UFC

UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FRANCISCO CAIO MAIA RODRIGUES

EVOLUÇÃO DE SPLIT GRAMMARS PARA OTIMIZAÇÃO DE CONSTRUÇÕES PROCEDURAIS

FORTALEZA, CEARÁ

2014

FRANCISCO CAIO MAIA RODRIGUES

EVOLUÇÃO DE SPLIT GRAMMARS PARA OTIMIZAÇÃO DE CONSTRUÇÕES PROCEDURAIS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Computação Gráfica

Orientador: Prof. Dr. Joaquim Bento Cavalcante Neto

Coorientador: Prof. Dr. Creto Augusto Vidal

FORTALEZA, CEARÁ

2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

Rodrigues, Francisco Caio Maia.
Evolução de split grammars para otimização de construções procedurais / Francisco Caio Maia Rodrigues. – 2014.
49 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2014.
Orientação: Prof. Dr. Joaquim Bento Cavalcante Neto.
Coorientação: Prof. Dr. Creto Augusto Vidal .

1. Modelagem procedural. 2. Algoritmo genético . 3. Split grammars. I. Título.

CDD 005

FRANCISCO CAIO MAIA RODRIGUES

EVOLUÇÃO DE SPLIT GRAMMARS PARA OTIMIZAÇÃO DE CONSTRUÇÕES PROCEDURAIS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Computação Gráfica

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. Joaquim Bento Cavalcante Neto
Universidade Federal do Ceará (UFC)
Orientador

Prof. Dr. Creto Augusto Vidal
Universidade Federal do Ceará (UFC)
Coorientador

Prof. Dr. Emanuele Marques dos Santos
Universidade Federal do Ceará (UFC)

Prof. Dr. Manuel Menezes de Oliveira Neto
Universidade Federal do Rio Grande do Sul
(UFRGS)

RESUMO

Modelagem procedural tem sido aplicada com sucesso para resolver o problema da construção automática de ambientes urbanos. Dentre as várias técnicas existentes para a geração procedural de construções utilizando gramáticas, *Split Grammars* têm especial destaque devido ao seu amplo uso, estando presente até mesmo em softwares comerciais de CAAD (Computer-Aided Architectural Design). Este trabalho propõe uma técnica para otimização de Split Grammars utilizando algoritmos genéticos. O objetivo é gerar, automaticamente, gramáticas capazes de criar modelos que apresentem alguma característica desejada, seja a partir de uma série de gramáticas feitas manualmente por um usuário ou de gramáticas geradas aleatoriamente. O método proposto realiza uma busca no espaço das regras das gramáticas dadas como entrada a fim de criar novos tipos de gramáticas melhores, ou seja, que possuam uma boa estrutura de acordo com algum critério pré-definido pelo usuário. Assim, é demonstrada a eficácia da técnica proposta aplicando-a ao problema de maximização de simetria em fachadas de construções, obtendo modelos realisticamente plausíveis.

Palavras-chave: Modelagem Prodecural. Split Grammars. Algoritmos Genéticos.

ABSTRACT

Procedural modeling has been successfully applied to the automatic building generation problem. Among several techniques to tackle the problem of procedural building generation, the use of Split Grammars has increased, even being deployed in commercial CAAD (Computer-Aided Architectural Design) software. This work proposes a technique to optimize Split Grammars using Genetic Algorithm. The main goal is to automatically create grammars that only generate models with certain desirable characteristics, either from a series of manually written grammars or randomly created ones. The proposed thecnique searches the space of the input grammars's rules to develop new better grammars, i.e., grammars that generate models with certain pre-defined feature. The proposed technique was successfully applied, as will be shown, to the maximization of symmetry of building facades, leading to the creation of realistic models.

Keywords: Procedural Modeling. Split Grammars. Genetic Algorithm.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Modelagem procedural é o nome dado a uma gama de técnicas de geração de modelos virtuais com base na aplicação de algoritmos. Neste paradigma, cenas ou modelos tridimensionais podem ser gerados após a aplicação de uma série de regras paramétricas, o que permite que formas complexas sejam representadas por estruturas compactas. Essa abordagem foi aplicada com sucesso para modelar certos objetos como plantas, terrenos, nuvens e ambientes urbanos (MUSIALSKI et al., 2013), conforme mostram as Figuras 1.1 e 1.2.

Percebe-se que modelagem procedural pode ser bem aplicada para representar estruturas que apresentem características como repetições, autossimilaridade ou aleatoriedade. Uma grande vantagem sobre a modelagem manual é o seu potencial de escalabilidade: inúmeros modelos diferentes podem ser criados variando-se apenas alguns parâmetros ou pequenos conjuntos de regras. Por exemplo, uma floresta inteira pode ser projetada a partir da aplicação de algumas dezenas de regras com parâmetros diferentes, enquanto que a modelagem manual desse tipo de cenário seria muito lenta e, portanto, financeiramente mais cara.

