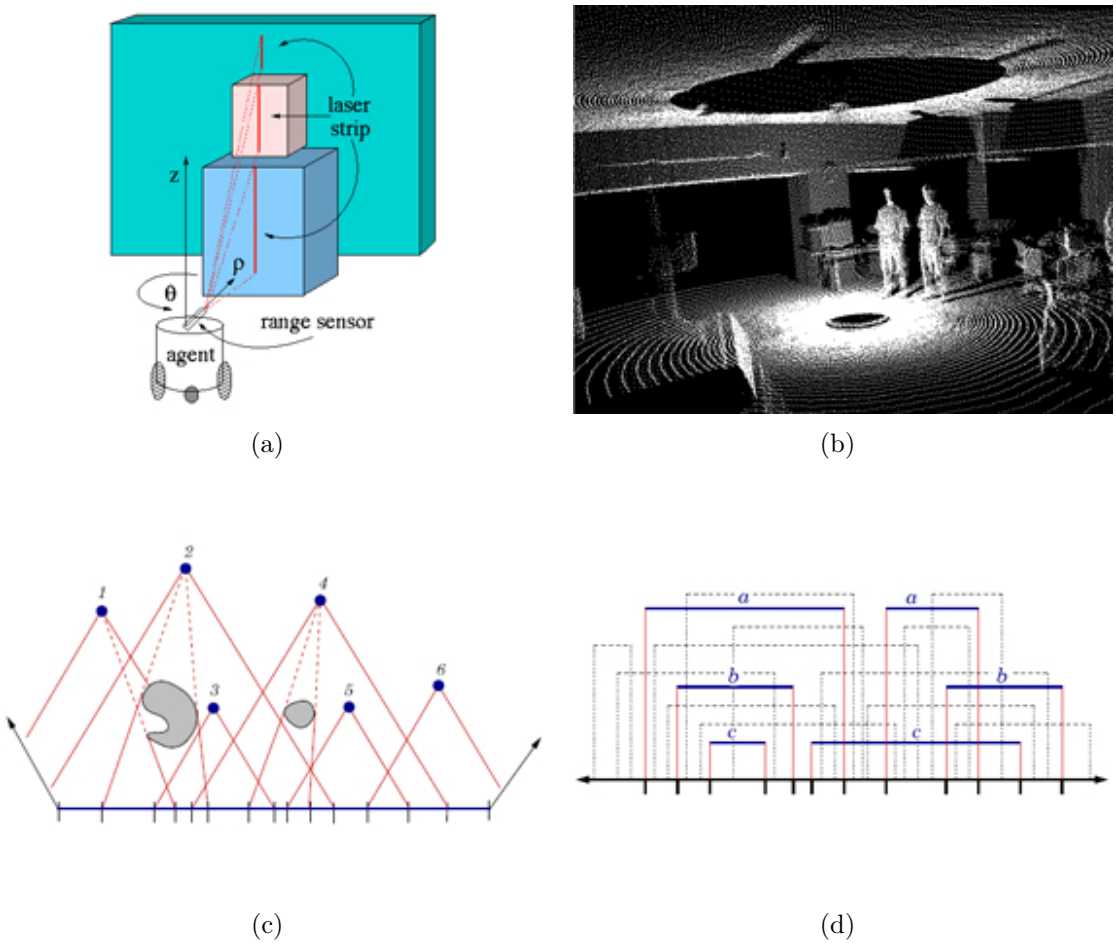
(c) (d)

Figura 2.3. Metodologia de Marengoni et al. [2000] para o Minimum Vertex Guard.

González-Banos [2001] investigaram algoritmos aproximados para calcular o posicionamento de um conjunto de observadores em um modelo poligonal bidimensional mapeado a partir de um ambiente 3D. Esse mapeamento foi feito a partir de sensores a laser fixados em um robô e, assim, uma representação planar do ambiente é criada. Essa metodologia pode representar ambientes ortogonais, não ortogonais, com e sem buracos, mas não ambientes não planares.

A partir da representação 2D, uma série de algoritmos aproximados foram investigados, com destaque para procedimentos de subdivisão e decomposição do espaço. Além disso, amostras de pontos e a transformação do problema da galeria de arte no problema do conjunto dominante de áreas indicaram um caminho promissor. Os autores concluíram que o mapeamento do ambiente 3D em uma representação 2D apresenta falhas e melhorias devem ser feitas.

A Figura 2.4(a) exemplifica o modelo de um robô com um scanner a laser giratório, utilizado para mapear o ambiente 3D. Já a Figura 2.4(b) mostra um espaço real 3D com obstáculos, onde as partes claras indicam o que foi mapeado e as sombras indicam os espaços ocultos. As Figuras 2.4(c) e 2.4(d) são exemplos de algoritmos de subdivisão e cobertura 2D utilizados pelos autores, baseados em triângulos e quadrados respectivamente. As manchas cinzas na Figura 2.4(c) representam os obstáculos.



(a) (b)

(c) (d)

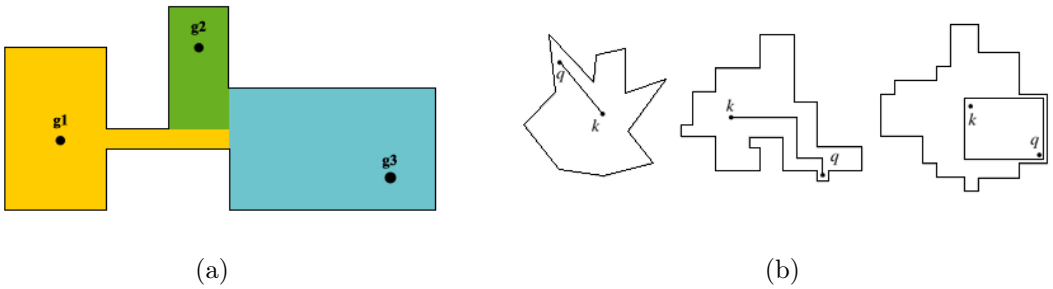
Figura 2.4. Metodologia de González-Banos [2001].

Efrat & Har-peled [2006] apresentaram uma abordagem baseada na discretização do ambiente planar em um grid denso, sendo ele ortogonal ou não, com buracos ou não.

A metodologia é similar à de Marengoni et al. [2000], porém os vértices do conjunto MVG podem ser posicionados em qualquer local do grid, e não apenas em seus vértices. Respeitando as restrições do ambiente citadas acima, o algoritmo de Efrat & Har-peled [2006] foi um dos primeiros no estado da arte a apresentar soluções exatas para algumas instâncias do PGA e sua aplicação se estende a ambientes tridimensionais ortogonais planares.

Worman & Keil [2007] trabalharam na decomposição mínima de ambientes bidimensionais ortogonais, ou seja, em sua metodologia buscaram a quantidade mínima de subdivisões de um polígono, sendo que cada subpolígono tinha sua área coberta por um único ponto interno [Figura 2.5(a)]. Foram considerados três tipos de subdivisões possíveis, chamadas de *star-shaped* ou *star polygons*. Essas subdivisões foram criadas a partir de um teste de visibilidade entre pontos de amostra, q e k, dentro do polígono, as quais podem ser de três tipos: *l-star*, *s-star* e *r-star* (Figura 2.5(b)).

Encontrando o número mínimo de subpolígonos *r-star* totalmente observáveis e sem áreas de interseção, encontra-se também o MVG. Foi demonstrado que esse processo de subdivisão em polígonos *r-star* a partir de um polígono ortogonal tem um custo de processamento polinomial.



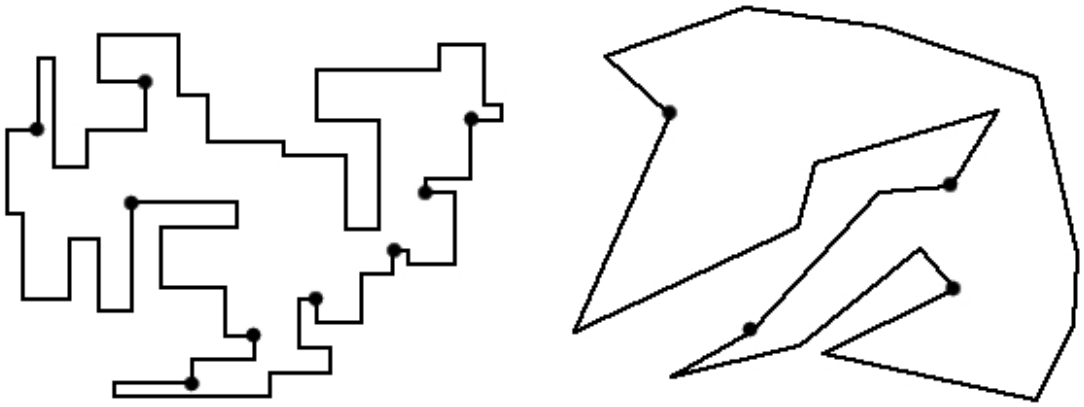
(a) (b)

Figura 2.5. Exemplo de três guardas monitorando três subpolígonos r-star (a). Exemplos de decomposição de acordo com a visibilidade entre duas amostras de pontos q e k: l-star, s-star e r-star da esquerda pra direita (b).

Bajuelos et al. [2008a, 2008b] criaram uma abordagem generalista baseada em algoritmos genéticos, ou seja, uma meta-heurística. Segundo os autores, a complexidade NP-Difícil do PGA permite duas linhas de pesquisa: a elaboração de algoritmos aproximados generalistas ou o desenvolvimento de algoritmos exatos para instâncias muito simples do problema.

Eles trabalharam com ambientes exclusivamente bidimensionais ortogonais [2008a] e bidimensionais não ortogonais [2008b], porém sem buracos [Figura 2.6]. Além disso, o posicionamento dos guardas era limitado aos vértices das bordas do polígono,

sendo a quantidade necessária e posições dos guardas definidas por um algoritmo genético. As soluções candidatas foram avaliadas a partir de uma função de *fitness*. Essa função considera a maximização da área coberta e do número de vértices cobertos, além da minimização do número de guardas e de interseções entre áreas monitoradas.



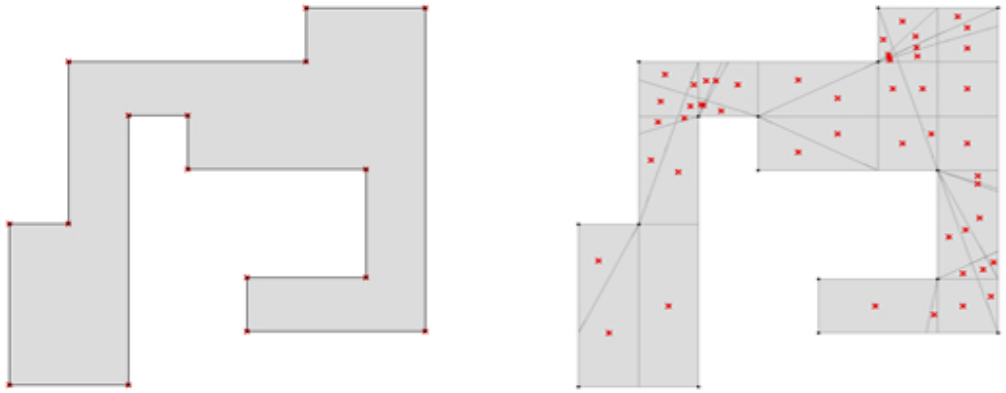
(a) (b)

Figura 2.6. Um polígono ortogonal (a) e outro não ortogonal (b), ambos sem buracos e com guardas posicionados nos vértices da borda.

Couto et al. [2008] abordaram uma metodologia baseada na triangularização de ambientes planares sem buracos (Figura 2.7(a)). Esses triângulos foram chamados de *Atomic Visibility Polygons* (AVP). Após a discretização do polígono, são gerados pontos nos centros dos AVPs (Figura 2.7(b)). Esses pontos, chamados de centroides, são usados para calcular o conjunto de cobertura por meio de um algoritmo polinomial. Esse conjunto é constituído por vértices suficientes para cobrir todos os triângulos, ou seja, toda a área do polígono.

Diferentemente das demais metodologias que buscam o MVG, os autores tratam o PGA como um problema de decisão, buscando um conjunto suficiente, ao invés do conjunto mínimo, para cobertura de áreas em instâncias específicas do problema. Porém, é demonstrado que a metodologia garante uma cobertura total da área com poucas interseções de visibilidade, o que reduz o tamanho dos conjuntos.

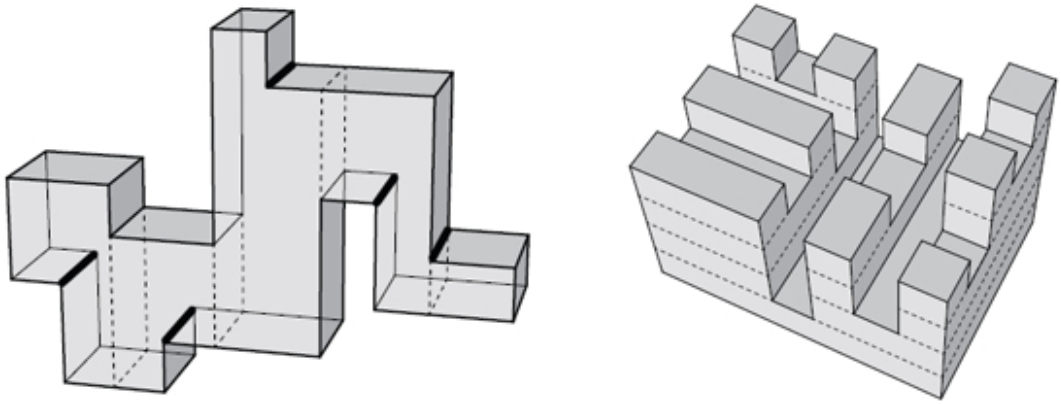
Em sua tese de doutorado, Viglietta [2012] demonstrou diversos teoremas do estado da arte descritos em [O'Rourke, 1987; Urrutia, 2000; Michael, 2009]. Além disso, para ambientes ortogonais, utilizou uma técnica chamada *reflex edges* de duas direções (*2-reflex polyhedra*), onde a cobertura de blocos de volumes ortogonais é feita por arestas adjacentes (Figura 2.8(a)). A *reflex edge* de um poliedro pode ser definida como uma aresta em que o ângulo diedro interno subtendido por duas faces incidentes é maior do que 180 graus.



(a) (b)

Figura 2.7. Exemplo de um polígono simples ortogonal sem buracos (a) e a triangularização correspondente, destacando-se os centroides de cada subpolígono (b).

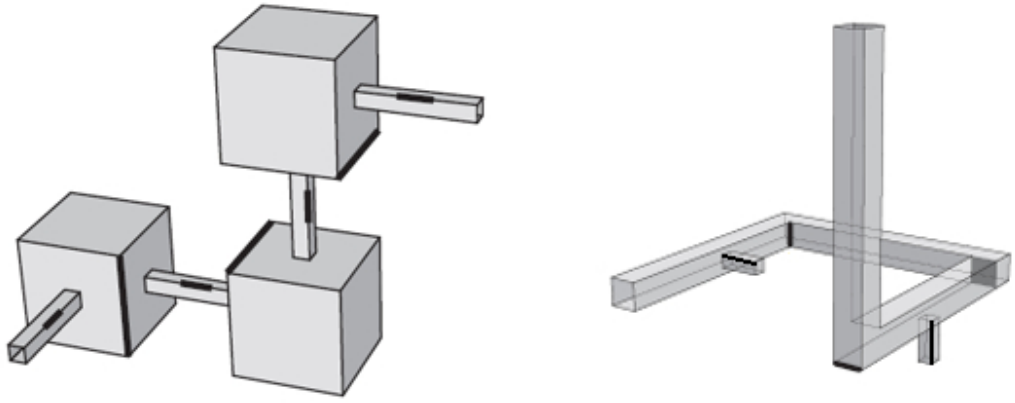
Para ambientes 3D ortogonais, planares ou não, o volume é representado por um conjunto de blocos chamados *bricks* (Figura 2.8(b)). Analisando esse conjunto de blocos, um algoritmo determina quais arestas das junções fazem parte de um conjunto de arestas observadoras, ou seja, o MEG aproximado (Figuras 2.9(a) e 2.9(b)).



(a) (b)

Figura 2.8. Em (a) estão destacadas as reflex edges, as quais indicam o posicionamento de guardas para ambientes ortogonais. Em (b) destacam-se em pontilhado as arestas de junções dos blocos de volumes.

Baseado em diversos experimentos, o autor demonstrou limites superiores para a quantidade de guardas/arestas suficientes para a cobertura de ambientes ortogonais com poucas restrições. Provou-se também que para averiguar se um volume composto por poliedros é completamente coberto por um conjunto mínimo de guardas/arestas é



(a) (b)

Figura 2.9. Exemplos de coberturas aproximadas por arestas utilizando reflex edges aplicadas a ambientes ortogonais não planares.

um problema NP-Difícil.

Uma técnica alternativa explorada foi o corte de um volume poliedro com planos paralelos, a fim de reduzir um problema 3D a um conjunto de problemas 2D. O principal desafio ocorre no momento de mesclar o conjunto de soluções 2D em uma solução 3D eficiente. Outras técnicas foram testadas, buscando limites superiores para a quantidade de arestas necessárias para cobrir volumes não ortogonais, porém sem sucesso para definir uma regra geral.

Ficaram em aberto os limites inferiores para a quantidade de guardas necessária para cobrir poliedros ortogonais, bem como limites inferiores e superiores para poliedros complexos. Porém, como citado anteriormente, há uma contribuição para a cobertura aproximada de ambientes não planares ortogonais por meio de *reflex edges*, outra lacuna do estado da arte anterior ao seu trabalho.

Inspirando-se em algumas ideias dos trabalhos relacionados, nossa metodologia avançou com técnicas adicionais buscando suprir a lacuna do estado da arte para soluções aproximadas em ambientes tridimensionais genéricos, ou seja, sem restrições [Fantini & Chaimowicz, 2013].

Como será apresentada em detalhes nos capítulos seguintes, nossa metodologia possui três etapas distintas. A primeira gera pontos de amostras no ambiente, a segunda cria um grafo de visibilidade entre as amostras e a terceira busca, por meio de uma meta-heurística genética, o MSC no grafo. Os resultados mostraram que as soluções candidatas convergem para uma cobertura total das amostras e para AMVGs reduzidos.

Na Tabela 2.1 é apresentado um resumo do estado da arte e da contribuição de nossa metodologia.

Tabela 2.1. Aplicabilidade das principais metodologias presentes no estado da arte.

Referência	*Ambientes 2D*		*Ambientes 3D*		
	Não ortogonais	*Com buracos*	*Não ortogonais*	*Com buracos*	*Não planares*
[Ghosh, 1987]	sim	sim	não	não	não
[Marengoni et al., 2000]	sim	sim	sim	sim	não
[González-Banos, 2001]	sim	sim	sim	sim	não
[Efrat & Har-peled, 2006]	sim	sim	sim	sim	não
[Worman & Keil, 2007]	não	sim	não	não	não
[Bajuelos et al., 2008b]	sim	não	não	não	não
[Couto et al., 2008]	sim	não	sim	não	não
[Viglietta, 2012]	sim	sim	não	sim	sim
[Fantini & Chaimowicz, 2013]	sim	sim	sim	sim	sim

Capítulo 3

Metodologia

Como mostrado no capítulo anterior, os trabalhos sobre o problema da galeria de arte abordam técnicas variadas e são aplicadas em instâncias restritas do mesmo. Com inspiração nesses trabalhos e em processos de computação gráfica foi formulada uma nova metodologia para tratar o PGA. O objetivo foi criar uma metodologia generalista, ou seja, aplicável a qualquer instância do problema, uma lacuna que permanecia no estado da arte até então.

Considere Q um volume delimitado por paredes, obstáculos e túneis não ortogonais de um ambiente tridimensional complexo. Sendo o objetivo monitorar o volume Q, cada ponto dentro do volume deve ser visível por pelo menos um observador. Transformemos, então, o volume Q em uma nuvem com P pontos. Se P tende ao infinito e garantimos que todos os pontos são cobertos por ao menos um elemento do conjunto de guardas, temos uma cobertura completa do ambiente. Se o tamanho do conjunto de guardas V for o menor possível, então V é o MVG, uma solução exata para o PGA.

Na metodologia desse trabalho, uma nuvem de pontos P é gerada por meio de uma amostragem de pontos. Em alguns casos, dependendo da aplicação, não é necessário monitorar 100% do volume do ambiente, mas o suficiente para impedir que algo ou alguém se esconda entre as amostras.