Figura 1.1: Exemplo de vegetação gerada proceduralmente usando um L-System. As regras para a geração deste modelo foram: $X \rightarrow F-[[X]+X]+F[+FX]-X$) e ($F \rightarrow FF$). *http://wikipedia.org/*

O crescimento da disponibilidade de dados sobre ambientes urbanos tem impulsionado pesquisas e o desenvolvimento de softwares que tratam de simulação de áreas urbanas. Simular ambientes urbanos é uma tarefa importante em um grande número de aplicações, por

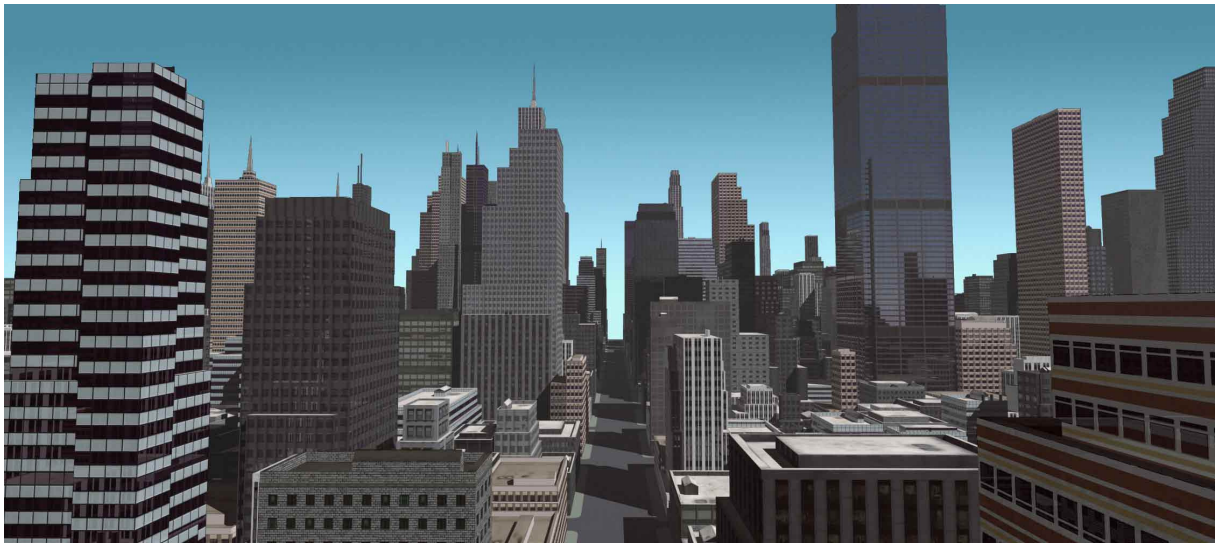


Figura 1.2: Cidade criada proceduralmente utilizando o método proposto em (PARISH; MüLLER, 2001)

exemplo:

Planejamento urbano: Para coordenar mudanças nos espaços urbanos ou visualizar impactos causados por novas construções, é interessante dispor de ferramentas para rápida e barata prototipação desse tipo de ambiente.

Entretenimento: A geração detalhada de cidades inteiras de maneira rápida e realista pode ser importante no desenvolvimento de jogos e filmes, que buscam cada vez mais a interatividade de imersão.

Logística: Inúmeras situações importantes podem se beneficiar de representações fidedignas de um espaço urbano tais como: treinamento de socorro em casos de emergências, análise de rotas, planos de evacuação, mapeamento de tráfego, entre outras.

Em certas aplicações, para que se faça algum tipo de simulação ou visualização sobre uma área urbana, é necessário um modelo virtual tridimensional da mesma. A produção deste modelo consiste basicamente em gerar a rede de ruas e estradas, os quarteirões delimitados por essas ruas e, por fim, as construções formadas por um contorno geométrico e sua fachada (VANEGAS et al., 2010). A Figura 1.3 apresenta um exemplo de como algoritmos podem ser empregados para facilitar a visualização dos resultados de simulações sobre ambientes urbanos.

Neste sentido, um exemplo interessante é mostrado em (KUNZE et al., 2011), que apresenta um *framework* para planejamento urbano participativo. Este *framework* foi apresentado e testado em 2010, no *workshop "World Cup 2014 Urban Scenarios"*, em Porto Alegre. Como resultado, foi exposto o *design* urbano da cidade esperado para os anos 2014, 2025 e 2050. Além da modelagem geométrica das cinco áreas urbanas escolhidas, simulações de acessibilidade, crescimento populacional e desemprego foram elaboradas. Para a criação dos modelos virtuais das construções foi usado o software *CityEngine* (PROCEDURAL, 2008), que utiliza técnicas de modelagem procedural (Figura 1.4 e Figura 1.5).