Após a aplicação de um método de amostragem apropriado (Figuras 3.1 e 3.2), transforma-se o conjunto de amostras em um grafo de visibilidade (Figura 3.3), estabelecendo quais pontos são visíveis entre si no volume a ser monitorado. Esse grafo é usado como entrada para a última etapa da metodologia, a qual busca a quantidade mínima e o posicionamento de guardas que cobrem as amostras do ambiente (Figuras 3.4 e 3.5). Encontrar o MVG é um problema de otimização NP-Difícil. Para contornar esse obstáculo foram utilizados algoritmos genéticos para buscar soluções aproximadas e, assim, viabilizar computacionalmente a metodologia. Mais detalhes sobre essa

metodologia serão apresentados nas próximas seções desse capítulo.

Figura 3.1. Passo 1a da metodologia. Amostras de pontos são geradas em um ambiente tridimensional arbitrário.

Figura 3.2. Passo 1b da metodologia. As amostras são convertidas em vértices para o grafo de visibilidade.

Figura 3.3. Passo 2 da metodologia. O grafo de visibilidade é gerado a partir dos vértices de amostra e as arestas indicam quais pontos são visíveis entre si no ambiente tridimensional.

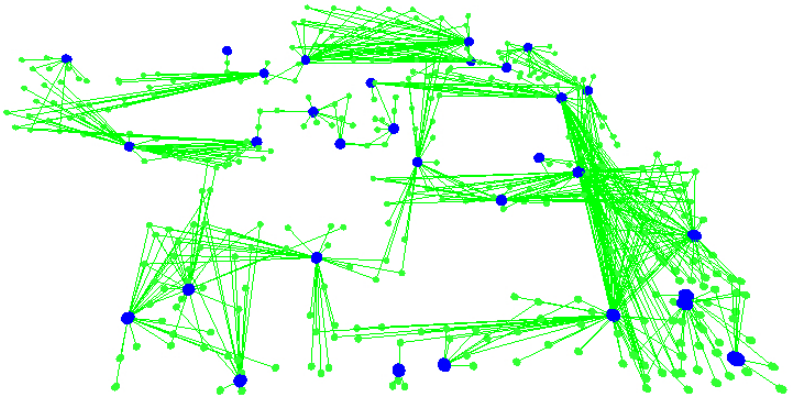


Figura 3.4. Passo 3a da metodologia. Calcula-se o MVG do grafo de visibilidade, atribuindo a alguns vértices a função de guardas do ambiente (vértices azuis).

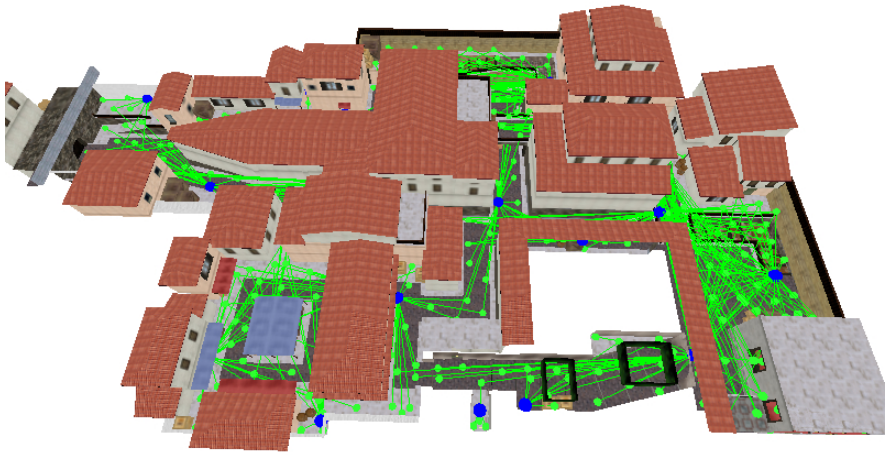


Figura 3.5. Passo 3b da metodologia. O MVG (vértices azuis) retorna ao ambiente original para marcar s posições dos guardas no espaço.

3.1 Primeiro passo: Amostragem de pontos

Na computação gráfica, a amostragem de pontos também conhecida como superamostragem é um processo largamente utilizado na geração de imagens, mais especificamente em técnicas de *antialiasing*, incrementando a qualidade de imagens. As superamostragens também são importantes para transformar um espaço geométrico 3D em uma imagem 2D a ser exibida na tela, um processo denominado *rasterization* [Suffern, 2007]. Existem diversas técnicas de amostragem, dentre as quais podemos destacar as mais comuns: amostragem uniforme, amostragem uniforme aleatória, amostragem aleatória (*jitter*) e distribuição de Poisson [Goss & Wu, 2000].

A Figura 3.6 exemplifica essas quatro técnicas clássicas. A amostragem uniforme gera um grid em todo o ambiente e cria uma amostra no centro de cada subdivisão (Figura 3.6(a)). A amostragem uniforme aleatória cria amostras em posições aleatórias dentro de cada subdivisão do *grid* (Figura 3.6(b)). Já a amostragem aleatória cria amostras em qualquer lugar do espaço, independentemente do *grid* (Figura 3.6(c)). Por fim, a distribuição de Poisson cria amostras aleatórias no espaço respeitando uma distância mínima entre elas, ajustada por meio dos raios dos discos, também conhecidos como discos de Poisson (Figura 3.6(d)) [Goss & Wu, 2000].

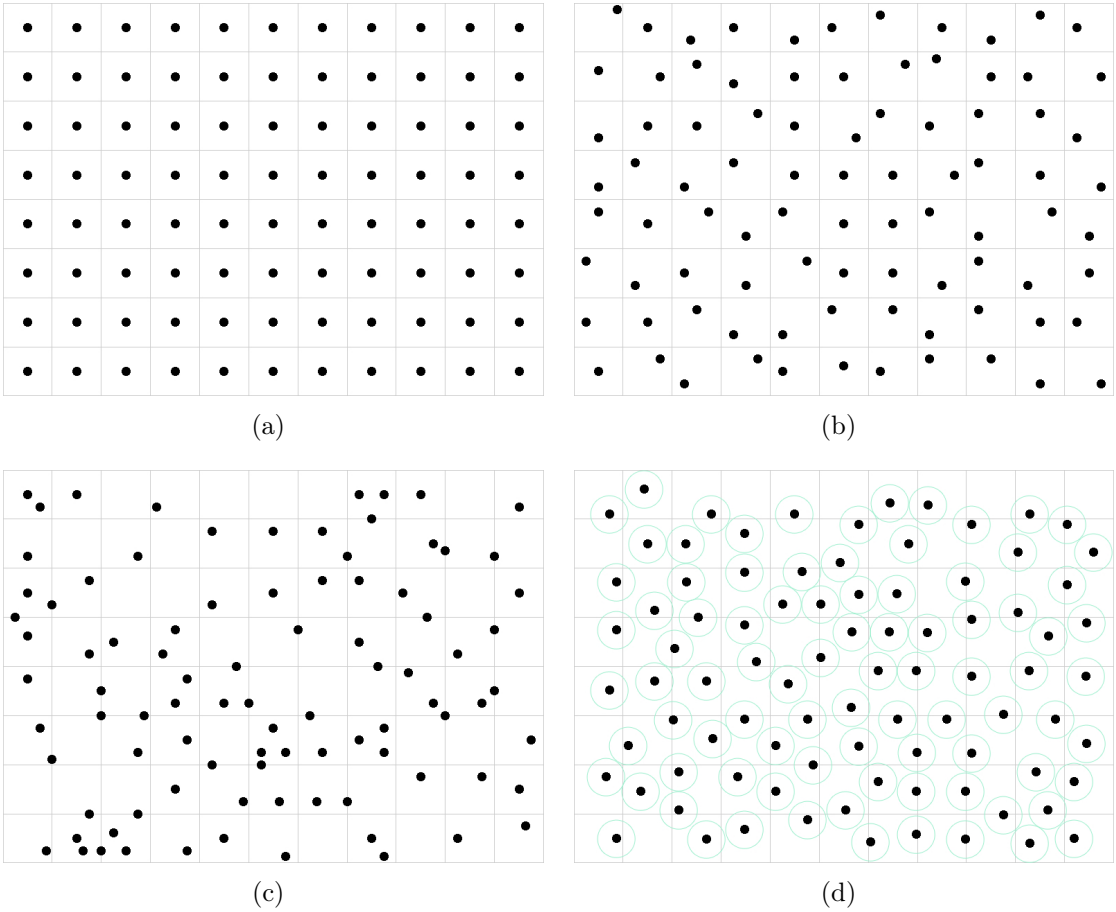


Figura 3.6. Exemplos de métodos de amostragem clássicos. Amostragem uniforme (a), amostragem uniforme aleatória (b), amostragem aleatória (c) e distribuição de Poisson (d).

Nesse trabalho foram exploradas duas delas, a distribuição de Poisson e a amostragem uniforme. A primeira por ser utilizada na geração de *waypoints* do jogo *Counter-Strike* e a segunda por ser largamente conhecida e simples de aplicar.

Os *waypoints* são vértices de um grafo de mobilidade do jogo *Counter-Strike* e podem ser gerados automaticamente enquanto jogadores reais se movem pelo mapa. Caso o jogador ocupe uma coordenada que se encontra a pelo menos 200 pixels dos demais *waypoints*, um novo ponto é criado. Ou seja, essa é a medida dos raios dos discos de Poisson nesse processo.

Já a amostragem uniforme, comumente aplicada em planos bidimensionais, foi adaptada para três dimensões considerando um grid 3D ao invés de um *grid* 2D. Sendo assim a denominamos amostragem uniforme por volume.

Esses conjuntos de amostras equivalem aos vértices do grafo de visibilidade e são fundamentais, pois a representação de um ambiente na forma de um grafo é que

permite a metodologia ser generalista. Além disso, em nossa metodologia, os guardas podem se posicionar apenas nas amostras.

3.1.1 Distribuição de Poisson (waypoints)

Waypoints podem ser definidos como pontos de referência no espaço físico utilizados para as finalidades de monitorar e controlar os movimentos de agentes. Na área de inteligência artificial aplicada a jogos, o uso de *waypoints* é comum em ambientes contínuos para planejar trajetórias de agentes virtuais (*bots*), além de armazenarem informações úteis para a tomada de decisão desses agentes [Millington & Funge, 2009].

A Figura 3.7 exemplifica os *waypoints* originais de um mapa do *Counter-Strike*. As barras verticais são as marcações desses vértices, sendo que algumas características definem a sua cor. Cada *waypoint* possui um conjunto de informações (texto na Figura 3.7) que são processadas pelos *bots*.

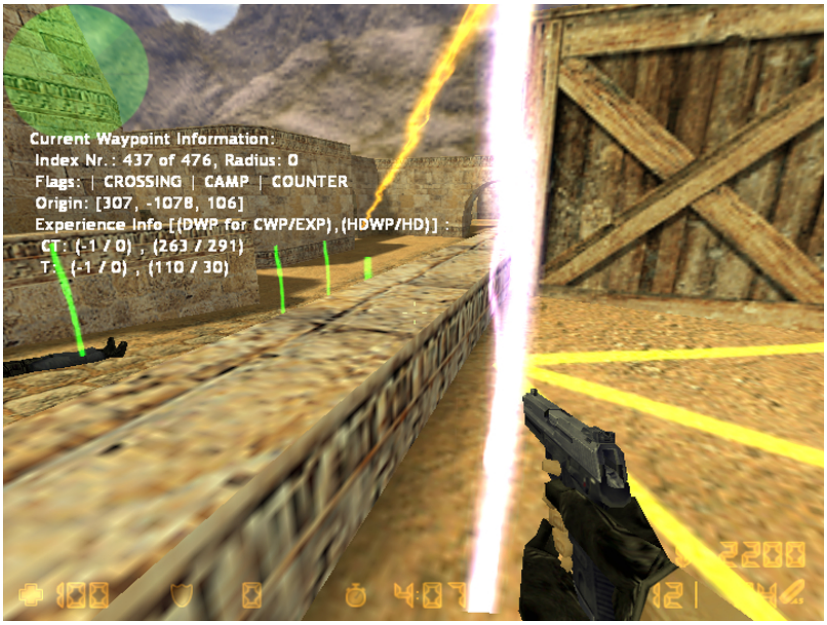


Figura 3.7. Editor de waypoints (barras verticais) do Counter-Strike. Detalhe para as informações do waypoint.

Como essa amostragem de vértices apresenta uma distribuição interessante e está disponível em todos os mapas do jogo, decidiu-se reaproveitá-las em alguns experimentos. A Figura 3.8 apresenta um conjunto de *waypoints* de um mapa do jogo distribuídos no espaço.

Cada mapa possui seu próprio conjunto de *waypoints* que pode ser extraído pelo motor do jogo (Half-Life 1 SDK). Cada vértice extraído possui uma coordenada no

espaço tridimensional e um índice que o identifica no jogo. Esses vértices são utilizados para criar o grafo de visibilidade no próximo passo da metodologia.

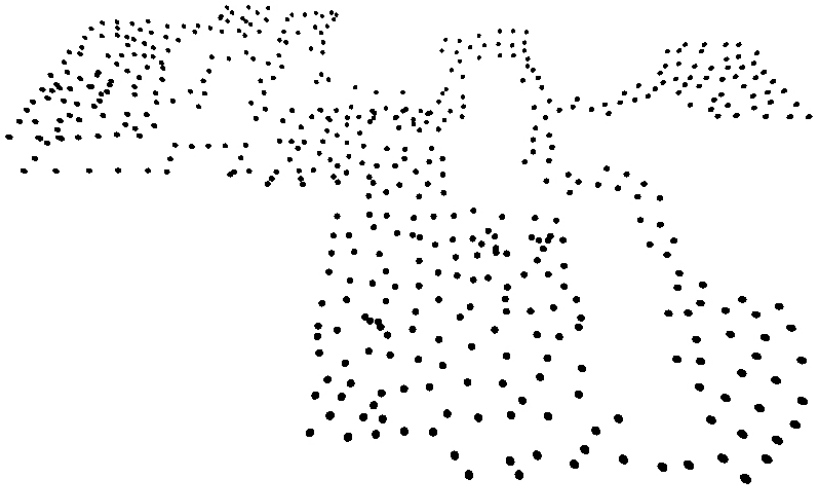
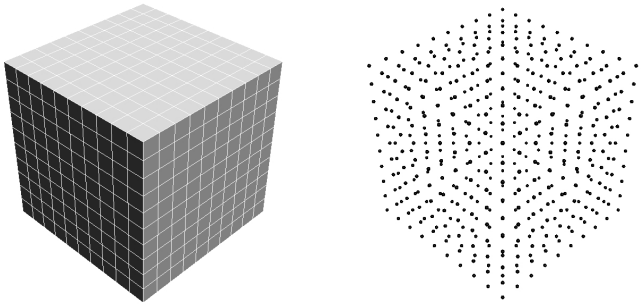


Figura 3.8. Distribuição dos waypoints no mapa De_airstrip do Counter-Strike.

3.1.2 Amostragem uniforme por volume

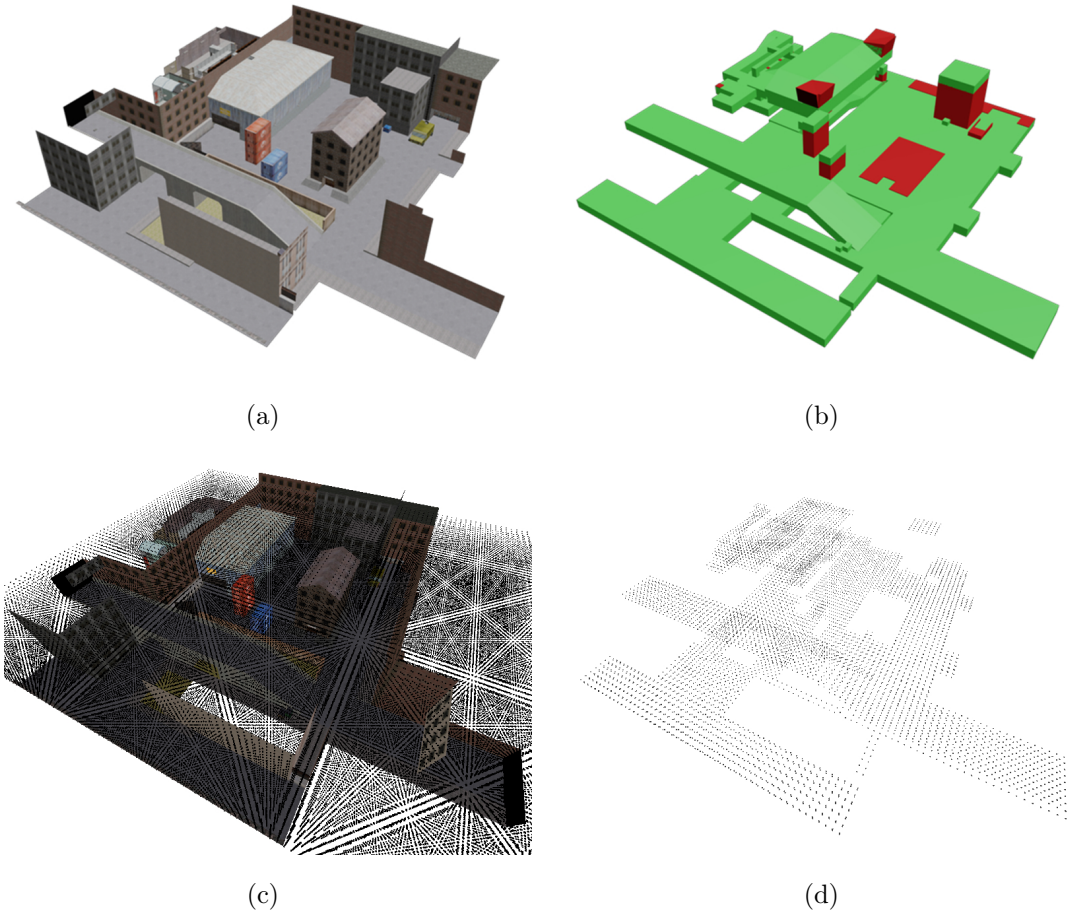
A amostragem uniforme por volume é baseada na clássica amostragem uniforme, porém, o ambiente é discretizado por um *grid* 3D (Figura 3.9(a)). Em seguida são criadas amostras no centro de cada subdivisão do *grid*, resultando num conjunto de vértices equidistantes nos eixos X, Y e Z (Figura 3.9(b)). A distância entre as amostras pode ser ajustada de acordo com a necessidade da cobertura.



(a) (b)

Figura 3.9. Exemplo de amostragem uniforme por volume.

Como a amostragem uniforme gera muitos pontos, eles podem ser otimizados dependendo do objetivo da aplicação. Por exemplo, se quisermos posicionar guardas com o intuito de monitorar a presença de jogadores em um determinado cenário de jogo, não é necessário cobrir amostras em locais inacessíveis. Pensando nisso foi criado o conceito de zonas de interesse para esse trabalho.



(a) (b)

(c) (d)

Figura 3.10. Etapas para a validação de amostras. A partir de um ambiente complexo (a) são criadas zonas verdes e vermelhas (b) que resultam na zona de interesse. Após a amostragem uniforme (c), apenas os vértices que pertencem à zona de interesse são considerados válidos (d).

Dado um ambiente tridimensional arbitrário complexo (Figura 3.10(a)), primeiramente são criados volumes em regiões ocupáveis do mapa, desconsiderando obstáculos, as quais receberam o nome de zonas verdes. Já os obstáculos são representados separadamente por volumes denominados zonas vermelhas (Figura 3.10(b)). Em seguida, realizando uma operação booleana de subtração entre as zonas verdes e zonas vermelhas, obtém-se um volume resultante, o qual é denominado zona de interesse. Após a

amostragem uniforme por volume (Figura 3.10(c)), os vértices são considerados válidos apenas se estiverem dentro da zona de interesse e dessa forma temos a amostragem otimizada correspondente (Figura 3.10(d)).

Nesse trabalho optou-se por criar zonas verdes e zonas vermelhas manualmente por meio de um *software* de modelagem, o *Autodesk 3D Max*. Porém, é possível desenvolver métodos automáticos para fazê-lo.

É importante observar que a zona de interesse preserva as características arbitrárias complexas do ambiente original, uma vez que as zonas verdes e vermelhas podem ser representadas por paralelepípedos, prismas e esferas redimensionáveis e as operações booleanas preservam e, em algum casos, aumenta a complexidade do volume resultante.

A partir da zona de interesse resultante é possível filtrar as amostras para a próxima etapa da metodologia. Um teste de geometria simples verifica se cada amostra pertence ao volume de interesse. Essas amostras válidas são, então, utilizadas como vértices do grafo de visibilidade.