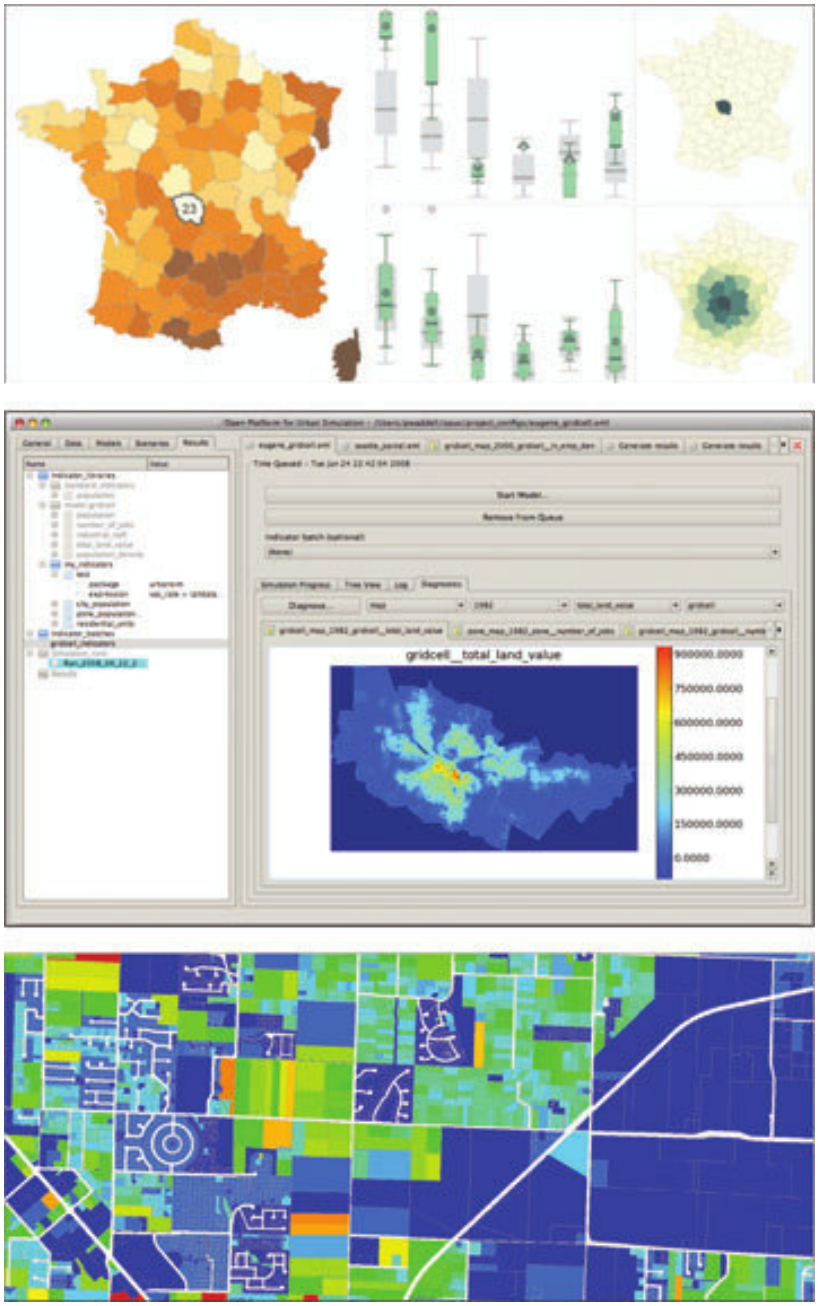


Figura 1.3: Exemplo de visualização de uma simulação urbana (VANEGAS et al., 2010).

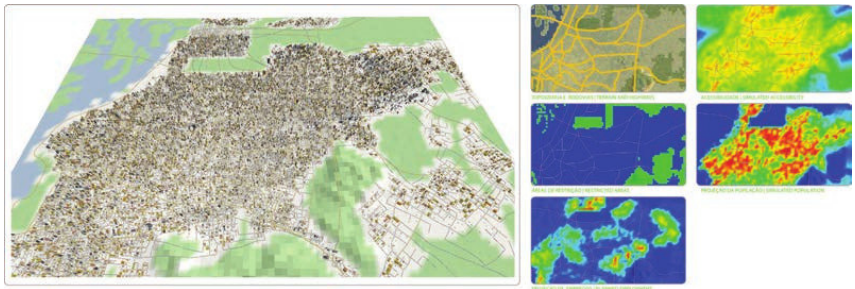


Figura 1.4: Visualização de Porto Alegre em 2025 e o resultado de simulações de acessibilidade, crescimento populacional e nível de desemprego (KUNZE et al., 2011).

Figura 1.5: Visualização de Porto Alegre em 2050 (KUNZE et al., 2011).

1.2 Objetivos

No contexto da geração de construções para representação do modelo virtual de uma área urbana, dois tipos de técnicas são comumente aplicadas: a reconstrução a partir de imagens ou nuvens de pontos e a geração procedural. O foco do trabalho aqui apresentado é a geração de modelos procedurais de construções, tratando apenas a representação das fachadas utilizando gramáticas.

O principal objetivo deste trabalho é apresentar uma técnica para evoluir *Split Grammars* (WONKA et al., 2003), a fim de realizar otimizações em fachadas de construções procedurais utilizando Algoritmos Genéticos (AG).

1.3 Justificativa

O uso de *Split Grammars* para representação de fachadas teve bastante sucesso, sendo inclusive utilizado no software comercial City Engine (PROCEDURAL, 2008), além de ter sido amplamente explorado no meio acadêmico. Por apresentar uma estrutura altamente regular, a representação de fachadas por gramáticas se mostrou muito eficiente.