3.2 Segundo passo: Criação do grafo de visibilidade

O grafo de visibilidade é formado por vértices amostras de pontos da primeira etapa da metodologia e arestas, as quais indicam que seus vértices adjacentes são visíveis entre si.

O segundo passo da metodologia é a criação desse grafo usando uma técnica de detecção de colisão denominada teste de interseção raio-malha, comumente conhecida como *piking* quando o objetivo é selecionar um objeto no espaço 3D ou *hit-testing* quando tratamos de testes colisão de projéteis [Foley et al., 1995; van den Bergen, 2003].

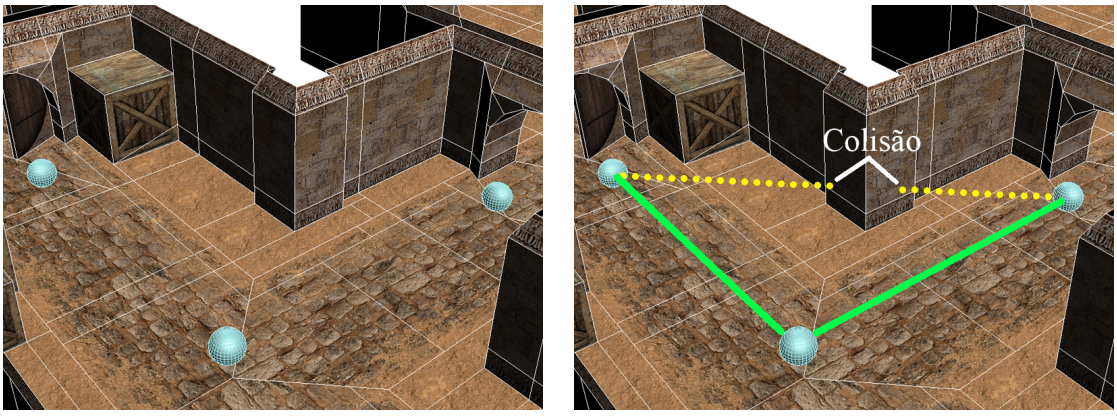
3.2.1 Teste de interseção raio-malha

Essa técnica recebe diversos nomes na literatura (*Ray-triangle test*, *Ray-polygon test*, etc.) e também nas implementações dos motores de jogos (*Trace-line test*, *Collision point from ray*, etc.). Basicamente consiste de um cálculo matemático para testar se uma linha intercepta um polígono de uma determinada malha geométrica [van den Bergen, 2003].

Para criar o grafo de visibilidade é gerado um raio entre cada par de amostras. Caso o raio colida com alguma malha da geometria do ambiente, significa que há um obstáculo entre elas e que não são intervisíveis. Porém, se não houver colisão alguma

com o raio, significa que o espaço está livre entre as amostras e elas são visíveis entre si (Figura 3.11).

É importante destacar que, no contexto do PGA, considera-se que um guarda tem visão de 360 graus. Essa característica foi considerada em nossa metodologia e em todos os nossos experimentos.



(a) (b)

Figura 3.11. Exemplo do funcionamento do hit-testing. Dado um conjunto de amostras em um ambiente 3D (a), são gerados raios entre cada par de arestas. Se houver uma colisão do raio (pontilhado amarelo) com o cenário, as amostras não são intervisíveis. Caso contrário, elas são intervisíveis e uma aresta (verde) é criada (b).

Decidimos utilizar essa técnica por ser computacionalmente viável e uma das formas mais simples de verificar se dois pontos são intervisíveis em um ambiente complexo. Além disso, é um método disponível nos principais motores de jogos. A complexidade desse procedimento leva em consideração todos os pares das N amostras e é dada pela Equação 3.1.

$$\theta\left(N(N-1)\right) \tag{3.1}$$

Para gerar o grafo de visibilidade com as amostras de *waypoints* do *Counter-Strike* foi utilizado o motor (Half-Life 1 SDK). Criou-se um novo comando de console para o jogo que utilizou o método nativo *TRACE_LINE* para gerar raios entre os *waypoints* de cada mapa. Esse método retorna uma variável que indica se o raio colidiu ou não com algum obstáculo. Dessa forma foram criadas as arestas de visibilidade e o grafo foi armazenado em uma matriz.

Já o grafo de visibilidade para as amostras do método uniforme por volume foi gerado por meio do método *CollisionPointFromRay* do motor gráfico *Irrlicht*. Foi desenvolvida uma aplicação em C++ que carrega uma malha poligonal do cenário e

gera uma amostragem uniforme de pontos sobre a mesma. Após o processo de validação de vértices das zonas de interesse, as arestas do grafo de visibilidade foram criadas e o mesmo foi armazenado em uma matriz. Após a criação dos grafos, algoritmos foram utilizados para buscar o MVG correspondente.

3.3 Terceiro passo: Busca pelo MVG

A terceira e última etapa da metodologia consiste na busca pelo MVG. Aplicado a um grafo, o MVG equivale ao problema de otimização *Minimum Set Cover* (MSC). O MSC consiste em um conjunto mínimo de vértices que cobre todos os demais em um grafo conexo, ou seja, há pelo menos uma aresta entre as amostras e algum elemento do conjunto MSC. Ambos são problemas NP-Difíceis.

Como os algoritmos exatos para problemas NP-Difíceis apresentam um comportamento assintótico exponencial para o tempo de execução, o cálculo de soluções é computacionalmente inviável a partir de determinada dimensão do problema. Para contornar esse obstáculo, foi proposta uma abordagem genética, a qual é capaz de fornecer soluções aproximadas em tempo satisfatório para problemas de maior escala.

Também foi desenvolvido um algoritmo exato, baseado no paradigma *backtracking*, para servir de *baseline* em comparações de resultados com o algoritmo genético.

3.3.1 Algoritmo exato

O algoritmo exato pode demandar várias iterações até encontrar um resultado. A primeira delas gera todos os conjuntos possíveis de guardas de tamanho igual a um e verifica se existe algum capaz de cobrir todos os demais vértices no grafo de visibilidade. A segunda iteração gera todos os conjuntos de tamanho igual a dois e assim por diante. Dessa forma, quando é encontrado um ou mais conjuntos de tamanho K que cobrem completamente o grafo, o algoritmo retorna essas soluções e finaliza. Esse conjunto mínimo que cobre todos os vértices é o MSC ou MVG.

Cada solução candidata é representada por um vetor binário contendo uma posição para cada vértice. Como exemplificado na Figura 3.12, quando uma posição tem o valor igual a 1 significa que o vértice correspondente pertence ao conjunto de guardas a ser testado.

1	0	1	0	0	0	0	0	1	0	...	0	0	0	0	1	0	0	0	0

Figura 3.12. Representação de uma solução candidata para o MVG ou MSC.

Esse algoritmo é sensível ao tamanho do conjunto exato de cobertura, ou seja, quanto maior for esse conjunto de guardas, mais iterações serão necessárias para concluir o processamento. Ao analisar a complexidade de tempo de execução do algoritmo verificamos que o seu comportamento assintótico é exponencial.

O algoritmo exato pode ser dividido em duas partes. Uma que gera as soluções candidatas possíveis (Algoritmo 3.1) e outra que testa se a solução candidata cobre todos os vértices do grafo (Algoritmo 3.2).

```
   entrada: adjMatrix, N; // matriz de adjacências e núm. de vértices
   saída  : MSCsolutions[]; // soluções exatas para o MSC
 1 for i ← 0 to N − 1 do // iterações
 2     if MSCsolutions.Size() ≠ 0 then // encontrou soluções?
 3         break; // condição de parada
 4     K ← i + 1; // guardas disponíveis
 5     positions.push(i); // guarda 1
 6     for j ← 1 to i do // mais guardas por iteração
 7         positions[j] ← j;
 8     if checkFullCoverage(adjMatrix, positions) = true then // alg. 3.2
 9         MSCsolutionsPush(positions); // armazena a solução
10     endflag ← false; // fim das combinações?
11     pivot ← K − 1; // pivô que cria as combinações
12     while endflag = false do
13         while positions[pivot] = (pivot + N − K) do // gera combinações
14             if pivot = 0 then
15                 endflag ← true;
16                 break;
17             else
18                 pivot ← pivot − 1;
19             if endflag = false then
20                 positions[pivot] ← positions[pivot] + 1;
21                 for m ← pivot + 1 to K − 1 do
22                     positions[m] ← positions[m − 1] + 1;
23                     if checkFullCoverage(adjMatrix, positions) = true then
24                         MSCsolutionsPush(positions);
25                         pivo ← K − 1;
26 Print(MSCsolutions[]);
```

Algoritmo 3.1: Algoritmo exato para calcular o MSC.

O Algoritmo 3.1 gera todas as formas distintas de indicar V guardas no conjunto de N vértices (Equação 3.2), ou seja, é uma combinação. No pior caso, o número

de guardas é igual ao número de vértices. Nos outros casos, são geradas todas as combinações até que o conjunto de guardas seja capaz de cobrir todos os vértices. De modo geral, podemos afirmar que as iterações do Algoritmo 3.1 vão de $V = 1$ até $V = K$, sendo K o número de guardas da solução exata (Equação 3.3).

$$C_V^N = \binom{N}{V} = \frac{N!}{V!\,(N-V)!} \tag{3.2}$$

$$\sum_{V=1}^{K} \frac{N!}{(N-V)!\;V!} \qquad para\ 1 \leq K \leq N \tag{3.3}$$

Para cada combinação distinta é feito um teste de cobertura que verifica se o conjunto com C guardas é capaz de cobrir todos os vértices (Algoritmo 3.2). Esse teste consiste em verificar todas as adjacências do conjunto de guardas por meio da matriz de adjacências. Logo, a complexidade do Algoritmo 3.2 é equivalente a VN.

```
   entrada: adjMatrix, N, positions; // matriz de adjacências, núm. de
            vértices e solução candidata
   saída  : true OU false; // a solução cobre todo o grafo ou não
 1 for i ← 0 to N − 1 do
 2 |  Covered[j] ← false; // reset do vetor de cobertura
 3 count ← 0;
 4 for i ← 0 to positionsSize − 1 do // laço para o número de guardas
 5 |  guard ← positions[i];
 6 |  if Covered[guard] = false then // marca apenas novos vértices
 7 |  |  Covered[guard] ← true; // guardas cobrem a si mesmos
 8 |  |  count ← count + 1; // incrementa cobertura
 9 |  for j ← 0 to N − 1 do // laço para demais vértices
10 |  |  if adjMatrix[guard][j] = true and Covered[j] = false then
11 |  |  |  Covered[guard] ← true;
12 |  |  |  count ← count + 1; // incrementa cobertura
13 if count ≠ N then // se não cobriu todos vértices
14 |  return false;
15 return true;
```

Algoritmo 3.2: Verificação de cobertura total de amostras. Função checkFullCoverage().

Considerando as análises de complexidade do Algoritmo 3.1 e do Algoritmo 3.2, podemos definir a complexidade geral do algoritmo exato, dada pela Equação 3.4. No pior caso, quando $K = N$, a complexidade do algoritmo pode ser representado pela

Equação 3.5. Considere N como o número de vértices do grafo, K o tamanho do MSC ou MVG e V o tamanho dos conjuntos de guardas da iteração corrente.

$$\sum_{V=1}^{K} \frac{N!}{(N-V)!\,V!}\,(VN) \qquad para\ 1 \leq K \leq N \tag{3.4}$$

$$\theta\left(\left(2^N - 1\right) N^2\right) \tag{3.5}$$

A Figura 3.13 apresenta alguns passos do Algoritmo 3.1 para auxiliar no seu entendimento. O vetor positions é uma representação da solução candidata. O seu tamanho indica o número de guardas (V) e os valores indicam as posições dos mesmos no vetor (vértice correspondente).

A Figura 3.13(a) consiste na primeira combinação de solução candidata, após o laço da linha 6 do Algoritmo 3.1. As combinações são geradas com o auxílio de uma variável chamada pivot que, no laço da linha 13 avança os guardas para a direita em um vetor de solução candidata até não seja mais possível (Figura 3.13(b)). Todas as iterações começam com os guardas posicionados à esquerda, como pode ser visto na Figura 3.13(c). Quando um guarda alcança o final do vetor (Figura 3.13(d)), o próximo passo é avançar o guarda adjacente da direita em uma casa e posicionar o guarda atual à sua direita (Figura 3.13(e)). A iteração termina com todos os guardas posicionados à direita do vetor de soluções candidatas (Figuras 3.13(b) e 3.13(f)).

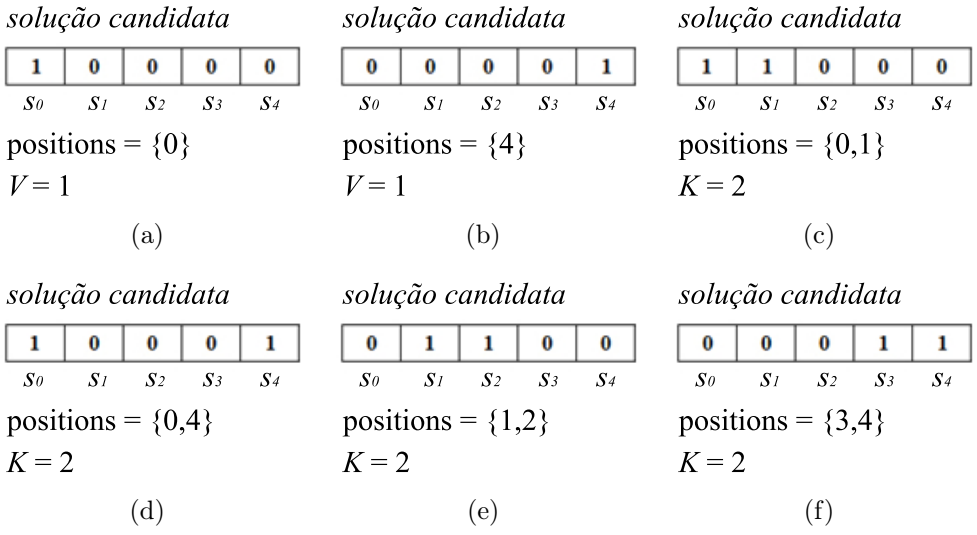


Figura 3.13. Exemplos de passos do Algoritmo 3.1.

Para reduzir o armazenamento de soluções candidatas na memória, elas são verificadas logo que criadas. Elas são armazenadas no vetor de soluções quando cobrem os

vértices do grafo e em caso contrário são descartadas. Como todas as soluções possíveis com tamanhos até o MVG são testadas, além do tempo de execução, o número de soluções candidatas também cresce exponencialmente. Porém, para instâncias com MVG reduzidos o algoritmo exato é útil como baseline para comparação de resultados com algoritmos aproximados, como é o caso do algoritmo genético de nossa metodologia descrito a seguir.

3.3.2 Algoritmo genético

Devido à alta complexidade computacional do algoritmo exato para o problema MVG ou MSC, optou-se por buscar soluções por meio de uma meta-heurística genética. Um algoritmo genético (AG) é uma técnica de busca utilizada para encontrar soluções aproximadas em problemas de otimização, usando técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e cruzamento [Goldberg, 1989].

Algoritmos genéticos são simulações de computador em que uma população de representações abstratas de soluções evoluem para um mínimo ou um máximo de uma função de avaliação. A evolução inicia a partir de um conjunto de soluções criado aleatoriamente. Então uma série de iterações (gerações) é processada até uma condição de parada. A cada geração, todas as soluções candidatas são avaliadas por uma função de *fitness*. Em seguida, alguns indivíduos são selecionados para a próxima geração de acordo com seus valores de *fitness*. Após essa seleção, os indivíduos podem ser cruzados e/ou sofrer mutação de acordo com parâmetros de probabilidade preestabelecidos. A nova população então é utilizada como entrada para a próxima iteração do algoritmo (Figura 3.14).

A função de *fitness* é um tipo particular de função usado para avaliar o quão perto uma determinada solução está de um objetivo fixado [Goldberg, 1989]. Ou seja, no caso da nossa metodologia, é importante avaliar o quão próxima uma solução candidata está da cobertura máxima dos vértices do grafo com o menor conjunto de guardas possível. Mais detalhes serão apresentados no próximo capítulo.

Ao analisar a complexidade de tempo de execução do AG (Algoritmo 3.3), verifica-se que o seu comportamento assintótico é polinomial. Basicamente o algoritmo possui três laços aninhados que percorrem o número de repetições R, o número de gerações G e o tamanho da população P.

A função de *fitness* contribui para a complexidade geral do algoritmo, pois é aplicada a todos os indivíduos em cada geração e em cada repetição. Ao percorrer a solução candidata, para cada vértice indicado como guarda, ele e todos os seus vértices adjacentes são marcados como cobertos. No final, a porcentagem de cobertura e o

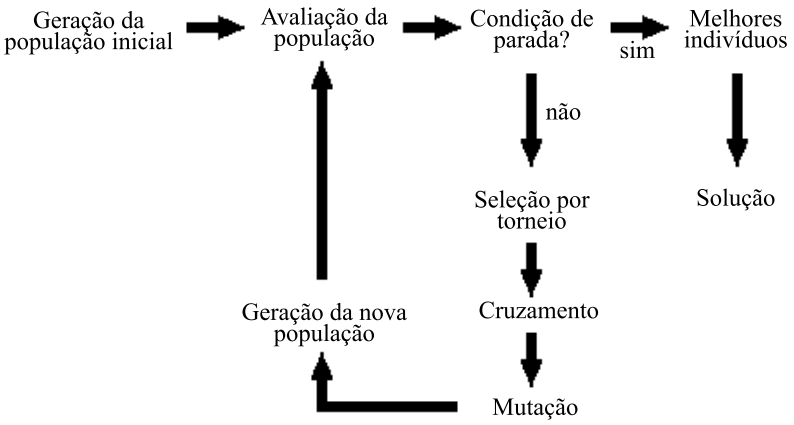


Figura 3.14. O funcionamento do algoritmo genético.

tamanho do conjunto de guardas influenciam na avaliação da solução.

A Equação 3.6 representa essa complexidade do AG, onde G indica o número de gerações da evolução, P o tamanho da população, N o número de vértices e E o número de arestas do grafo. Sendo que *(NE)* representa a complexidade da função de *fitness*.

$$\theta\left(GP\left(NE\right)\right) \tag{3.6}$$

Em algoritmos genéticos, o vetor binário que representa uma solução candidata recebe o nome de genótipo. Sendo assim, considere a Figura 3.11 como um exemplo de genótipo. Além disso, uma solução candidata única recebe o nome de indivíduo. Em nossos experimentos o tamanho do vetor equivale ao número de amostras e assim permanece durante toda a execução do AG.

O ajuste e o controle de parâmetros do AG são etapas importantes. Elas servem não apenas para guiar o comportamento da evolução das soluções, mas também para validar os resultados [Eiben et al., 1999].

Em nossos experimentos, que serão apresentados no Capítulo 4, o processo de seleção foi baseado no método de torneio, onde T indivíduos são pré-selecionados entre toda a população e o de melhor avaliação pela função *fitness* é usado no cruzamento com outro indivíduo selecionado pelo mesmo processo, ou seja, para cada cruzamento são necessários dois torneios. O cruzamento utilizado foi por ponto, ou seja, uma posição do genótipo é selecionada aleatoriamente e os segmentos formados são trocados entre os dois pais, gerando dois filhos (Figura 3.15). A mutação também foi pontual: uma

```
   entrada: R, G, P, T, Pcross, Pmut; // repetições, gerações, tamanho
            da população, indivíduos por torneio, probabilidades de
            cruzamento e de mutação
   saída  : BestCandidate; // melhor solução aproximada
 1 for seed ← 1 to R do // repetições
 2     generateInitialRandomPopulation(); // população aleatória
       inicial
 3     for g ← 1 to G do // gerações
 4         for p ← 1 to P do // indivíduos
 5             fitnessEvaluation(candidate[p]); // avaliar candidatos
 6         for p ← 1 to P do // gerando nova população
 7             winnerA ← selectionByTournament(T, candidate[]);
               // torneio 1
 8             winnerB ← selectionByTournament(T, candidate[]);
               // torneio 2
 9             news[] ← crossingOver(Pcross, winnerA, winnerB);
               // cruzamento
10             news[] ← mutation(Pmut, news[]); // mutação
11             checkBest(BestCandidate, news[]); // salva o melhor
12             p ← p + 2; // novos indivíduos
13 return BestCandidate;
```

Algoritmo 3.3: Algoritmo genético.

posição aleatória do genótipo tem o seu valor invertido.