Como apontado por (BAO et al., 2013), inúmeros problemas de geração automática de modelos procedurais podem ser formulados como problemas de otimização ou amostragem probabilística. Neste contexto, algoritmo genético é uma ótima ferramenta para realizar uma busca no espaço de soluções para problemas de otimização. O modo como uma busca combina pedaços de soluções, através de algoritmos genéticos, para gerar outras soluções melhores é apropriado para o problema da busca de um layout de fachadas procedurais, que são basicamente formadas por uma dada configuração de blocos, tais como janelas, portas, paredes, etc.

Pelos motivos apresentados nos parágrafos anteriores, a escolha pela utilização de *Split Grammars* e algoritmos genéticos para explorar a linguagem de gramáticas que descrevem fachadas parece natural e é o que é proposto neste trabalho.

1.4 Organização do Trabalho

Este texto está organizado da seguinte maneira:

- **Capítulo 2 - Background:** Neste capítulo, são apresentados os conceitos básicos utilizados na abordagem do problema. O capítulo está dividido em 3 partes. A primeira, sobre Modelagem Procedural de fachadas, discute os princípios da geração de modelos de fachadas através de um conjunto de regras utilizando *Split Grammars*. A segunda parte introduz Algoritmos Genéticos e mostra como essa heurística pode ser utilizada para solucionar problemas de otimização. Já a última parte mostra as principais estruturas de dados que foram implementadas para representação da geometria e das gramáticas.

- **Capitulo 3 - Trabalhos Relacionados:** Neste capítulo, é apresentada a evolução do estado da arte do problema de geração automática de modelos procedurais de construções. Mostram-se outras abordagens, sejam elas evolutivas ou não, para gerar os prédios de forma procedural que sigam certos critérios pré-determinados.

- **Capítulo 4 - Técnica Proposta:** Neste capítulo, é apresentada a técnica proposta, que utiliza Algoritmo Genético para a evolução de gramáticas de modelagem procedural de fachadas. É descrita a implementação dos operadores do AG: geração da população inicial de gramáticas, combinação de gramáticas para gerar novas soluções (*crossover*) e a mutação.

- **Capítulo 5 - Resultados:** Neste capítulo, são apresentados os resultados obtidos pela aplicação da técnica proposta em dois problemas de exploração de *layout* envolvendo simetria. A modelagem matemática da função *fitness*, utilizada pelo AG para caracterizar boas soluções, para estes exemplos é apresentada, assim como a emergência da característica desejada a partir de gramáticas aleatórias. Por fim, são discutidas as vantagens e as desvantagens da aplicação do método nestes problemas.

- **Capítulo 6 - Conclusões e Trabalhos Futuros:** Neste capítulo, apresentam-se as conclusões sobre o trabalho proposto e discutem-se possíveis problemas que poderiam ser resolvidos utilizando a técnica proposta.

2 BACKGROUND

2.1 Modelagem Procedural de Fachadas com Split Grammars

Conforme indicado no Capítulo 1, este trabalho faz uso de *Split Grammars* para modelar a estrutura de fachadas. *Split Gammars* são gramáticas livres de contexto usadas para modelagem procedural de construções originalmente apresentadas em (WONKA et al., 2003) e, posteriormente, estendidas pelo trabalho de (MüLLER et al., 2006), que desenvolveu uma gramática para modelagem de construções chamada *CGA shape grammar*.

Uma *Split Grammar* é uma tupla $G = \langle N, T, \sigma, R \rangle$, onde N é o conjunto de símbolos não-terminais, T o conjunto dos símbolos terminais, $\sigma \in R$ é a regra inicial e R é o conjunto de regras. Cada símbolo terminal $\tau \in T$ está associado a um valor semântico usado para identificar o elemento da fachada que ele representa, como janela, parede, coluna, etc.

As regras $r \in R$ têm a forma

$$LHS \rightarrow RHS, \tag{2.1}$$

em que $LHS \in N$ é um único símbolo não-terminal e RHS é uma operação *split* que aplicará as regras *sucessoras* de LHS.

Uma operação *split* divide uma região geométrica horizontal ou verticalmente e tem a seguinte forma:

$$split(axis) \rightarrow (repeat)\{\mu_0 s_0 : r_0, \ldots, \mu_n s_n : r_n\}, \tag{2.2}$$

onde *axis* representa o eixo em que a região do espaço será dividida; o símbolo *repeat* é um modificador de regra opcional que indica se as divisões devem ser aplicadas novamente, caso haja espaço sobrando (a Figura 2.2 exemplifica o uso deste modificador de regra); s_i representa o tamanho da divisão a ser aplicada na geometria que resultará em uma nova região retangular, sobre a qual será aplicada a produção $r_i \in (N \cup T)$ e μ_i é o modificador da divisão. Esse modificador pode ser de três tipos:

Absoluto: Cria uma nova região de tamanho extamente igual a s_i ao longo do eixo selecionado. Se não houver espaço suficiente, a nova região não será criada. Regras que possuem apenas divisões com esse tipo de modificador podem ser problemáticas, pois é possível que uma região da fachada não seja associada a nenhum símbolo terminal, caracterizando um modelo de fachada inválido. A vantagem deste tipo de *split* é que o usuário sabe exatamente o tamanho das regiões criadas enquanto define a regra.

Aproximado: Representado na gramática por $\sim$, este modificador força a nova região a ter tamanho pelo menos s_i. Se, ao fim de todas as divisões da operação *split* ainda houver espaço sobrando, as regiões marcadas com este modificador aumentarão de tamanho por um número proporcional a s_i, de modo que toda a região passada como entrada para a regra seja ocupada. Este tipo de modificador garante que todas as regiões da fachada estejam associadas a algum símbolo terminal.

Relativo: Seja t o tamanho do *shape* inicial, o tamanho da nova região criada será $s_i * t$. Para que toda a região sobre a qual a regra é aplicada seja utilizada, é preciso garantir que a soma dos tamanhos do split com modificadores relativos seja igual a $1,0$.

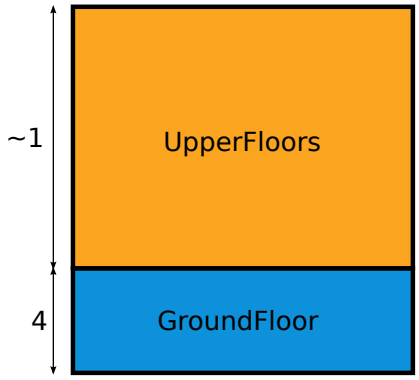


Figura 2.1: Exemplo de *split* com o modificador aproximado (~). A regra que leva a essa derivação é: $Facade \rightarrow split(Y)\{4 : GroundFloor, \sim 1 : UpperFloors\}$.

Em uma única operação *split* é possível que cada divisão possua um modificador diferente. A Figura 2.1 mostra um exemplo de uso dos modificadores absoluto e aproximado em uma mesma operação, enquanto que a Figura 2.2 mostra o uso do *repeat*.

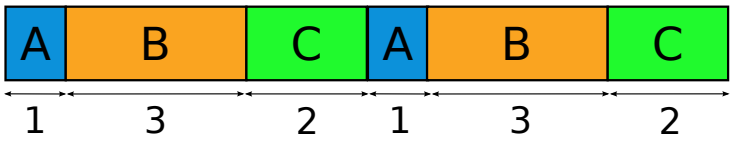


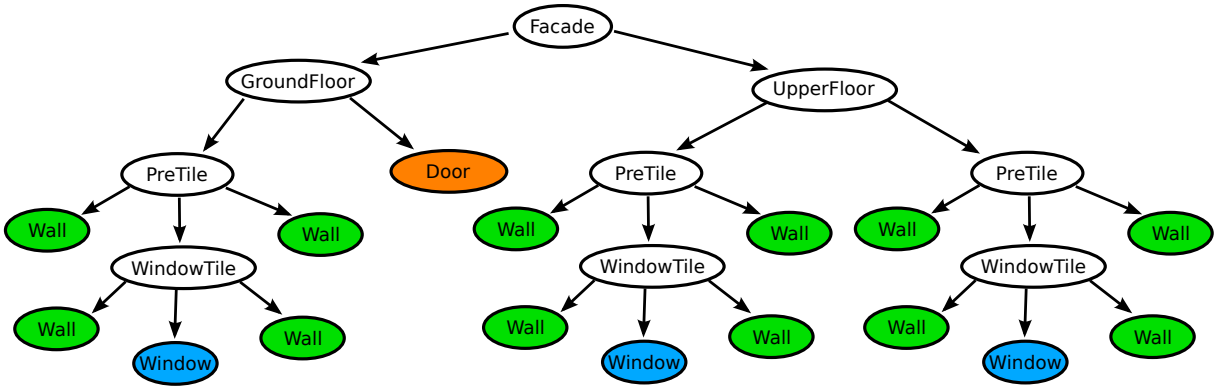
Figura 2.2: Exemplo de *split* com repetição. Regra que leva a esse tipo de configuração: $S \rightarrow repeat\ split(X)\{1 : A, 3 : B, 2 : C\}$.

Uma fachada é representada por uma *árvore de derivação* de uma palavra de uma dada gramática. Nesta árvore, cada nó associa uma região retangular do espaço a um símbolo da gramática $\phi \in (T \cup N)$, de modo que os nós intermediários particionam essa região pela aplicação de uma regra $\eta \in R$ e os nós folhas representam os elementos atômicos que compõem a fachada (janelas, paredes, colunas, etc), identificados pela semântica dos símbolos terminais $\tau \in T$ associados a ele. A Figuras 2.3a representa a árvore de derivação correspondente à fachada da Figura 2.3b.