Também foi utilizado o elitismo na meta-heurística. Nesse caso, o melhor indivíduo de uma geração é preservado, enquanto o restante da população é gerado por cruzamentos e mutações.

O número de repetições é importante para verificar a confiabilidade das soluções. Analisando o desvio padrão da média da *fitness* média é possível concluir se a população convergiu de forma homogênea a um resultado médio ao longo das repetições. Além disso, o número de cada repetição foi usado como semente para a função que gera números aleatórios.

A escolha dos métodos de seleção, cruzamento e mutação foi baseada nos benefícios que trazem para a diversidade da população, ou seja, reduzindo as chances de uma convergência prematura para mínimos ou máximos locais em um problema de otimização. Já os parâmetros de probabilidade de cruzamento e mutação são especificados após uma sistemática calibração, descrita em detalhes no Capítulo 4. Na calibração são considerados os aspectos de média da *fitness* média, média de indivíduos idênticos por geração, média de filhos melhores que os pais e os respectivos desvios padrão.

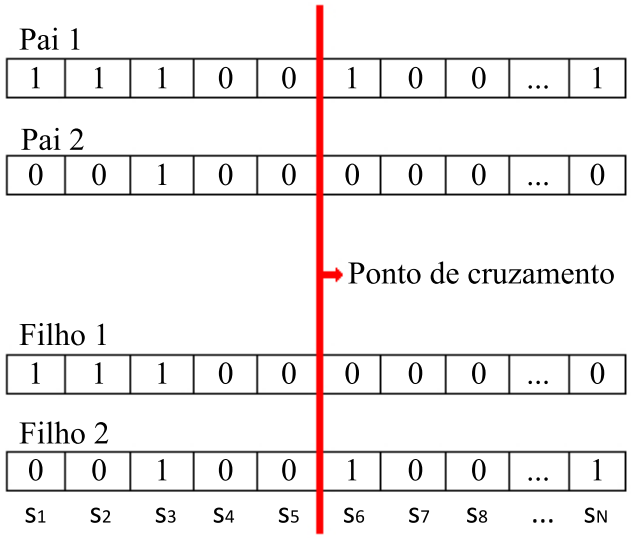


Figura 3.15. Exemplo de cruzamento por ponto. Os genótipos pais são segmentados em uma posição aleatória e geram dois indivíduos novos.

O tamanho da população e o número de gerações do AG foram ajustados de acordo com a instância de entrada, observando o comportamento de convergência das soluções ao longo das gerações.

A metodologia descrita nesse capítulo, segmentada em três etapas distintas, foi aplicada em uma série de experimentos, cujos resultados são apresentados a seguir.

3.3.2.1 Função de fitness

O objetivo de uma função *fitness* é avaliar as soluções candidatas de acordo com o objetivo de busca, ou seja, máxima cobertura com AMVG de tamanhos mínimos. Quanto mais próximo uma solução candidata está do objetivo, maior é o valor de sua avaliação.

A Equação 3.7 representa a função de *fitness* utilizada, onde M indica o número de vértices cobertos, N indica o número total de vértices do grafo e V indica o tamanho do conjunto de guardas da solução candidata.

$$fitness = 10 * \left(10 * \frac{M}{N} - \left(\frac{V}{N} * \frac{1}{100} \right) \right) \tag{3.7}$$

É importante ressaltar que a função de *fitness* deve evitar mínimos ou máximos locais, além de ser capaz de avaliar o espaço de soluções candidatas por completo. Com o intuito de verificar e ilustrar essas características, foi gerado um gráfico da

função. Esse gráfico está representado na Figura 3.16, onde se percebe claramente que a porcentagem de cobertura tem um peso dominante na avaliação de soluções candidatas.

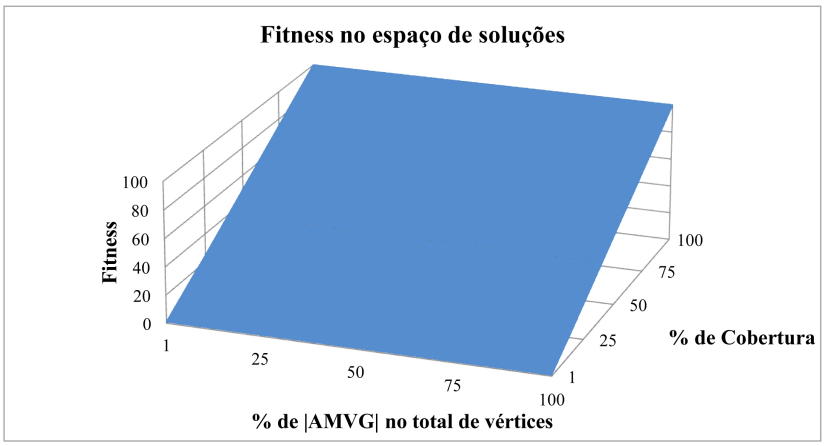


Figura 3.16. Comportamento da função fitness no espaço de soluções candidatas.

Porém, há um ajuste fino na função para diferenciar soluções com conjuntos de guardas de tamanhos diferentes. Como pode ser visto na Figura 3.17, uma ampliação da Figura 3.16, os menores conjuntos são mais bem avaliados pela *fitness*.

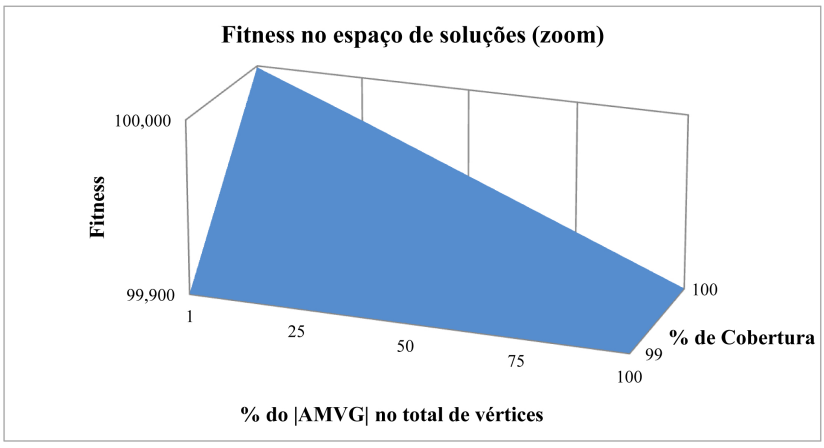


Figura 3.17. Zoom do comportamento da função fitness no espaço de soluções candidatas.

Definida a função de *fitness*, o próximo passo da aplicação do algoritmo genético no problema MVG (ou MSC) consiste na calibração de parâmetros. Esse ajuste é importante para conduzir a evolução das soluções candidatas para resultados próximos ou iguais ao ótimo para o problema em questão.

3.3.2.2 Calibração do algoritmo genético

A calibração engloba um conjunto de parâmetros do AG gerados sistematicamente e utilizados como configuração no intuito de analisar o comportamento da evolução das soluções candidatas.

Para avaliar os conjuntos de parâmetros do AG aplicados ao problema foram analisadas as evoluções das médias de: *fitness* média (MFM), tamanho médio dos AMVGs (MTM), média da melhor *fitness* (MMF), quantidade média de soluções idênticas (MSI) e número médio de filhos melhores que os pais (MFP). Além, é claro, do desvio padrão da média da *fitness* média que indica, de modo geral, a convergência das soluções nas repetições do algoritmo genético. Como são feitas várias repetições do algoritmo genético, trabalha-se com as médias das médias, como será visto nos gráficos e análises a seguir.

Em nossos experimentos, a MFM e a MTM altas indicam a convergência da população na direção do objetivo de busca, ou seja, máxima cobertura com AMVGs reduzidos. Já a MMF indica se a população está evoluindo. Por meio da MSI é possível acompanhar a evolução da convergência dos indivíduos, ou seja, se a MSI aumenta em poucas gerações indica uma convergência prematura, o que é ruim. Já a MFP mostra se a população possui uma diversidade suficiente para gerar populações melhores a cada geração, ou seja, quanto maior, melhor.

Testes preliminares foram feitos para definir o tamanho da população inicial e do número de gerações do AG. Observando a evolução das soluções nos critérios acima, foram estabelecidos os parâmetros de população inicial representada por 250 indivíduos aleatórios e número de gerações igual a 200 para um grafo esparso de entrada. O número de gerações foi definido após verificar se o valor é suficiente para a convergência das soluções candidatas, podendo variar de acordo com o número de vértices do grafo de entrada. Para o processo de calibração utilizamos um grafo esparso com 200 vértices e MVG predefinido com 4 elementos. Grafos esparsos são representações de ambientes com pouca visibilidade e por esse motivo esse tipo de grafo foi utilizado.

Para minimizar a convergência prematura das soluções candidatas foi utilizada a seleção por torneio de dois em dois indivíduos e os métodos de mutação e cruzamento por ponto aleatório.

Também foi aplicado o método do elitismo, onde o indivíduo melhor avaliado pela função de *fitness* de cada geração é preservado para a próxima iteração do algoritmo.

Após a definição do tamanho da população inicial igual a 250 e do número de gerações igual a 200, foram feitos nove testes com variações nos parâmetros de probabilidade de cruzamento ($Pcross$) e probabilidade de mutação ($Pmut$) para concluir

o conjunto de parâmetros final para aplicação do AG no problema em estudo. Os resultados dos testes são apresentados na Tabela 3.1.

Tabela 3.1. Resultados dos testes de calibração do AG.

Teste	*Pcross*	*Pmut*	$MMF \pm \sigma$	$MFM \pm \sigma$	$MTM \pm \sigma$	$MSI \pm \sigma$	$MFP \pm \sigma$	*AMVG*
1	0.60	0.01	99.97 ± 0.00	99.97 ± 0.00	45.26 ± 4.51	250.0 ± 0.0	0.0 ± 0.0	37
2	0.60	0.05	99.99 ± 0.00	99.96 ± 0.05	4.40 ± 0.66	236.8 ± 3.2	6.3 ± 12.2	4
3	0.60	0.10	99.99 ± 0.00	99.93 ± 0.09	4.11 ± 0.02	226.2 ± 4.6	1.6 ± 1.4	4
4	0.75	0.01	99.98 ± 0.00	99.98 ± 0.00	37.16 ± 5.02	250.0 ± 0.0	0.0 ± 0.0	28
5	0.75	0.05	99.99 ± 0.00	99.97 ± 0.04	4.22 ± 0.59	237.6 ± 3.9	0.9 ± 1.6	4
6	0.75	0.10	99.99 ± 0.00	99.95 ± 0.05	4.10 ± 0.02	226.4 ± 4.3	1.6 ± 1.5	4
7	0.90	0.01	99.98 ± 0.00	99.98 ± 0.00	32.96 ± 3.47	250.0 ± 0.0	0.0 ± 0.0	27
8	0.90	0.05	99.99 ± 0.00	99.97 ± 0.04	4.06 ± 0.06	237.1 ± 3.2	2.3 ± 9.4	4
9	0.90	0.10	99.99 ± 0.00	99.93 ± 0.08	4.10 ± 0.02	225.8 ± 3.5	2.0 ± 1.4	4

Todos os testes apresentam uma evolução satisfatória da média da *fitness* média e desvio padrão baixo, indicando que o número de repetições executadas no AG foi adequado. A Figura 3.18 ilustra o comportamento da MFM em todos os testes. As curvas estão sobrepostas no gráfico.

Constatou-se que a baixa probabilidade de mutação de 1%, presente nos testes 1, 4 e 7, causou a convergência prematura das soluções (Figura 3.21), a convergência prematura dos tamanhos dos conjuntos de guardas (Figura 3.19), uma estaguinação na evolução da melhor *fitness* (Figura 3.20) e afetou a melhoria das soluções filhas em relação às soluções pais (Figura 3.22). Além disso, esses três testes não encontraram AMVGs adequados (Tabela 3.1). Portanto, os parâmetros dos testes 1, 4 e 7 foram descartados.

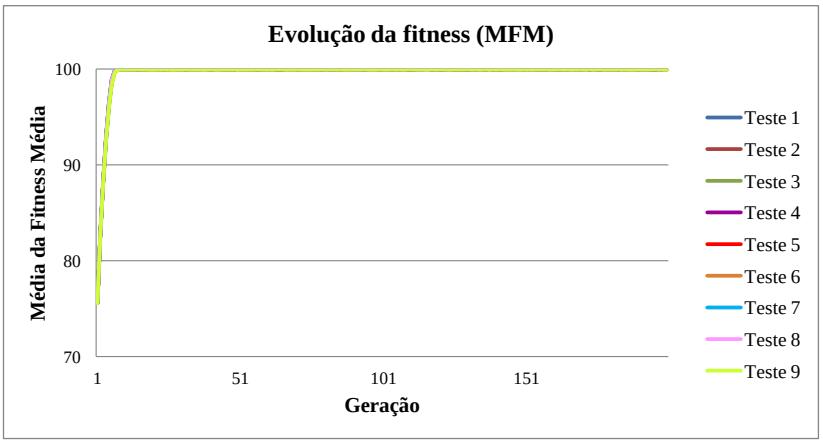


Figura 3.18. Média da fitness média ao longo das gerações na calibração do AG.

Analisando os demais testes, verificamos que ao utilizar probabilidades de mutação iguais a 5% nos testes 2, 5 e 8, os mesmos apresentam instabilidade na melhoria

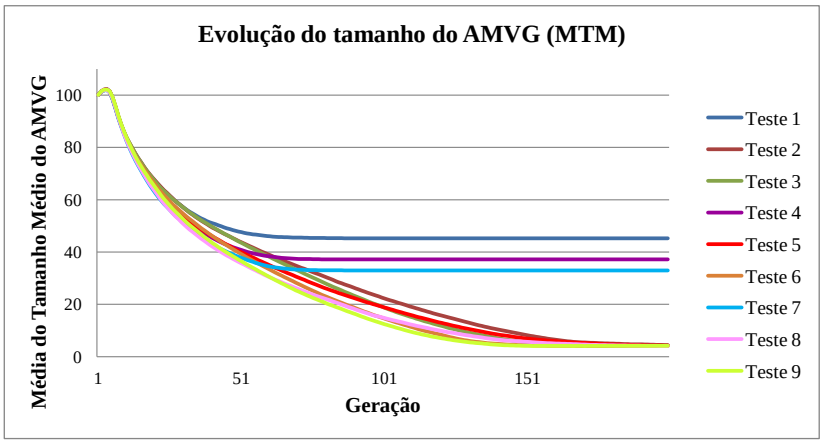


Figura 3.19. Média do tamanho médio do AMVG ao longo das gerações na calibração do AG.

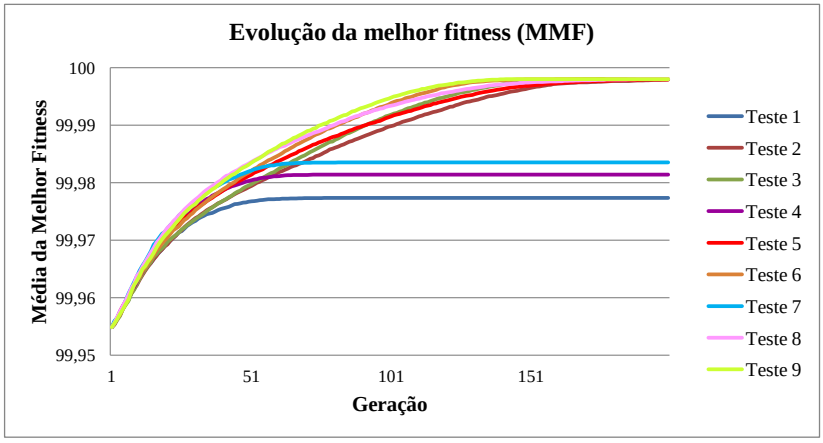


Figura 3.20. Média da melhor fitness ao longo das gerações na calibração do AG.

das médias dos filhos em relação aos pais (MFP) ao longo das repetições, o que pode ser constatado pelo alto desvio padrão apresentado (Tabela 3.1). Logo, os parâmetros desses testes também foram descartados.

Os três conjuntos de parâmetros restantes, referentes aos testes 3, 6 e 9, apresentaram bons resultados em todos os quesitos de avaliação descritos na Tabela 3.1. Porém, o teste 9 apresentou resultados um pouco melhores na média de soluções idênticas (MSI) e na média de filhos melhores que os pais (MFP) como pode ser verificado na Tabela 3.1. Uma média menor de soluções idênticas (MSI) mostra que há diversidade na população e uma média maior de filhos melhores que os pais (MFP) indica que a população tem a capacidade de evoluir para o objetivo de busca ao longo das gerações.

De modo geral, constatou-se que o parâmetro de probabilidade de mutação com valor alto influenciou consideravelmente nos resultados, justamente por auxiliar na ma-

nutenção da diversidade da população. A influência da probabilidade de cruzamento é mais sutil, mas valores altos desse parâmetro também apresentaram resultados melhores. A partir dessas análises, ficaram definidos como parâmetros padrão: $Pmut = 10\%$ e $Pcross = 90\%$.

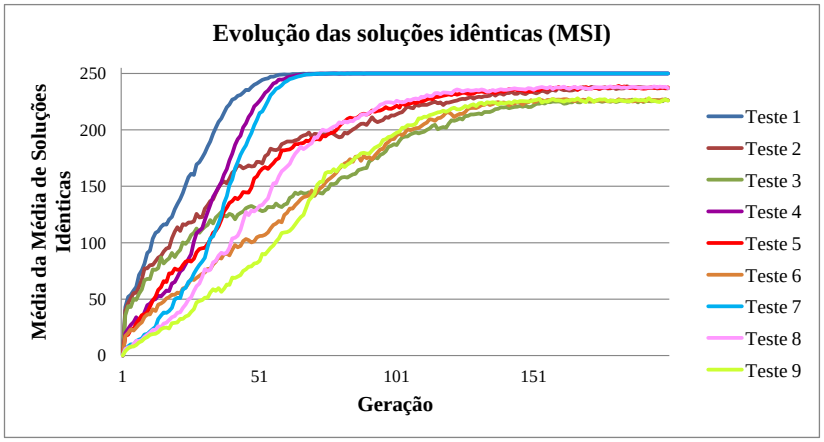


Figura 3.21. Média da média de soluções idênticas ao longo das gerações na calibração do AG.

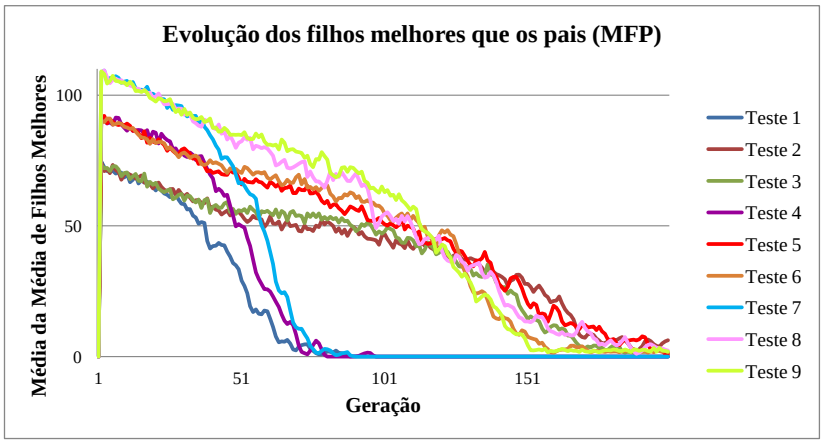


Figura 3.22. Média da média de filhos melhores que os pais ao longo das gerações na calibração do AG.

Capítulo 4

Experimentos e resultados

Ao longo da pesquisa vários experimentos foram realizados de forma a avaliar e validar a metodologia proposta. A partir da análise dos primeiros resultados, novas ideias surgiram para explorar o tema e enriquecer o trabalho.

As linguagens de programação utilizadas em todas as implementações foram C ou C++ e os testes foram realizados em um computador constituído por uma CPU Intel® Core™ i7 860 de 2.80 GHz com 4GB de memória RAM.

São descritos a seguir os cinco grupos de experimentos concluídos. O primeiro consiste em um comparativo entre os algoritmos genético e exato que buscam o MVG em grafos esparsos. O segundo é uma aplicação do algoritmo genético em amostras (*waypoints*) dos mapas do jogo *Counter-Strike*. O terceiro apresenta alterações na função de *fitness* do algoritmo genético para buscar conjuntos de observadores que cobrem porcentagens diferentes de amostras. O quarto experimento aplica do algoritmo exato em alguns mapas do *Counter-Strike* para comparar com os resultados do algoritmo genético. E o quinto e último experimento é uma aplicação da metodologia em um ambiente tridimensional arbitrário, utilizando amostragem uniforme e algoritmo genético.