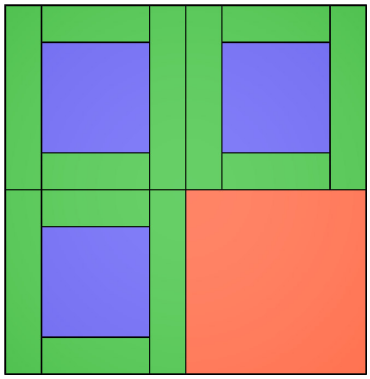
A Figura 2.4 exemplifica a aplicação das seguintes regras em uma região retangular qualquer:

$$PreTile \rightarrow split(X)\{1 : wall, \sim 2 : Tile, 1 : wall\},$$
$$WindowTile \rightarrow split(Y)\{1 : wall, \sim 2 : window, 1 : wall\}.$$

A Figura 2.4b mostra como ficou a região retangular após a aplicação da regra $PreTile$. As regiões da cor verde foram associadas ao símbolo terminal *wall* e a região branca será passada para a regra $WindowTile$. A Figura 2.4c mostra o resultado da aplicação da regra



(a) Árvore de derivação.



(b) Fachada.

Figura 2.3: Exemplo de uma fachada e sua árvore de derivação.

WindowTile, que resulta na adição de dois símbolos terminais do tipo *wall* e um símbolo terminal *window*. Por fim, a Figura 2.4d mostra a árvore resultante da aplicação destas duas regras.

2.2 Algoritmos Genéticos

Algoritmo Genético (AG) é uma heurística de busca de soluções baseada no processo de seleção natural, geralmente aplicado em problemas de otimização. Seu funcionamento é inspirado em conceitos biológicos, tais como herança genética, seleção, mutação e recombinação. A técnica foi desenvolvida nos anos 1960 e 1970 por John Holland, em meio ao surgimento de uma série de outros métodos inspirados em evolução (MITCHELL, 1998).

Algoritmo genético realiza uma busca no espaço de soluções, combinando partes delas para encontrar resultados melhores. A ideia é que boas soluções sejam formadas por bons fragmentos de solução que, se combinados entre si, resultam em respostas melhores. Em geral, algoritmo genético é utilizado quando o espaço de soluções é muito grande, sendo inviável uma busca por força bruta, e é possível medir a qualidade das soluções propostas ao problema, usando essa função de classificação de soluções para guiar a busca.

O processo de busca no espaço de soluções ocorre de maneira iterativa. A cada passo (*geração*), um conjunto de soluções (*indivíduos*) é analisado e, a cada elemento deste conjunto,

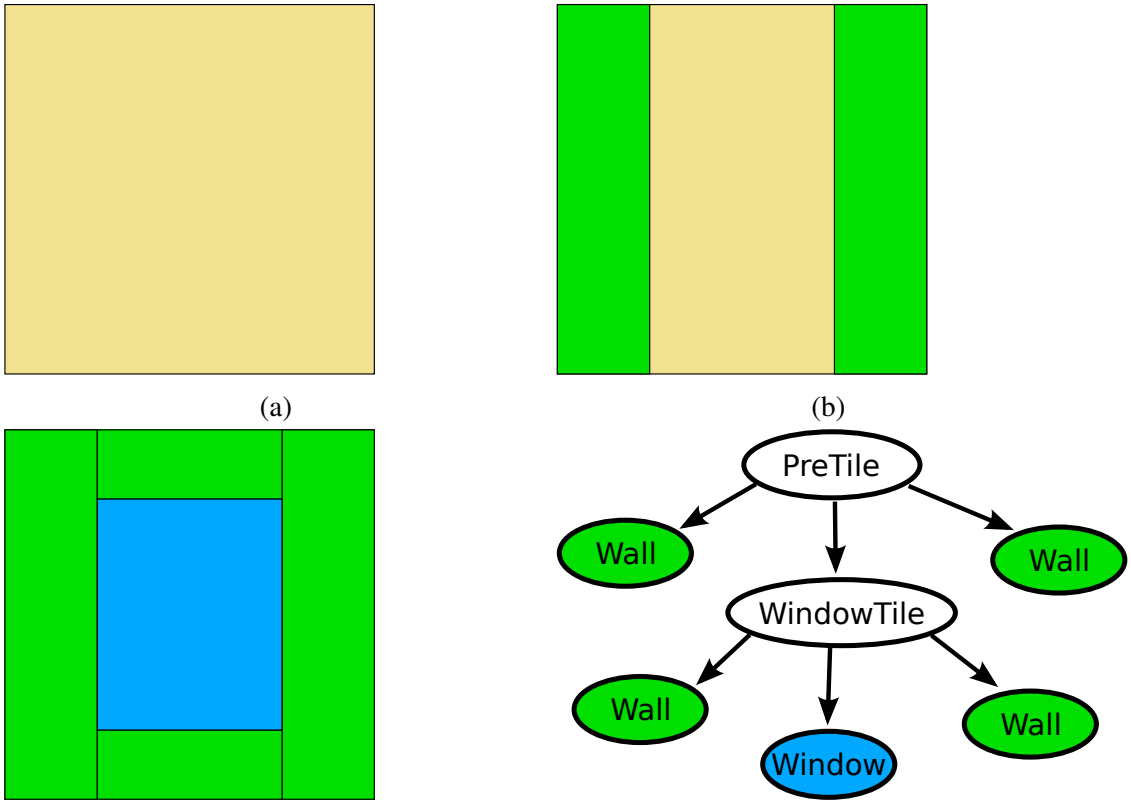


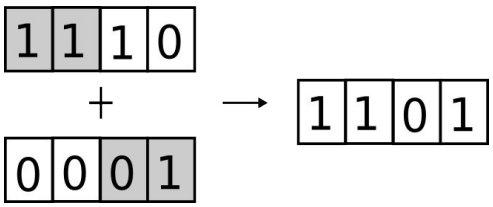
Figura 2.4: Passo a passo da aplicaçao de duas regras em uma região retangular e a árvore de derivação gerada por este processo.