4.1 Algoritmos exato e genético em grafos esparsos

Como citado anteriormente, tanto o MVG quanto o equivalente MSC são problemas de otimização NP-Difíceis. Logo, o algoritmo para encontrar uma solução exata é exponencial e o tamanho do MVG impacta seriamente no tempo de execução.

Para o primeiro experimento foi desenvolvido um aplicativo em C++, chamado *GraphGenerator*, que gera grafos esparsos com MVGs predefinidos. Esses grafos foram utilizados como entradas para comparar o desempenho entre os algoritmos exato e

genético. Percebe-se, portanto, que esse experimento foca apenas na terceira etapa de nossa metodologia, a busca pelo MVG (ou MSC) em grafos.

O algoritmo para gerar esses grafos é simples. Primeiramente são criados os vértices do MVG e arestas entre eles formando um anel. Em seguida, os demais vértices são distribuídos uniformemente como adjacentes de um dos vértices do conjunto MVG. Um exemplo de um grafo com 100 vértices e MVG de tamanho 4 pode ser visto na Figura 4.1.

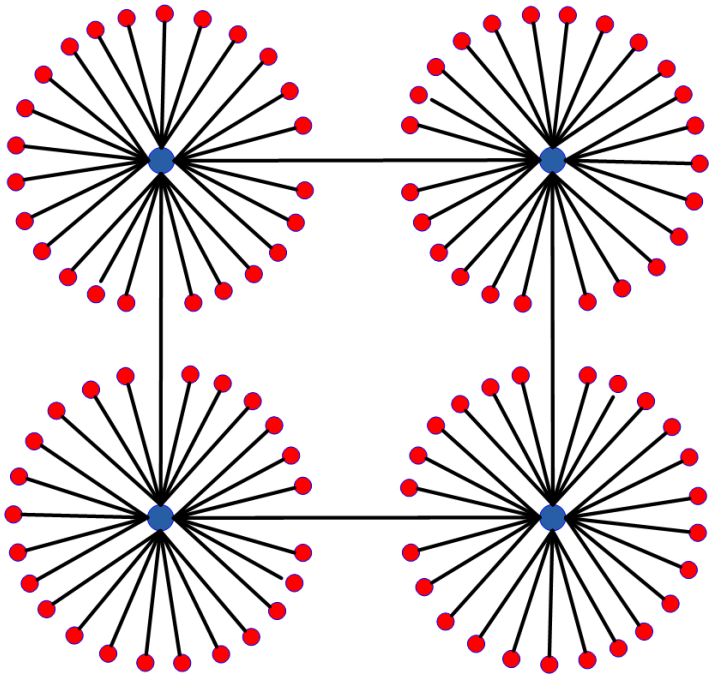


Figura 4.1. Exemplo de um grafo esparso com 4 guardas.

Optou-se por utilizar grafos esparsos como entrada pelo fato dos mesmos serem os piores casos de visibilidade. No contexto de monitoramento, um grafo esparso equivale a um ambiente com pouca visibilidade. Em contrapartida, um grafo completo seria um caso de visibilidade perfeita, pois qualquer vértice sozinho cobre todos os demais.

4.1.1 Algoritmo genético

Como o objetivo do AG é encontrar uma solução aproximada com máxima cobertura por meio de um conjunto mínimo de vértices guardas, foi elaborada uma função de *fintess* para avaliar cada solução candidata.

4.1.1.1 Aplicação em grafos esparsos

Após a definição dos parâmetros na calibração do AG, vistos no Capítulo 3, foram feitos experimentos com os grafos esparsos utilizando os algoritmos genético e exato com o intuito de comparar os resultados. As características desses grafos foram: 100, 500 e 1000 vértices e MVGs predefinidos de tamanhos iguais a 1, 2, 3 e 4.

O conjunto de parâmetros do AG para cada gráfico foi: Repetições = 30; População Inicial = 250; *Pcross* = 90%; *Pmut* = 10%; Gerações (Grafos de 100 vértices) = 200; Gerações (Grafos de 500 vértices) = 600; Gerações (Grafos de 1000 vértices) = 1100; Seleção por torneio entre 2 indivíduos; Mutação pontual; Cruzamento pontual; Elitismo ativo;

Verificou-se que o AG apresentou satisfatória convergência da MFM, da MTM e da MMF como pode ser visto nas Figuras 4.2, 4.3 e 4.4. Na Figura 4.2, as curvas estão sobrepostas. A cobertura foi de 100% das amostras em todos os casos.

O tempo de execução do AG, além da média da *fitness* média e a média do tamanho médio do AMVG são apresentados na Tabela 4.1. É importante ressaltar que o tempo de execução do AG apresentado considerou as 30 repetições. Constatou-se, também, que o AG encontrou as soluções exatas para todos os grafos esparsos do experimento.

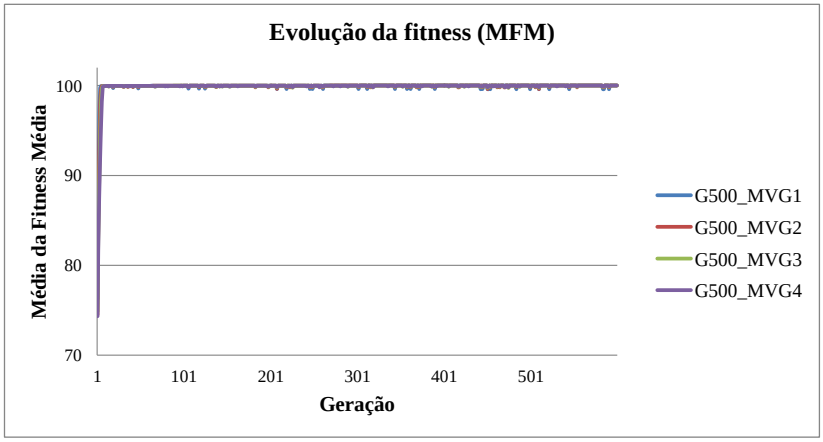


Figura 4.2. Evolução da média da fitness média no grafo esparso de 500 vértices.

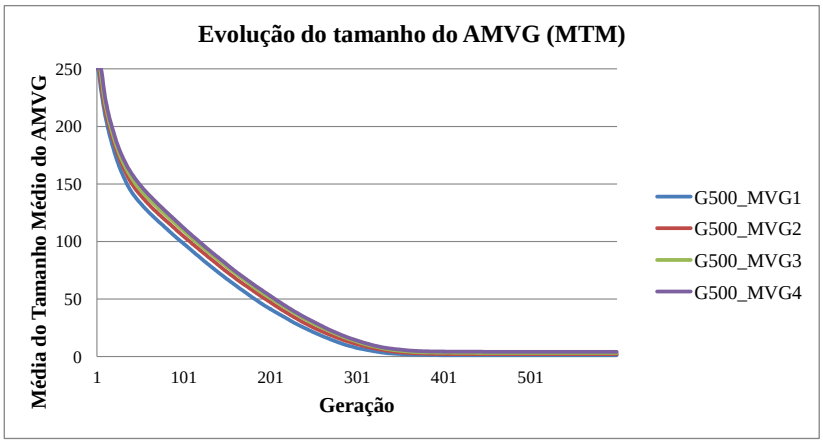


Figura 4.3. Evolução da média do tamanho médio do AMVG no grafo esparso de 500 vértices.

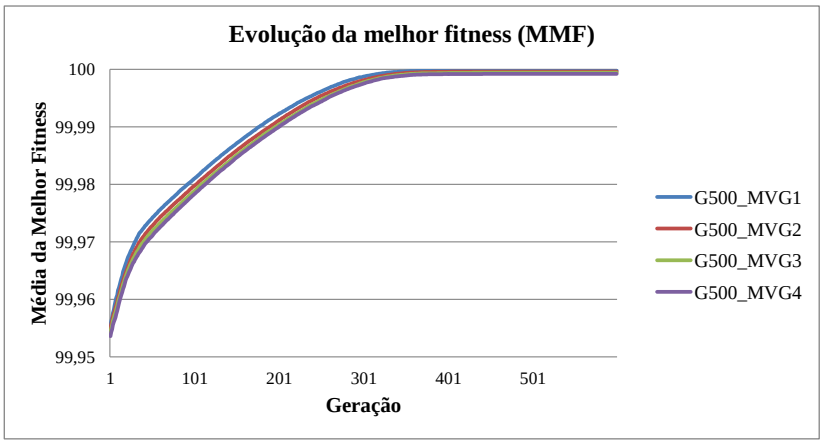


Figura 4.4. Evolução da melhor fitness no grafo esparso de 500 vértices.

4.1.2 Algoritmo exato

O algoritmo exato, descrito no Capítulo 3, foi aplicado aos mesmos grafos de entrada do AG, ou seja, grafos esparsos com 100, 500 e 1000 vértices e MVGs predefinidos de tamanhos iguais a 1, 2, 3 e 4.

Com base nos resultados apresentados na Figura 4.5 é possível observar o comportamento exponencial do algoritmo exato em relação ao tempo de execução quando o tamanho do MVG é aumentado. Em contrapartida, o algoritmo genético apresentou um comportamento polinomial, o que viabiliza a sua utilização em grafos mais complexos.

Os resultados comparativos são apresentados em detalhes na Tabela 4.1, onde verificamos a sensibilidade do algoritmo exato em relação ao tamanho do MVG. Já no AG o tempo de execução pouco variou de acordo com o MVG em grafos com tamanhos similares de vértices. Para grafos com MVG de tamanhos reduzidos, mais especificamente abaixo de 4, o algoritmo exato encontrou soluções mais rápido que o AG.

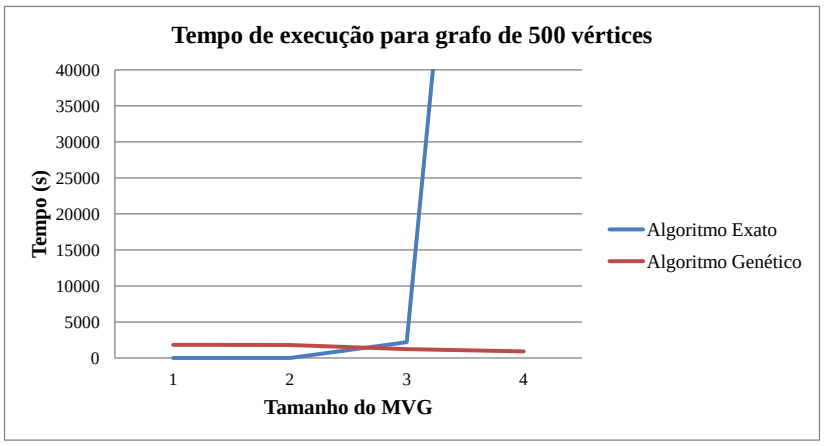


Figura 4.5. Comparação do tempo de execução dos algoritmos exato e genético para um grafo esparso de 500 vértices.

Tabela 4.1. Comparativo entre a aplicação dos algoritmos exato e genético em grafos esparsos.

Grafo		*Algoritmo exato*		*Algoritmo genético*			
Vértices	*MVG*	*MVG*	*Tempo(s)*	*AMVG*	$MFM \pm \sigma$	$MTM \pm \sigma$	*Tempo(s)**
100	1	1	0	1	99.99 ± 0.13	1.11 ± 0.02	181
100	2	2	0	2	99.99 ± 0.13	2.11 ± 0.02	186
100	3	3	3	3	99.87 ± 0.13	3.10 ± 0.02	186
100	4	4	77	4	99.90 ± 0.12	4.10 ± 0.02	151
500	1	1	0	1	99.99 ± 0.10	1.11 ± 0.02	1830
500	2	2	10	2	99.99 ± 0.05	2.11 ± 0.02	1811
500	3	3	2201	3	99.99 ± 0.06	3.11 ± 0.02	1230
500	4	4	171312	4	99.99 ± 0.04	4.11 ± 0.02	930
1000	1	1	0	1	99.99 ± 0.10	1.11 ± 0.02	4481
1000	2	2	141	2	99.99 ± 0.06	2.11 ± 0.02	4392
1000	3	3	39067	3	99.99 ± 0.04	3.11 ± 0.02	3740
1000	4	**	**	4	99.99 ± 0.03	4.11 ± 0.02	2490

* O tempo do algoritmo genético considerou 30 repetições.
** O algoritmo exato ficou em execução por mais de 1000000 segundos, quando foi interrompido pelo usuário.

4.2 Algoritmo genético em waypoints do Counter-Strike

O segundo experimento teve como objetivo aplicar o AG para encontrar AMVG entre os *waypoints* dos mapas do *Counter-Strike*. Com esse resultado em mãos é possível posicionar agentes e monitorar todos os demais *waypoints* do grafo de mobilidade utilizado pelos agentes virtuais do jogo.

Como mencionado, os *waypoints* são vértices de um grafo utilizado para planejamento de trajetórias de agentes virtuais em ambientes de jogos, uma técnica de inteligência artificial. Em alguns casos, como no *Counter-Strike*, os *waypoints* podem armazenar informações úteis para os agentes, as quais são utilizadas em tomadas de decisão. Uma dessas informações é uma *flag* que determina se o vértice é um *camp point* ou não. *Camp points* são *waypoints* de posicionamento estratégico criados pelo projetista de fases do jogo. Os *camp points* não são definidos por uma análise computacional dos mapas, logo, não necessariamente equivalem ao MVG ou AMVG. Sendo assim, esses dados foram extraídos para efeitos comparativos de cobertura.

Os grafos de mobilidade dos agentes em mapas do *Counter-Strike* são gerados automaticamente a partir de vários registros de movimentação de jogadores reais seguidos de ajustes manuais. Ao longo dos anos esses grafos foram aperfeiçoados e atualmente são usados por todos os sistemas de *bots* desenvolvidos para o jogo.

Detalhes técnicos sobre como utilizar o *Half-Life I SDK* para a extração de *waypoints* e a geração do grafo de visibilidade no jogo *Counter-Strike* são descritos no Apêndice B.

Após coletar os *waypoints* de cada mapa, foram gerados os grafos de visibilidade por meio do teste de interseção raio-malha. Sendo assim, no segundo passo da metodologia, foram criadas as arestas de visibilidade.

Para encontrar o AMVG no grafo de visibilidade, terceiro passo da metodologia, foi utilizado o AG. As três etapas da metodologia foram aplicadas em todos os mapas do *Counter-Strike* e os resultados são apresentados na Tabela 4.2.

O conjunto de parâmetros do AG para todos os mapas foi: Repetições = 30; População Inicial = 250; $Pcross$ = 90%; $Pmut$ = 10%; Gerações = 700; Seleção por torneio entre 2 indivíduos; Mutação pontual; Cruzamento pontual; Elitismo ativo;

Houve convergência satisfatória da média da *fitness* média e da média do tamanho médio do AMVG e da média da melhor *fitness* em todos os mapas do *Counter-Strike*. Podemos observar alguns exemplos nas Figuras 4.6, 4.7 e 4.8.

Tabela 4.2. Algoritmo genético em waypoints dos mapas do Counter-Strike.

Mapa de entrada			*Camp points*		*Algoritmo genético*	
Nome	*Vértices*	*Arestas*	*Vértices*	*Cobertura (%)*	*AMVG*	$MFM \pm \sigma$
As_oilrig	487	6313	39	70.2	32	99.98 ± 0.01
Cs_747	388	6210	24	79.1	33	99.97 ± 0.01
Cs_assault	470	17782	50	94.9	18	99.99 ± 0.01
Cs_backalley	375	4592	44	66.7	2	99.96 ± 0.01
Cs_estate	399	9350	52	97.7	23	99.98 ± 0.01
Cs_havana	396	3883	30	69.9	34	99.98 ± 0.01
Cs_italy	391	5116	72	97.4	31	99.98 ± 0.01
Cs_militia	603	12194	63	90.7	29	99.98 ± 0.01
Cs_office	386	4968	52	96.6	3	99.98 ± 0.06
Cs_siege	557	12845	51	74.9	30	99.98 ± 0.01
De_airstrip	521	7596	45	63.7	39	99.98 ± 0.00
De_aztec	521	11495	61	81.0	31	99.99 ± 0.00
De_cbble	631	14106	67	82.6	40	99.98 ± 0.00
De_chateau	540	6627	22	53.5	42	99.98 ± 0.00
De_dust	476	9619	43	81.3	26	99.98 ± 0.01
De_dust2	433	7572	54	96.5	26	99.98 ± 0.01
De_inferno	440	5206	49	83.2	25	99.98 ± 0.01
De_nuke	618	12325	41	69.6	16	99.99 ± 0.00
De_piranesi	572	8531	75	77.3	47	99.98 ± 0.00
De_prodigy	337	2878	40	79.2	34	99.97 ± 0.01
De_storm	492	10095	41	75.6	26	99.98 ± 0.01
De_survivor	513	10121	46	81.7	4	99.98 ± 0.05
De_torn	418	4645	33	58.6	36	99.98 ± 0.01
De_train	521	11667	42	89.4	15	99.99 ± 0.01
De_vertigo	383	4529	51	88.3	36	99.98 ± 0.01

Além disso, é importante ressaltar que todas as soluções de AMVG para os mapas cobriram 100% das amostras, diferentemente dos conjuntos de *camp points* do jogo que não o fazem. Como podemos averiguar na Tabela 4.2, o mapa De_dust possui 43 *camp points* que cobrem 81.3% dos *waypoints*, enquanto o AMVG possui 26 guardas que cobrem 100% dos mesmos. A Figura 4.9 ilustra esse exemplo, onde os pontos azuis representam os guardas, os pontos pretos as amostras não cobertas e os pontos verdes as arestas cobertas.

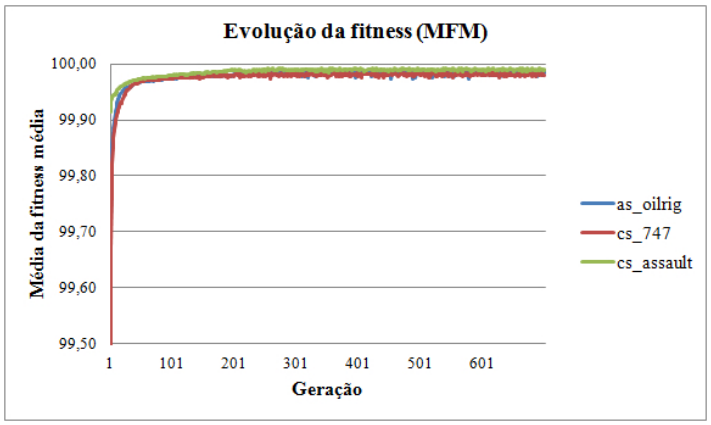


Figura 4.6. Evolução da média da fitness média em alguns mapas do Counter-Strike.

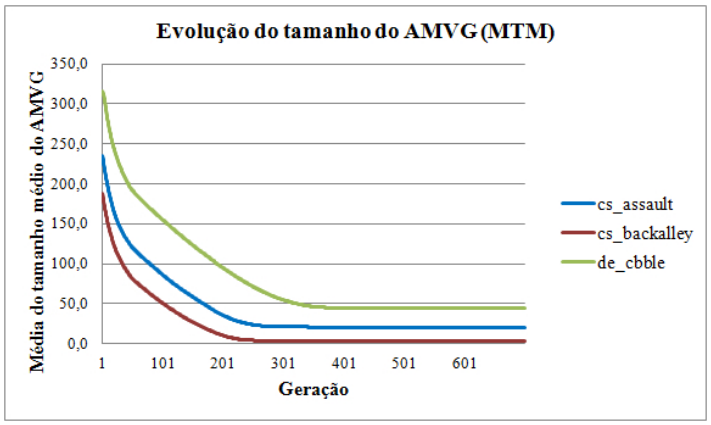


Figura 4.7. Evolução da média do tamanho médio do AMVG em alguns mapas do Counter-Strike.

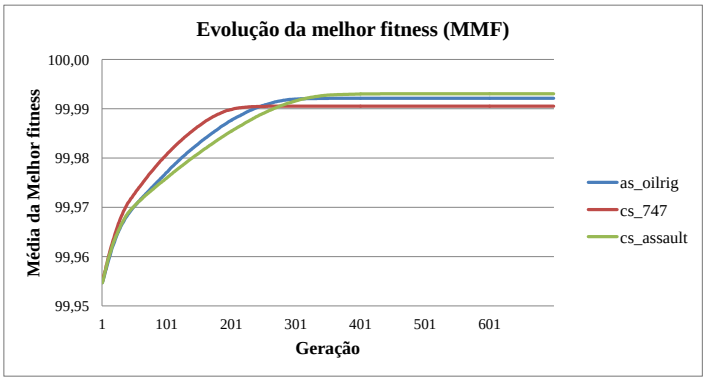
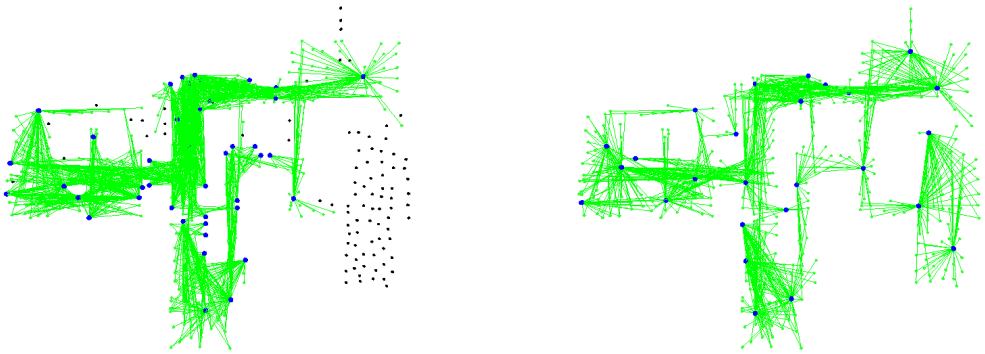


Figura 4.8. Evolução da melhor fitness em alguns mapas do Counter-Strike.



(a) (b)

Figura 4.9. Cobertura por camp points (a) e pelo AMVG (b) no mapa De_dust.

4.3 Tradeoff entre cobertura e número de guardas

No experimento anterior foi possível observar que os mapas Cs_747, Cs_havana, De_chateau e De_torn apresentam conjuntos de *camp points* menores do que os AMVGs encontrados pelo algoritmo genético, porém com coberturas inferiores a 100% das amostras. O mapa Cs_747, por exemplo, possui 24 *camp points* que cobrem 79.1% dos *waypoints*, enquanto o AMVG possui 33 guardas e cobre totalmente o grafo (Tabela 4.2).

Esses resultados inspiraram um novo experimento, o qual consiste em analisar o *tradeoff* entre a porcentagem de cobertura de amostras e o número de guardas necessários para tal. A função de *fitness* foi modificada para guiar as populações para coberturas de 60%, 70%, 80% e 90% no AG. Como forma ilustrativa, os resultados dos AMVGs encontrados foram comparados com os *camp points*.

Na *fitness* foi feito um corte na avaliação das coberturas, ou seja, o ganho de avaliação referente à cobertura não ultrapassa a cobertura objetivo. Por exemplo, se o objetivo for cobrir 60% das amostras, as soluções candidatas que atingirem 60% ou mais de taxa de cobertura recebem a mesma pontuação nesse quesito e o passa a diferenciá-las é o tamanho do AMVG. Resumindo, a partir da cobertura objetivo, o tamanho dos conjuntos é que guia a evolução. Esse exemplo de corte pode ser visto na Figura 4.10, onde a linha preta define o limite em que a cobertura influencia na avaliação dos indivíduos. As soluções da área destacada em amarelo são diferenciadas pelo tamanho de seus AMVGs.

No contexto de jogos de tiro, analisar os mapas por essa perspectiva de diferentes taxas de coberturas é bastante interessante. Como os jogos são dinâmicos e os jogadores

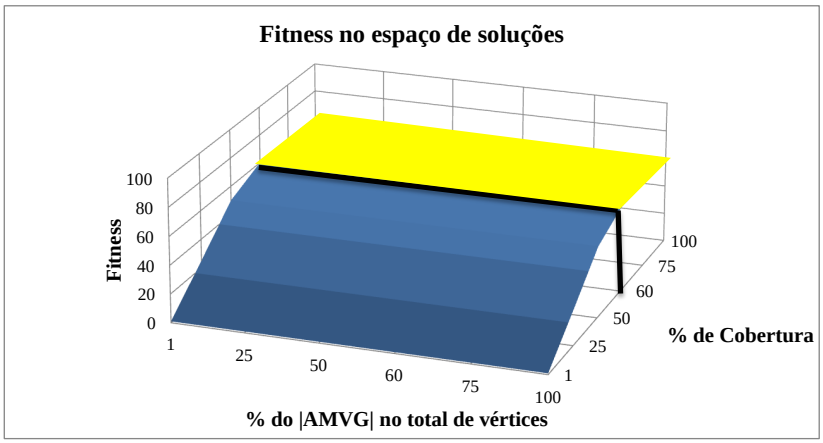


Figura 4.10. Avaliação da função fitness no espaço de soluções para cobertura de 60%.

são abatidos ao longo do tempo, as equipes ficam reduzidas e precisam se reposicionar para cobrir o máximo dos mapas. Além disso, se o foco da ação de combate por uma determinada área do cenário, como no caso de atacar uma base inimiga específica, a demanda pela cobertura passa a ser parcial e é importante identificar quantos jogadores são necessários para monitorar essas regiões.

Já no contexto de monitoramento, é interessante analisar esse *tradeoff* entre a porcentagem de cobertura e quantidade mínima de guardas para tal, ou seja, dependendo dos recursos disponíveis, pode ser mais interessante cobrir 90% de um ambiente se houver uma redução significativa do número de guardas, economizando recursos.

Além dos mapas Cs_747, Cs_havana, De_chateau e De_torn, foram selecionados outros com características distintas em número de vértices e arestas para completar o experimento. Foram eles: Cs_italy, Cs_militia, De_cbble, De_dust e De_prodigy.

O conjunto de parâmetros do AG para todos os mapas foi: Repetições = 30; População Inicial = 200; $Pcross$ = 90%; $Pmut$ = 10%; Gerações = 400; Seleção por torneio entre 2 indivíduos; Mutação pontual; Cruzamento pontual; Elitismo ativo;

Na Figura 4.11 são apresentadas as evoluções das médias da *fitness* média para diferentes coberturas do mapa De_cbble. A Figura 4.13 apresenta a evolução da melhor *fitness* ao longo das gerações em um dos testes de *tradeoff*. Observa-se que o valor máximo da *fitness* não ultrapassa o objetivo de cobertura.

Já na Figura 4.12, observa-se a evolução dos tamanhos médios do AMVG para o mesmo mapa. Um detalhe curioso ocorre na cobertura objetivo de 100%, onde a MTM da população cresce enquanto a cobertura máxima ainda não foi alcançada e posteriormente diminui quando o ajuste fino do tamanho dos conjuntos passa a dominar a avaliação da função de *fitness*.

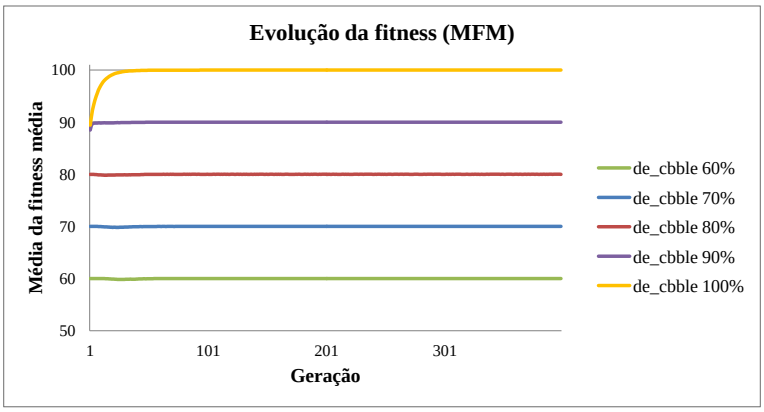


Figura 4.11. Evolução da média da fitness média para taxas de cobertura distintas no mapa De_cbble.

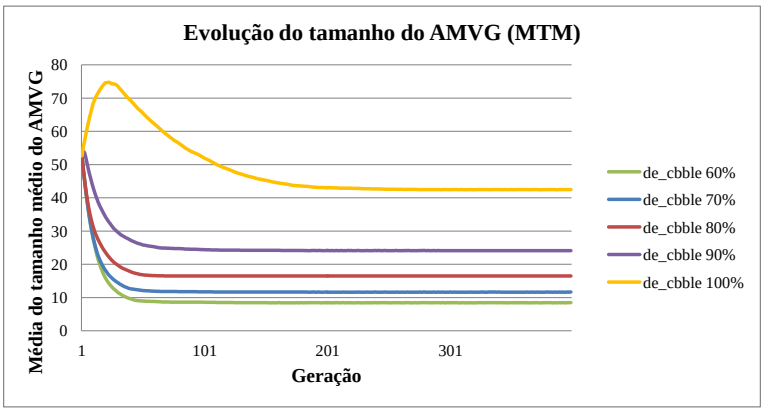


Figura 4.12. Evolução da média do tamanho médio do AMVG para taxas de cobertura distintas no mapa De_cbble.

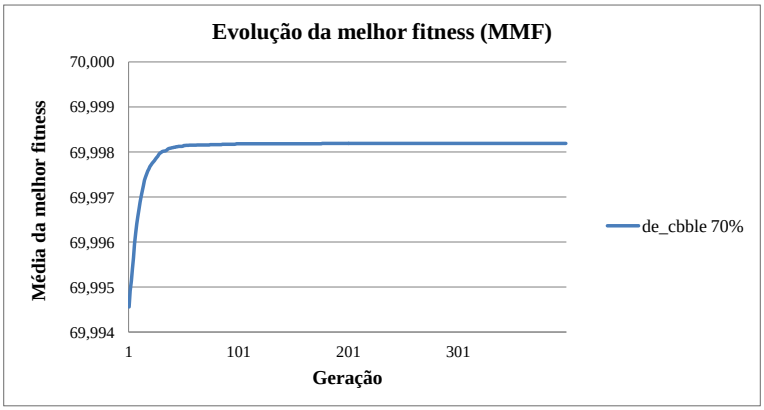


Figura 4.13. Evolução da melhor fitness crobrindo 70% do mapa De_cbble.

A Tabela 4.3 mostra o *tradeoff* das diferentes taxas de cobertura e o número de guardas necessário para cobri-las. No mapa De_cbble, por exemplo, são necessários 40 guardas para cobrir todas as amostras e 22 para cobrir 90% delas, ou seja, uma economia de quase 50% no número de guardas para uma perda de 10% de cobertura. A Tabela 4.4 apresenta as médias da *fitness* média e os respectivos desvios padrão do experimento, indicando uma convergência satisfatória.

Tabela 4.3. Tradeoff dos tamanhos do AMVG para diferentes taxas de cobertura das amostras.

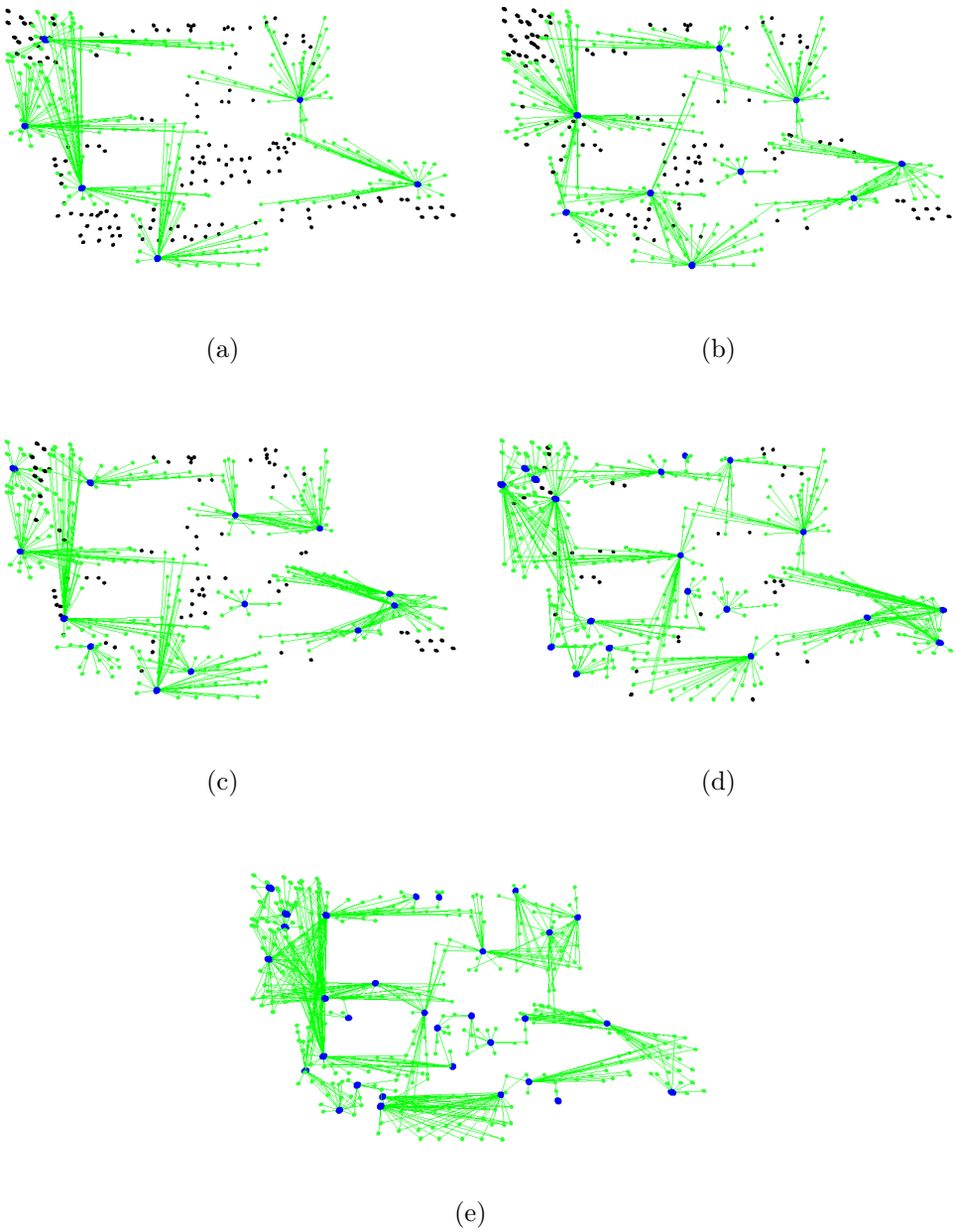
Mapa	*AMVG*				
	Cobertura de 60%	*Cobertura de 70%*	*Cobertura de 80%*	*Cobertura de 90%*	*Cobertura de 100%*
Cs_747	5	7	10	15	33
Cs_havana	5	8	12	19	34
Cs_italy	6	9	13	19	31
Cs_militia	6	7	7	12	29
De_cbble	6	9	14	22	40
De_chateau	10	13	17	24	42
De_dust	5	6	9	12	26
De_prodigy	7	10	15	20	34
De_torn	8	11	14	19	36

Tabela 4.4. Resultado das MFMs para diferentes taxas de cobertura das amostras.

Mapa	$MFM \pm \sigma$				
	Cobertura de 60%	*Cobertura de 70%*	*Cobertura de 80%*	*Cobertura de 90%*	*Cobertura de 100%*
Cs_747	59.96 ± 0.05	69.98 ± 0.04	79.98 ± 0.04	89.97 ± 0.03	99.97 ± 0.01
Cs_havana	59.98 ± 0.03	69.98 ± 0.02	79.98 ± 0.02	89.98 ± 0.02	99.98 ± 0.01
Cs_italy	59.98 ± 0.01	69.97 ± 0.03	79.98 ± 0.01	89.97 ± 0.02	99.98 ± 0.01
Cs_militia	59.99 ± 0.00	69.99 ± 0.01	79.99 ± 0.00	89.99 ± 0.00	99.98 ± 0.01
De_cbble	59.99 ± 0.01	69.99 ± 0.01	79.98 ± 0.01	89.99 ± 0.00	99.98 ± 0.00
De_chateau	59.98 ± 0.01	69.98 ± 0.01	79.98 ± 0.01	89.98 ± 0.01	99.98 ± 0.00
De_dust	59.98 ± 0.02	69.98 ± 0.03	79.98 ± 0.02	89.98 ± 0.01	99.98 ± 0.01
De_prodigy	59.98 ± 0.02	69.97 ± 0.01	79.96 ± 0.02	89.97 ± 0.01	99.97 ± 0.01
De_torn	59.98 ± 0.01	69.97 ± 0.01	79.97 ± 0.02	89.98 ± 0.01	99.98 ± 0.01

De modo geral, foi observada uma redução significativa no tamanho dos conjuntos AMVGs ao diminuir a porcentagem da cobertura objetivo. Também foi demonstrado que os AMVG são menores que os conjuntos de *camp points* para as mesmas faixas de coberturas de amostras em todos os casos. Por exemplo, o mapa Cs_747 possui 24

camp points que cobrem 79.1% das amostras (Tabela 4.2) e em contrapartida, necessita de 10 guardas para cobrir 80% e 7 para cobrir 70% (Tabela 4.3). Porém, é importante ressaltar que muitos *camp points* são posições no cenário que apresentam certa proteção, ou seja, evitam locais abertos, diferentemente do AMVG que não leva em consideração essa característica.



(a) (b) (c) (d) (e)

Figura 4.14. AMVGs para coberturas de (a) 60%, (b) 70%, (c) 80%, (d) 90% e (e) 100% das amostras do mapa Cs_italy.

Um exemplo visual de *tradeoff* no mapa Cs_italy é apresentado na Figura 4.14.

Nela são apresentadas as coberturas de 60%, 70%, 80%, 90% e 100% respectivamente, onde os vértices azuis são os guardas e os pretos são os *waypoints* não cobertos. Na Figura 4.14(e), todos os *waypoints* foram cobertos por um conjunto de 31 guardas.

4.4 Algoritmo exato aplicado no Counter-Strike

Na Tabela 4.5 é possível observar que o algoritmo genético encontrou conjuntos AMVGs significativamente pequenos para os mapas Cs_backalley, Cs_office e De_survivor.

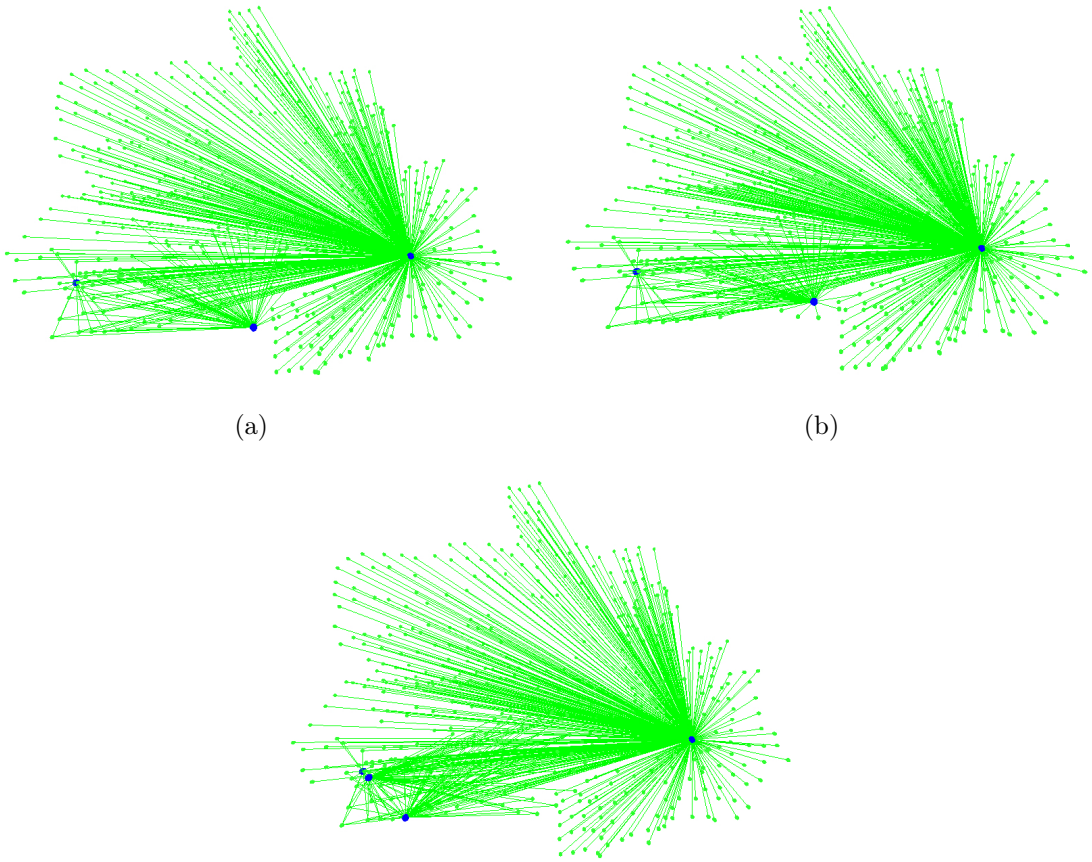
Já que o algoritmo genético convergiu para coberturas completas de amostras para todos os mapas do *Counter-Strike*, podemos afirmar que o algoritmo exato encontrará MVGs menores ou iguais aos AMGs. Como vimos no primeiro experimento, sessão 4.1, é viável aplicar o algoritmo exato em grafos com MVG de tamanho reduzido.