é atribuido um valor chamado de *fitness*, que indica o quanto a solução é boa para o problema. Os indivíduos são geralmente representados como vetores de *bits*. No entanto, qualquer outro tipo de estrutura pode ser utilizada para representá-los, desde que os operadores do AG sejam implementados de acordo com o tipo de representação usada. Para que seja criada uma nova geração, utiliza-se o processo de *seleção*. Nesta etapa, os indivíduos da população são escolhidos aleatoriamente para gerar a nova população, de modo que as melhores soluções, ou seja, aquelas que tenham maior *fitness*, tenham maior probabilidade de serem escolhidas. Para criar os novos indivíduos são aplicados os operadores do AG, que funcionam de maneira análoga aos processos biológicos de reprodução, como descritos a seguir:

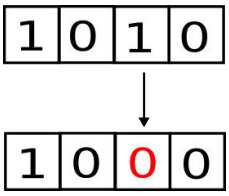
- **Crossover:** Este operador combina dois indivíduos previamente escolhidos pela etapa de seleção para que novas soluções sejam criadas. Os "filhos"serão formados por pedaços dos "pais", ou seja, herdarão suas características. Como o processo de seleção em geral escolhe indivíduos bons, espera-se que o filhos possuam valores *fitness* bons, se herdarem as boas caracteristicas dos ascendentes, seguindo a ideia de que boas soluções podem ser formadas por bons pedaços de solução. A Figura 2.5a ilustra esse processo de aplicação de um *crossover* sobre indivíduos representados por vetores de *bits*.

- **Mutação:** Este operador modifica um indivíduo da população, alterando alguma de suas características escolhidas aleatoriamente. O objetivo da aplicação deste operador é adicionar pequenas perturbações na representação dos indivíduos, de modo a evitar que

a busca por soluções fique concentrada em uma região de máximo ou mínimo local. A Figura 2.5b mostra a aplicação de uma mutação em um indivíduo representado por um vetor de *bits*.



(a) Crossover.



(b) Mutação.

Figura 2.5: Operadores de algoritmos genéticos.

A seguir é apresentado o funcionamento básico de um algoritmo genético para operar em indivíduos representados por vetores de *bits*:

1. Começa com uma população inicial (provavelmente gerada de maneira aleatória).
2. Calcula o *fitness* $f(x)$ de cada elemento desta população.
3. Enquanto um valor de *fitness* suficientemente bom não tenha sido encontrado ou o número máximo de iterações ainda não tenha sido alcançado, repita os seguintes passos:
   (a) Selecione um par de elementos da população aleatoriamente, com probabilidade proporcional ao *fitness*.
   (b) Decida, de acordo com uma probabilidade p_c, se haverá cruzamento entre os pares A e B. Seja A_i o i-ésimo gene de A no cromossomo. Escolha um ponto k aleatoriamente no cromossomo onde haverá a recombinação e gere dois filhos. O primeiro será formado por $A_1,...,A_k,B_{k+1},...,B_n$ e o segundo será $B_1,...,B_k,A_{k+1},...,A_n$.
   (c) Aplique uma mutação, com probabilidade p_m, aos dois filhos gerados no passo anterior e coloque-os na nova população.
   (d) Calcule o *fitness* dos novos indivíduos criados.
   (e) Substitua a população atual pela nova população.
4. Retorna população atual.

O algoritmo básico apresentado acima é representado na Figura 2.6.

Uma das grandes dificuldades de se aplicar algoritmo genético para a resolução de problemas é modelar matematicamente uma boa função objetivo. Sem uma função objetivo

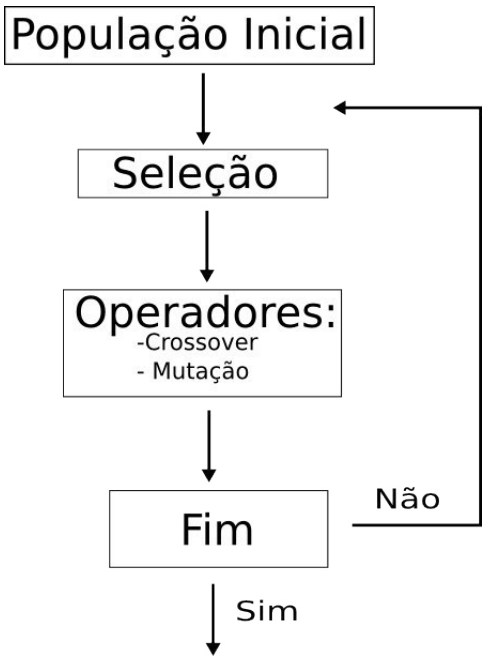


Figura 2.6: Fluxograma de um algoritmo genético.

capaz de capturar bem as alterações no *fitness*, o que o AG faz é muito próximo de uma busca aleatória no espaço de soluções, o que torna o método ineficiente.