O objetivo desse experimento foi aplicar o algoritmo exato nos mapas Cs_backalley, Cs_office e De_survivor e comparar os MVGs do algoritmo exato com os AMVGs do algoritmo genético. Os resultados foram apresentados na Tabela 4.5, onde podemos comparar o desempenho dos algoritmos em relação ao tempo de execução do conjunto de guardas.

Tabela 4.5. Resultados comparativos entre os algoritmos exato e genético para mapas com AMVGs reduzidos.

Mapa de entrada	*Algoritmo exato*		*Algoritmo genético*		
	MVG	*Tempo(s)*	*AMVG*	$MFM \pm \sigma$	*Tempo (s)*
Cs_backalley	2	2.92	2	99.961 ± 0.112	1.52e+003
Cs_office	3	436.11	3	99.986 ± 0.057	1.96e+003
De_survivor	3	1.41e+003	4	99.989 ± 0.048	1.76e+003

Foi observado que o algoritmo genético encontra AMVGs iguais ou muito próximos para os mapas em que foi possível aplicar o algoritmo exato. Especificamente, apenas no mapa De_survivor o resultado foi diferente. Nesse caso, o algoritmo exato encontrou dois MVG de tamanho igual a 3, como pode ser visualizado nas Figuras 4.15(a) e 4.15(b). Já o algoritmo genético encontrou um AMVG de tamanho igual a 4, apresentado na Figura 4.15(c). Observa-se, também, a cobertura total das amostras nessas três figuras.



(a) (b)

(c)

Figura 4.15. Coberturas completas no mapa De_survivor. As Figuras (a) e (b) foram os resultados do MVG de tamanho 3 e a Figura (c) foi o resultado do AMVG de tamanho 4.

4.5 Algoritmo genético em um ambiente arbitrário

Considerando que mapas de um determinado jogo não possuam *waypoints* como no *Counter-Strike* ou que se deseje aplicar a metodologia em um ambiente tridimensional genérico fora do contexto de jogos. Nesses casos é possível utilizar métodos alternativos de amostragem, entre eles, a clássica amostragem uniforme.

A amostragem uniforme em um volume consiste em um conjunto de pontos equidistantes nos eixos X, Y e Z, como um *grid* tridimensional com os pontos de amostra nos centros das subdivisões. A distância entre as amostras pode ser ajustada de acordo com a necessidade da aplicação.

Para esse experimento foi selecionado o mapa Cs_assault do jogo *Counter-Strike* o qual possui características de um ambiente arbitrário, ou seja, não planar, não ortogonal e com buracos. A malha poligonal do mesmo foi exportada para um arquivo

por meio de um software chamado *GCFScape*, próprio para extração de arquivos do *Half-Life I* e similares.

Para esse experimento foi desenvolvido em C++ um aplicativo chamado *Cover-Test* utilizando o motor gráfico *Irrlicht*. O mapa selecionado foi carregado e uma amostragem uniforme foi gerada. Observou-se que foram gerados pontos em excesso em algumas regiões, mais especificamente em grandes áreas abertas, e poucos pontos em outras, como em túneis estreitos. Sendo assim, foi necessário ajustar a distância entre as amostras até que todas as regiões fossem preenchidas por pontos.

Como mencionado no Capítulo 3, com o intuito de aperfeiçoar o conjunto de amostras para o objetivo da aplicação, que é monitorar a presença de pessoas no ambiente, foi feita uma delimitação da zona de interesse, formada por zonas verdes e vermelhas. Como já mencionado, zonas verdes são volumes de validação que, unidas, delimitam toda a região que pessoas podem ocupar no ambiente desconsiderando os obstáculos. Zonas vermelhas são volumes que representam obstáculos ou buracos dentro das zonas verdes. Essas zonas são representadas pelas geometrias redimensionáveis de paralelepípedos, esferas e prismas, sobre as quais são aplicadas operações booleanas que resultam no volume de validação final, a zona de interesse.

As zonas verdes e vermelhas foram inseridas manualmente por meio do software de modelagem tridimensional *Autodesk 3D Max 2010*. Ao todo foram criadas 97 geometrias para compor essas zonas no mapa Cs_assault. É importante ressaltar que a delimitação manual dessas zonas verdes e vermelhas pode proporcionar resultados diferenciados dependendo do usuário ou da escolha de formas geométricas para criá-las.

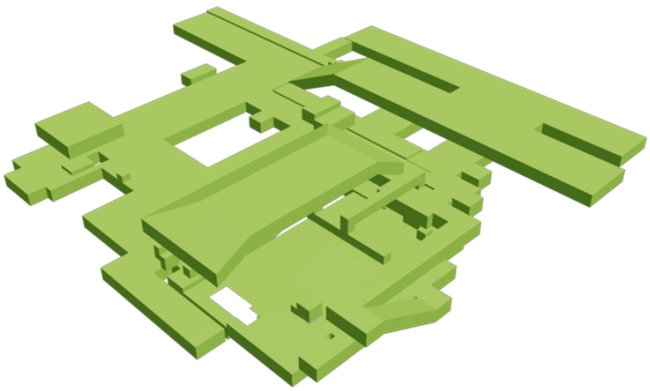


Figura 4.16. Volume de validação criado para otimizar as amostras da distribuição uniforme no cenário Cs_assault.

As coordenadas dessas geometrias foram exportadas para um arquivo que serviu de entrada para o aplicativo *CoverTest*. O software *CoverTest* foi, então, responsável pela operação de subtração booleana entre zonas verdes e vermelhas, obtendo a zona de interesse resultante ilustrada na Figura 4.16.

Por meio de cálculos matemáticos, cada ponto da amostragem uniforme passa por uma verificação que determina se ele está dentro ou fora de uma zona vermelha ou verde. Os que não pertencem a alguma zona ou pertencem às zonas vermelhas são descartados e os demais são validados. Dessa forma, portanto, foi feita a otimização da amostragem para a aplicação desejada. Um exemplo desse processo pode ser visto nas Figuras 4.17 e 4.18, onde um conjunto de amostras uniformes passa por uma filtragem e resultam em pontos válidos, destacados na cor verde. A complexidade dessa etapa de filtragem leva em consideração o número de amostras N e o número de zonas (vermelhas ou verdes) Z e é dada pela Equação 4.1.

$$\theta\left(NZ\right) \tag{4.1}$$

A partir da geometria do mapa e dos pontos válidos, foi aplicado o teste de interseção raio-malha por meio do método *getSceneNodeAndCollisionPointFromRay* do motor *Irrlicht* para gerar o grafo de visibilidade correspondente.

Para concluir o experimento foi utilizado o algoritmo genético para buscar o AMVG no grafo de visibilidade. O conjunto de parâmetros do AG para o ambiente foi: Repetições = 30; População Inicial = 500; $Pcross$ = 90%; $Pmut$ = 10%; Gerações = 1500; Seleção por torneio entre 2 indivíduos; Mutação pontual; Cruzamento pontual; Elitismo ativo;

Tabela 4.6. Resultados do algoritmo genético em diferentes métodos de amostragem.

Mapa Cs_ assault			*Algoritmo genético*		
Método de amostragem	*Vértices*	*Arestas*	*AMVG*	$MFM \pm \sigma$	*Tempo (s)*
Waypoints	487	6313	18	99.990 ± 0.006	0.18+e004
Uniforme	8184	4110026	48	99.999 ± 0.001	1.17+e008

O algoritmo genético teve um comportamento satisfatório, convergindo para soluções com cobertura total das amostras (Figura 4.19) e com AMVGs reduzidos (Figura 4.20). Também observamos a evolução satisfatória da média da melhor *fitness* na Figura 4.21. Apesar de a amostragem uniforme gerar um grafo com muitos vérti-

ces e arestas, o conjunto AMVG resultante foi relativamente pequeno, como podemos averiguar na Tabela 4.6.

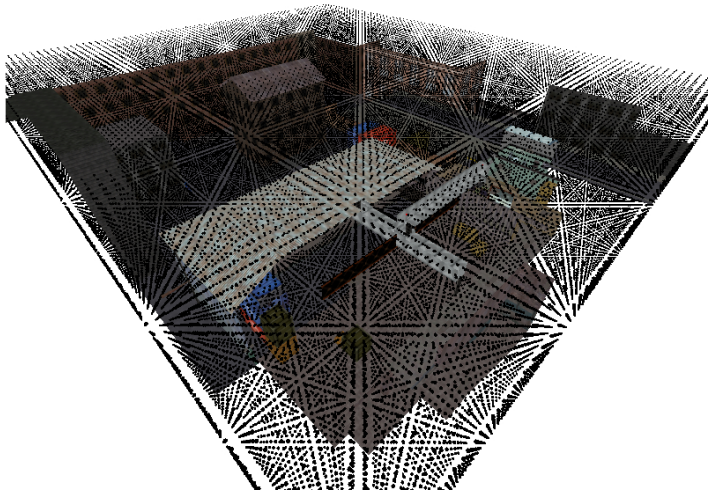


Figura 4.17. Amostragem uniforme no cenário Cs_assault.

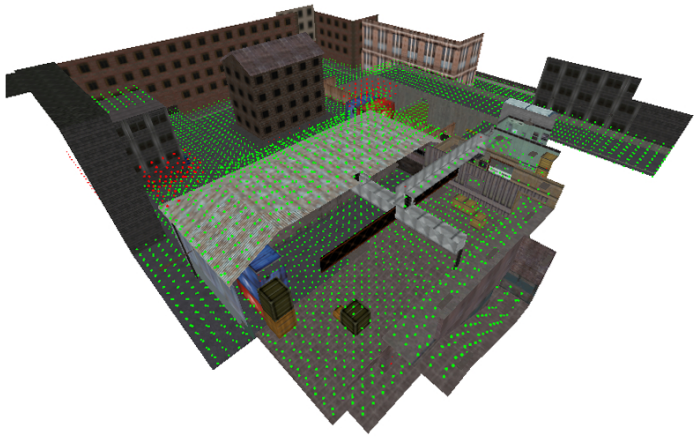


Figura 4.18. Amostras validadas no cenário Cs_assault a partir do volume de interesse.

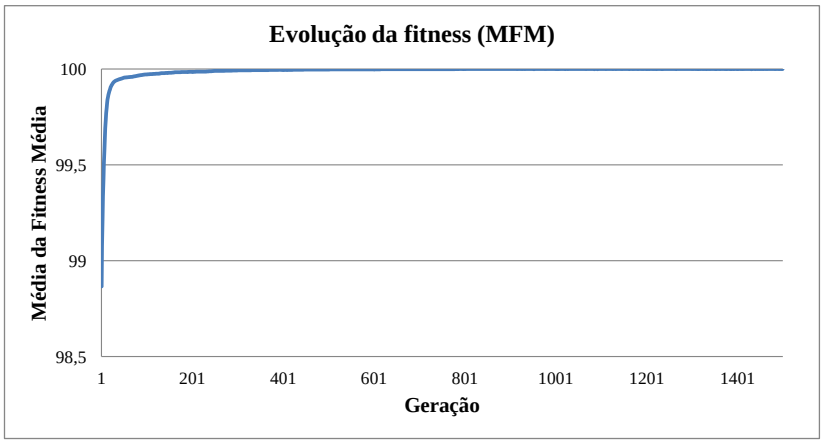


Figura 4.19. Evolução da média da fitness média do AG sobre as amostras uniformes no cenário Cs_assault.

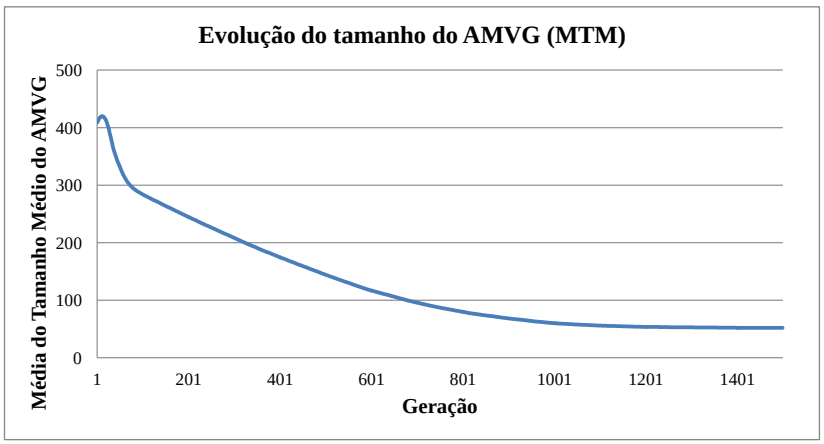


Figura 4.20. Evolução da média do tamanho médio dos AMVGs do AG sobre as amostras uniformes no cenário Cs_assault.

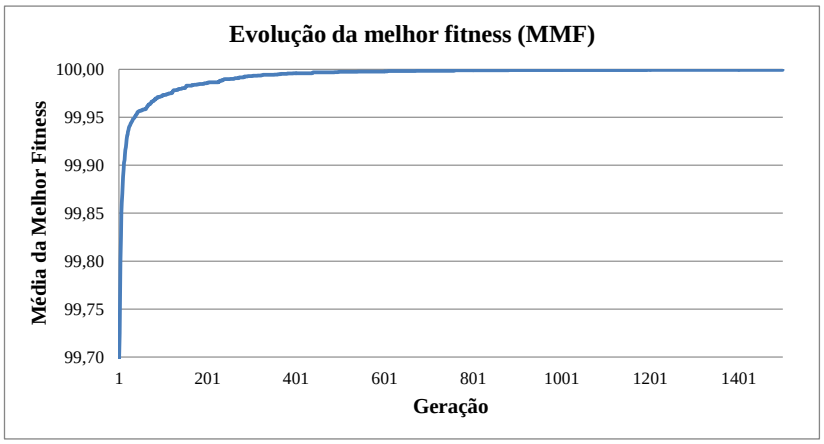
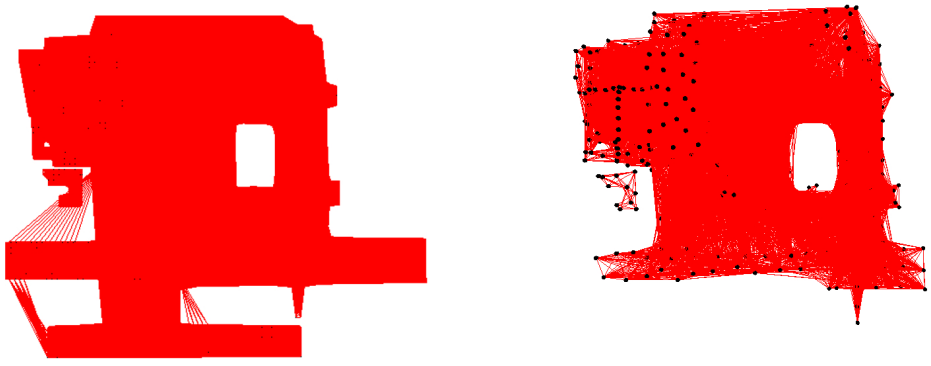


Figura 4.21. Evolução da melhor fitness do AG sobre as amostras uniformes no cenário Cs_assault.

Constatamos que o método de amostragem é uma escolha importante na metodologia. Nesse experimento foram comparados os grafos de visibilidade e os AMVGs usando amostragens distintas para diferentes aplicações num mesmo cenário. A amostragem uniforme gerou um excesso de vértices, mas permite uma cobertura mais precisa do ambiente, característica importante para o monitoramento de forma geral. Já a amostragem por *waypoints*, equivalente à distribuição de Poisson, apresentou menos amostras, mas é satisfatória para aplicação no jogo *Counter-Strike*.

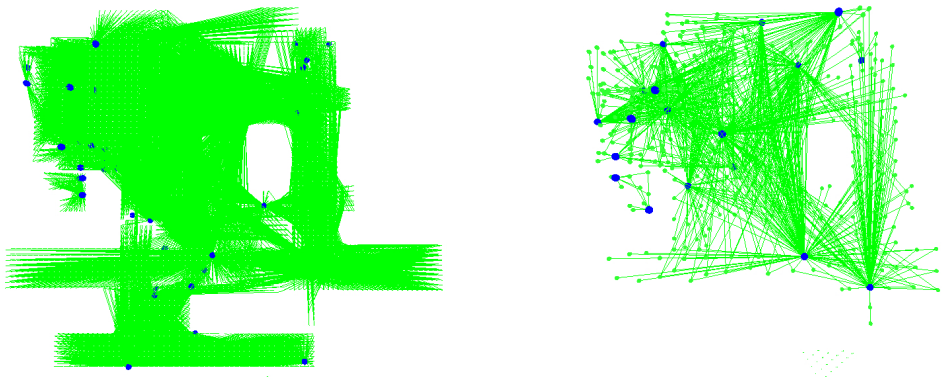
No experimento com a amostragem uniforme ainda foram criadas amostras em regiões não utilizadas pelo jogo *Counter-Strike* com o intuito de diferenciar o foco das aplicações, ou seja, trabalhamos no mapa Cs_assault como se fosse um ambiente tridimensional genérico sem nenhuma referência prévia de amostras, diferentemente do caso dos experimentos com *waypoints*. Os grafos de visibilidade da Figura 4.22 ilustram a diferença das regiões cobertas por amostras. Já os AMVGs correspondentes são mostrados na Figura 4.23, onde se percebe a cobertura total das amostras.

Resumindo, o que define qual o método de amostragem deve ser utilizado é o objetivo da aplicação. Outros, além dos métodos aqui citados também podem ser explorados em trabalhos futuros, como por exemplo, amostragens de *Hammersley* ou de *Halton Point*, os quais apresentam boa distribuição de amostras [Wong et al., 1997].



(a) (b)

Figura 4.22. Grafos de visibilidade da amostragem uniforme (a) e da amostragem por waypoints (b) do mapa Cs_assault.



(a) (b)

Figura 4.23. AMVGs da amostragem uniforme (a) e da amostragem por waypoint (b) do mapa Cs_assault.

Capítulo 5

Conclusão e trabalhos futuros

Esse trabalho apresentou uma metodologia baseada em uma abordagem genética aplicável às diversas instâncias do Problema da Galeria de Arte. Os experimentos focaram em ambientes tridimensionais arbitrários complexos, até então inexplorados pelo estado da arte.

5.1 Conclusão

Analisando os resultados dos experimentos podemos concluir que a metodologia proposta atingiu o objetivo principal desse trabalho, além de propiciar um grande leque de oportunidades para trabalhos futuros em diversas áreas de aplicação. Como vimos no Capítulo 2, a revisão do estado da arte aponta uma lacuna no Problema da Galeria de Arte em ambientes tridimensionais não planares e não ortogonais, ou seja, ambientes com essas características não podiam ser tratados.. Com esse trabalho propusemos uma metodologia capaz de solucionar, de forma aproximada, esse problema.

Sobre a metodologia, podemos afirmar que a etapa de amostragem é essencial para a qualidade da solução. Podemos ver a amostragem como uma personalização de cada aplicação, além de ser um ajuste fino de cobertura. A amostragem é o que torna a metodologia genérica e existem técnicas distintas aplicáveis a cada caso. Tendo um conjunto de amostras válidas, as características do ambiente também são utilizadas para gerar o grafo de visibilidade e, por fim, o cálculo do MVG aproximado é independente do ambiente. Seja o ambiente 2D ou 3D, planar ou não planar, ortogonal ou não ortogonal, com ou sem buracos, a metodologia que desenvolvemos é aplicável.

Além disso, podemos apontar a amostragem como uma limitação da metodologia, pois uma amostragem ruim ou inapropriada pode gerar resultados ruins. Ou seja,

algoritmo genético busca o AMVG no grafo de visibilidade gerado pelas amostras, logo, um resultado bom depende de uma amostragem boa.

Como demonstramos no Capítulo 4, o algoritmo genético pode ser ajustado por meio da função *fitness* para convergir a uma porcentagem de cobertura de amostras desejada, além de evoluir para conjuntos de observadores reduzidos. Na metodologia proposta, o algoritmo genético é o que torna o cálculo da solução aproximada computacionalmente viável, já que tratamos de um problema NP-Difícil.

Analisando aplicações reais do Problema da Galeria de Arte, percebemos que, em muitos casos, lida-se com sistemas multiagentes, informação imperfeita e processos não determinísticos oriundos de eventos aleatórios, como por exemplo o monitoramento de multidões, o posicionamento unidades militares em pleno combate, etc. Acreditamos que a aplicação de uma metodologia inicialmente no contexto de jogos, o qual oferece um domínio auto-suficiente com a simulação dessas características, é um caminho mais fácil e barato para testes, avaliações, validações e primeiras conclusões experimentais. Ao obter bons resultados, pode-se pensar na implantação e testes em escalas maiores e contextos reais.

A partir da metodologia proposta e dos resultados alcançados, diversas pesquisas podem se estender, enriquecendo ainda mais o estado da arte. Algumas sugestões seguem abaixo.

5.2 Trabalhos futuros

Nos experimentos realizados, uma solução candidata é representada por uma *string* binária do tamanho do número de vértices e um vértice guarda pode ocupar qualquer um desses pontos de amostra. Se o foco da aplicação fosse o posicionamento de câmeras de um circuito fechado de TV, por exemplo, seria necessário limitar a *string* binária aos pontos viáveis de se posicionar uma câmera. Dessa forma, uma solução candidata teria um genótipo reduzido. Caso fosse necessário restringir o alcance e o ângulo de visão de uma câmera, isso deveria ser aplicado na etapa de geração do grafo de visibilidade. Esse é um exemplo de trabalho futuro a ser desenvolvido.

Considerando a idéia de *tradeoff*, um trabalho interessante seria modificar a função de *fitness* para encontrar a cobertura máxima que um número predefinido de guardas pode proporcionar, além de identificar as suas posições no ambiente.

Aproveitando o conceito de zonas de interesse criado nesse trabalho, seria interessante desenvolver um método automático para gerar esses subvolumes, além de estudar e definir a quantidade necessária e a posição de amostras dentro de cada volume para

garantir que o mesmo tenha uma alta taxa de cobertura se todas as suas amostras forem cobertas por um vértice guarda. O excesso de amostras causa redundâncias no cálculo da cobertura final, portanto, quanto maior for a redução do número de amostras, mais rápido será o processamento do resultado.

Também seria interessante explorar outros conceitos do estado da arte dos algoritmos genéticos, buscando manter a diversidade de indivíduos por mais tempo e, assim, evitar convergências prematuras e alcançar resultados ainda mais próximos do exato. Como trabalhos futuros, seria útil explorar o cruzamento uniforme devido ao *bias* posicional e também os conceitos de nichos de espécies e *fitness sharing* em novas funções de *fitness* multimodais. Outra opção seria utilizar o conceito de dominância de pareto para objetivos múltiplos na evolução das soluções candidatas do AG.

Outro trabalho interessante de se analisar seria a utilização de uma heurística para o MSC como *baseline* e comparar os resultados, ao invés do algoritmo exato como foi feito em nossos experimentos.

Além de explorar outros métodos de amostragem em volumes ou superfícies, desenvolver uma métrica para avaliar a porcentagem de cobertura real dos volumes seria importante para o estado da arte. Em nossos experimentos foi comparada a cobertura de amostras do algoritmo genético com a de um algoritmo exato. Mas definir se uma técnica de amostragem é eficiente para representar um volume qualquer é outro problema. Mesclando o conceito de zonas de interesse e o Método de Monte Carlo (MMC), seria possível gerar amostras de pontos em posições aleatórias do espaço tridimensional, verificar se cada ponto pertence ou não ao volume a ser monitorado para validá-lo. Em seguida, testar se o conjunto aproximado de guardas cobre todas as amostras válidas e determinar a porcentagem real do volume de interesse coberto.

Implementar a metodologia desenvolvida nessa dissertação em um contexto real de aplicação do PGA, onde novos desafios possam surgir e gerar soluções que reduzam custos e ampliem benefícios.

Como é possível observar, a contribuição dessa pesquisa para o estado da arte gerou uma série de novas possibilidades de estudo envolvendo o Problema da Galeria de Arte. Sendo assim, trabalharemos para estender e publicar nosso trabalho em um periódico com o intuito de dar maior visibilidade aos avanços e contribuir para o avanço científico da área.

Referências Bibliográficas

Amit, Y.; Mitchell, J. S. B. & Packer, E. (2007). Locating guards for visibility coverage of polygons. Em *Proceedings of the Workshop on Algorithm Engineering and Experiments*, ALENEX '07, pp. 120--134. SIAM.

Bajuelos, A. L.; Canales, S.; Hernández, G. & Martins, A. M. (2008a). Minimum vertex guard problem for orthogonal polygons: A genetic approach. Em *Proceedings of the 10th WSEAS International Conference on Mathematical Methods, Computational Techniques and Intelligent Systems*, MAMECTIS '08, pp. 78--84. World Scientific and Engineering Academy and Society (WSEAS).

Bajuelos, A. L.; Canales, S.; Hernández, G. & Martins, A. M. (2008b). Optimizing the minimum vertex guard set on simple polygons via a genetic algorithm. *WSEAS Transactions in Information Science and Applications*, 5(11):1584--1596.

Ben-Moshe, B.; Katz, M. J. & Mitchell, J. S. B. (2005). A constant-factor approximation algorithm for optimal terrain guarding. Em *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pp. 515--524, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Berg, M.; Cheong, O.; Kreveld, M. & Overmars, M. (2008). *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edição.

Choe, J. (2011). *First-Person Shooter Video Game Strategy: The How-To Guide.* Vook.

Chvatal, V. (1975). A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, 18:39--41.

Couto, M. C.; de Souza, C. C. & de Rezende, P. J. (2008). Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. Em *Proceedings of the 7th International Conference on Experimental Algorithms*, WEA '08, pp. 101–113. Springer.

Efrat, A. & Har-peled, S. (2006). Guarding galleries and terrains. Em *Information Processing Letters*, pp. 181--192. Kluwer.

Eiben, A. E.; Hinterding, R. & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transations on Evolutionary Computation*, 3(2):124--141.

Fantini, E. & Chaimowicz, L. (2013). Coverage in arbitrary 3d environments: The art gallery problem in shooter games. Em *Proceedings of the XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, SBGAMES '13, pp. 146--155.

Foley, J. D.; Dam, A. V.; Feiner, S. K. & Hughes, J. F. (1995). *Computer Graphics: Principles and Practice in C.* Addison-Wesley, 2nd ed. edição.

Ghosh, S. K. (1987). Approximation algorithms for art gallery problems. Em *Proceedings of the Canadian Information Processing Society Congress*, pp. 429--434.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edição.

González-Banos, H. (2001). A randomized art-gallery algorithm for sensor placement. Em *Proceedings of the 17th Annual Symposium on Computational Geometry*, SCG '01, pp. 232--240, New York, NY, USA. ACM.

Goss, M. E. & Wu, K. (2000). Study of supersampling methods for computer graphics hardware antialiasing. Em *HP Laboratories Technical Report.* Hewlett-Packard Laboratories, Palo Alto, CA, USA.

Honsberger, R. (1976). *Mathematical Gems II.* Mathematical Association of America.

King, J. (2006). A 4-approximation algorithm for guarding 1.5-dimensional terrains. Em *In Proceedings of 7th Latin American Symposium on Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pp. 629–640. Springer.

Lee, D. T. & Lin, A. K. (1986). Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276--282.

Machado, M. C.; Fantini, E. P. C. & Chaimowicz, L. (2011a). Player modeling: Towards a common taxonomy. Em *In the 16th International Conference on Computer Games*, CGAMES '11, pp. 50–57. IEEE.

Machado, M. C.; Fantini, E. P. C. & Chaimowicz, L. (2011b). Player modeling: What is it? how to do it? (tutorial). Em *Proceedings of the X Simpósio Brasileiro de Jogos e Entretenimento Digital*, number 5 in SBGAMES '11.

Marengoni, M.; Draper, B. A.; Hanson, A. R. & Sitaraman, R. (2000). A system to place observers on a polyhedral terrain in polynomial time. *Image and Vision Computing*, 18(10):773–780.

Michael, T. S. (2009). *How to guard an art gallery and other discrete mathematical adventures.* Johns Hopkins University Press, Baltimore, MD, USA.

Millington, I. & Funge, J. (2009). *Artificial Intelligence for Games.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edição.

Nüchter, A.; Surmann, H. & Hertzberg, J. (2003). Planning robot motion for 3d digitalization of indoor environments. Em *In Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, pp. 222--227.

Nilsson, B. J. (2005). Approximate guarding of monotone and rectilinear polygons. Em *In Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pp. 1362–1373. Springer.

O'Rourke, J. (1987). *Art Gallery Theorems and Algorithms.* Oxford University Press, Inc., New York, NY, USA.

Suffern, K. (2007). *Ray Tracing from the Ground Up.* A. K. Peters, Ltd., Natick, MA, USA.

Urrutia, J. (2000). Art gallery and illumination problems. Em Sack, J. R. & Urrutia, J., editores, *Handbook of Computational Geometry.* Elsevier, North-Holland, Netherlands.

van den Bergen, G. (2003). *Collision Detection in Interactive 3D Environments.* The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann.

Viglietta, G. (2012). Guarding and searching polyhedra. *Computing Research Repository (CoRR)*, abs/1211.2483. Ph.D. thesis, University of Pisa.

Wong, T.-T.; Luk, W.-S. & Heng, P.-A. (1997). Sampling with hammersley and halton points. *Journal of Graphics Tools*, 2(2):9--24.

Worman, C. & Keil, J. M. (2007). Polygon decomposition and the orthogonal art gallery problem. *International Journal of Computational Geometry and Applications*, 17(2):105–138.

Apêndice A

Jogos aplicáveis em experimentos

A pesquisa na área de inteligência artificial para jogos é um tópico relativamente novo e multidisciplinar. Por esse motivo, diversos pesquisadores que hoje trabalham na área de video games já atuaram em outras áreas e usualmente aplicam seus conhecimentos adquiridos também nos jogos [Machado et al., 2011a].

Pensando em gerar uma lista de plataformas de jogos aplicáveis em pesquisas para a computação, algo difícil de se encontrar de forma organizada na literatura especializada, fizemos uma revisão em vários trabalhos que utilizaram jogos de gêneros distintos em suas bases experimentais, como será melhor apresentado a seguir. Essa revisão foi publicada como parte de um trabalho sobre modelagem de jogadores, uma técnica emergente de inteligência artificial [Machado et al., 2011a]. A lista estendida foi apresentada em um tutorial no SBGAMES [Machado et al., 2011b] e outra lista com *links* para *download* foi disponibilizada no *website*[1] do Laboratório Multidisciplinar de Pesquisa em Jogos.

A.1 Tipos de plataformas

Existem três maneiras convencionais para acessar a mecânica de jogos e utilizá-la em experimentos, sendo elas: *open source*, *game modifications* e *emulators*. *Open source* é um tipo de jogo que disponibiliza o seu código fonte. Muitas vezes tais códigos são distribuídos livremente e, em alguns casos, são multiplataforma. *Game modifications* ou *mods* são camadas de conteúdo aplicadas sobre um jogo existente, incluindo novos gráficos, itens, áudio e inteligência artificial. Geralmente os mods são criados sobre jogos comerciais com kits de desenvolvimento disponibilizados pelo fabricante.

[1] Disponível em: `<http://www.j.dcc.ufmg.br/platforms.html>`

Já os emulators são jogos que simulam um jogo original com comportamentos bastante semelhantes. Usualmente são criados para copiar jogos de sistemas obsoletos ou, simplesmente para criar uma cópia quase fiel de jogos com código fechado e, assim, disponibilizar o código alternativo para a comunidade de desenvolvedores [Machado et al., 2011b].

A.2 Lista de plataformas por gênero

A.2.1 Jogos de ação

O *Counter-Strike Source* é um jogo FPS multijogadores desenvolvido pela *Valve Software* usando o motor *Source*. Seu SDK, na linguagem C++, está disponível para desenvolvedores que desejam criar *mods* ou *bots* personalizados. Versões antigas do jogo também possuem SDK, como é o caso do *Counter-Strike 1.6* com a *Half-Life I Engine*.

O *Quake II* é um jogo de FPS desenvolvido pela *id Software*. Atualmente é um open source e está disponível para download no *ftp* da *id Software* assim como diversos outros jogos da empresa.

A.2.2 Jogos de aventura

O *Adventure Game Studio* é um motor multiplataforma que inclui um editor de fácil compreensão, com gráficos similares às séries *Sierra* e *Lucas Arts* dos anos 90. O motor permite a criação de scripts em C# para programação de lógica e importação de áudio e vídeo.

A.2.3 Jogos de plataforma

O *Infinite Mario Bros* é um jogo *open source* na linguagem Java que imita o clássico *Super Mario Bros*. Ele foi criado para competições de programação, como a *Mario AI Championship*. Outro jogo similar é o *Secret Maryo Chronicles* em C++, também *open source*.

Open Sonic é outro jogo *open source*, mas com plataforma 2D que imita o clássico *Sonic The Hedgehog* da *Sega*. Novos itens e inimigos podem ser criados a partir de *Object Scripts*, ou simplesmente alterando o código fonte, na linguagem C.

A.2.4 Jogos de interpretação

O *Baldur's Gate* é um jogo de interpretação de personagens lançado pela *BioWare* e desenvolvido com o motor *Infinity*. Existem diversas ferramentas não oficiais, tais como *Near Infinity*, *Dialog Checker* e *Weidu*, utilizadas para personalizar o jogo com *mods*.

Neverwinter Nights é outro jogo de interpretação lançado pela mesma empresa. Para a criação de *mods* existe a ferramenta oficial *Neverwinter Toolset*, que permite modificações em diversos aspectos do jogo. Utiliza script próprio, o *NWScript*.

RunUO é um servidor de jogo de interpretação massivo *open source* que permite conexão do jogo *Ultima Online*. O *RunUO* utiliza a linguagem C# e oferece uma total personalização dos aspectos do jogo.

A.2.5 Jogos de simulação

O *Flight Gear* é um projeto *open source* que oferece um sofisticado *framework* para simulações realísticas de voo. Utiliza a linguagem C++ e algumas personalizações podem ser feitas por *plugins* disponibilizados pelo desenvolvedor.

O *TORCS* é um simulador de corridas 3D *open source* com um sistema sofisticado de física. Utiliza as linguagens C/C++ e modificações na mecânica do jogo podem ser feitas pelo código fonte. Costuma ser utilizado em competições de computação natural.

A.2.6 Jogos de esporte

O *Super Tux Kart* é um jogo de corrida *open source*, similar ao *Super Mario Kart*. Novas configurações de veículos e *plugins* de inteligência artificial podem ser aplicados. A linguagem utilizada é o C++.

A.2.7 Jogos de estratégia

O *Civilization IV* é um jogo de estratégia baseado em turnos desenvolvido pela *Firaxis Games*. Uma interface XML permite a personalização de diversos parâmetros do jogo. Outra forma de incluir modificações é por meio do *Civ4 SDK*.

ORTS é um motor de jogo de estratégia em tempo real implementado com uma arquitetura cliente/servidor. A inteligência artificial de *bots* do jogo pode ser desenvolvido em C++. Essa plataforma foi usada por muitos anos na *ORTS Game AI Competition* promovida pela *AIIDE*.

O *StarCraft* é um jogo de estratégia em tempo-real desenvolvido pela *Blizzard Entertainment*. Para esse jogo existe uma *API* não oficial chamada *BWAPI* utilizada para a criação de agentes inteligentes personalizados. Essa *API* é usada na *StarCraft AI Competition*. A segunda versão desse jogo possui uma ferramenta oficial para a criação de *mods* chamada *StarCraft II Editor*.

O *Wargus* é uma imitação do jogo *Warcraft II*, um jogo de estratégia em tempo real também desenvolvido pela *Blizzard*. *Wargus* é um projeto *open source* organizado em dois níveis: o motor *Stratagus* com a mecânica do jogo e uma camada de maior nível para criar a lógica do jogo, a qual utiliza a linguagem Lua.

Esperamos que essa revisão de jogos utilizados em pesquisas possa contribuir para trabalhos futuros da comunidade de computação. Em alguns casos perde-se muito tempo buscando uma opção para a base de experimentos em estudos na área de jogos. Sugerimos, de forma complementar, a leitura das seções sobre plataformas nos artigos Machado et al. [2011a, 2011b] para mais exemplos e para a consulta de referências utilizadas nessa revisão.

Apêndice B

Utilizando o SDK do Half-Life I

A *Valve*, empresa desenvolvedora do jogo *Half-Life I*, disponibilizou um conjunto de bibliotecas para acessar as rotinas principais do jogo, permitindo a criação de modificações como o *Counter-Strike* e também o desenvolvimento de agentes virtuais que utilizam o *Half-Life I* como jogo base.

O SDK do *Half-Life I* é utilizado por todos os sistemas de *bots* disponíveis para o *Counter-Strike*. O *E[POD]-Bot 5.3* é um desses sistemas, aplicável tanto no Windows quanto no Linux, com código fonte em C/C++ disponível para *download*[1]. Por ser um sistema de fácil manuseio, ele foi utilizado em nossas pesquisas.

B.1 Instalação do E[POD]-Bot

A instalação é simples. Além do código fonte do sistema de *bots*, citado acima, é necessário possuir o *Counter-Strike*, que pode ser adquirido pelo gerenciador de jogos *Steam*[2].

Após instalar o *Counter-Strike* é necessário executar o instalador[3] do *E[POD]-Bot* e indicar a pasta do conteúdo do jogo `"...\ProgramFiles\Steam\steamapps\common\Half-Life\cstrike"`.

Na pasta do código fonte há um projeto do *Microsoft Visual Studio C++* chamado *EPB.sln* que compila o sistema de *bots* e gera um arquivo DLL que será utilizado como *plugin* pelo *Counter-Strike*. Ao executar o jogo, o *E[POD]-Bot* já estará ativo.

[1]Disponível em: <http://filebase.bots-united.com/uploads/epbv5.3src.zip>
[2]Disponível em: <http://media.steampowered.com/client/installer/SteamSetup.exe>
[3]Disponível em: <http://filebase.bots-united.com/uploads/EPBv5.3.exe>

B.2 Extração de dados com o E[POD]-Bot

Como todos os sistemas de *bots* para o *Counter-Strike* usam os *waypoints* predefinidos para os mapas, foi utilizado o código fonte do *E[POD]-Bot* para acessar as informações necessárias para os experimentos.

No *E[POD]-Bot* há um editor de *waypoints*, assim como um conjunto de rotinas para gerenciá-los. Também é possível acessar rotinas do motor do jogo *Half-Life I*, como o *hit-testing*.

Sendo assim, desenvolvemos um novo comando de console para o *Counter-Strike*, o qual extrai a lista de *waypoints* dos mapas com as coordenadas correspondentes, gera e exporta o grafo de visibilidade entre os *waypoints* por meio do *Ray-Mesh Intersection Test* (rotina *TRACE_LINE* no motor do *Half-Life I*). O comando criado também salva a lista de *camp points* de cada mapa.

Com esses dados extraídos para arquivos foi possível usá-los como entradas para os experimentos relacionados ao *Counter-Strike*. O Algoritmo B.1 ilustra o método desenvolvido para efetuar as extrações de informações referentes aos *waypoints*.

```
1  for i ← 0 to NumWaypoints − 1 do // percorre waypoints
2      origin ←= waypointCoord[i]; // ponto inicial
3      for j ← 0 to NumWaypoints − 1 do
4          destiny ← waypointCoord[j]; // ponto final
5          collision ← traceLine(origin, destiny); // ray-mesh intersection
           test
6          if !collision then
7              print(FileGraph, 1); // gera aresta
8          else
9              print(FileGraph, 0); // não gera aresta
10     print(FileGraph, "\n");
11     print(FileCoords, waypointCoord[i]); // salva coordenadas
12     if waypointFlag[i] = CAMP_POINT then // é camp point?
13         print(FileCamps, i); // salva camp points
14 return;
```

Algoritmo B.1: Algoritmo para extração de dados no Counter-Strike.

De modo geral, *mods* de jogos são bastante personalizáveis e podem ser usados em diversos experimentos nas áreas de inteligência artificial, computação gráfica e geometria computacional.

O *E[POD]-Bot* é um sistema de *bots* completo, disponibilizando aos agentes virtuais todas as funcionalidades de um jogador real do *Counter-Strike*, tais como se

esconder, buscar por objetivos, conversar via *chat*, etc. Portanto, é um sistema útil e aplicável a um grande leque de opções de pesquisas em jogos.