2.3 Estruturas de Dados

Para representar geometricamente as fachadas, utilizou-se a estrutura de dados *Half-Edge*. Dentre os principais motivos para a sua escolha está a naturalidade da implementação de operações *split*, feita utilizando *Operadores de Euler*. Além disso, o uso de half-edge proporciona grande facilidade para estender a gramática utilizada com outros operadores e formas de modelagem, como o operador de extrusão e permite representar uma construção inteira, cujas fachadas são apenas as faces laterais do modelo, tal como apresentado em (MüLLER et al., 2006).

O texto apresentado nesta seção é um pequeno resumo sobre a estrutura de dados *Half-Edge* e sobre os Operadores de Euler, os quais foram utilizados para a implementação do método proposto. Para uma melhor referência sobre esses tópicos indica-se a leitura do livro (MANTYLA, 1988), que serviu como base para todo o restante desta seção.

Half-edge é uma estrutura de dados baseada em aresta comumente utilizada para modelagem de sólidos, representando-os pela fronteira do sólido, ou seja, as faces. Por ser baseada em aresta, entende-se que as faces que compõe a fronteira do sólido são representadas por uma sequência fechada de arestas. Por serem o elemento principal da estrutura, é nas arestas que se armazenam as informações de conectividade e vizinhança dos elementos, muito importantes para simplificar a implementação de algoritmos que alterem o modelo a ser representado. Nesta estrutura, uma aresta é representada por duas *half-edges*, cada uma associada a um dos vértices da aresta que é o seu ponto de partida. Como cada aresta pertence ao mesmo tempo a duas faces,

cada uma das suas *half-edges* representa a direção da aresta em uma das face. A Figura 2.7 descreve visualmente o funcionamento da estrutura.

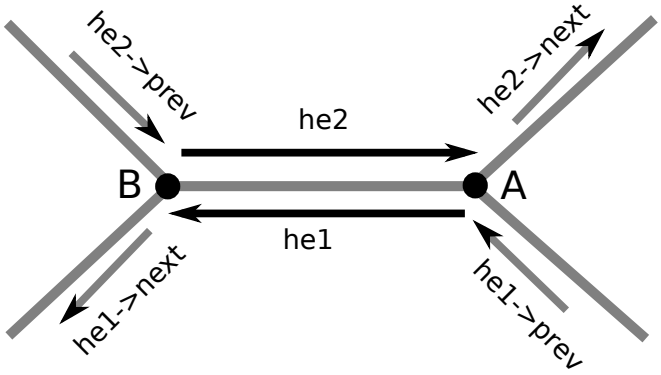


Figura 2.7: Visão esquemática da half-edge.

Apenas três tipos básicos de operações são necessárias para descrever todos os modelos planos:

1. Uma operação para criação de uma primitiva que represente o modelo plano mais básico possível.
2. Duas operações topológicas que dividem as sequências de arestas de uma face ou de um vértice.
3. Uma operação topológica que implementa a soma conectada de dois polígonos, não necessariamente do mesmo modelo.

Operadores de Euler são a realização algoritmica dessas operações teóricas sobre modelos planos. Estes operadores atuam de modo a garantir a validade da fórmula de Euler-Poincaré após a aplicação de uma operação qualquer. Seja V o número de vértices, E o número de arestas, F o número de faces, S o número de sólidos, H o número de buracos e R o número de *rings*, a equação de Euler-Poincaré diz que:

$$V - E + F = 2(S - H) + R. \tag{2.3}$$

Uma breve descrição de alguns dos principais operadores é realizada abaixo. Outros operadores de Euler não serão apresentados neste texto por não terem relevância para a implementação do trabalho proposto nesta dissertação.

- **Make Vertice Face Solid (mvfs):** Responsável pela criação do sólido inicial, esta operação adiciona um vértice, uma face e um sólido à estrutura de dados (Figura 2.8). A inversa desta operação é a *Kill Vertice Face Solid (kvfs)*.
- **Make Edge Vertex (mev):** Divide o ciclo de arestas de um vértice em dois, de modo que parte das arestas que partiam deste vértice passarão a ter o vértice recém criado como uma de suas extremidades. Entre o vértice antigo e o novo é criada uma aresta. Essa operação

adiciona um vértice e uma aresta à estrutura (Figura 2.9). A operação inversa ao *mev* é um *Kill Edge Vertex (kev)*, que remove uma aresta e um vértice que era um dos extremos desta aresta.

- **Make Edge Face (mef):** Esta operação une dois vértices por uma aresta formando uma face. Adiciona uma aresta e uma face à estrutura de dados (Figura 2.10). A operação inversa ao *mef* é a *Kill Edge Face (kef)*.

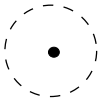
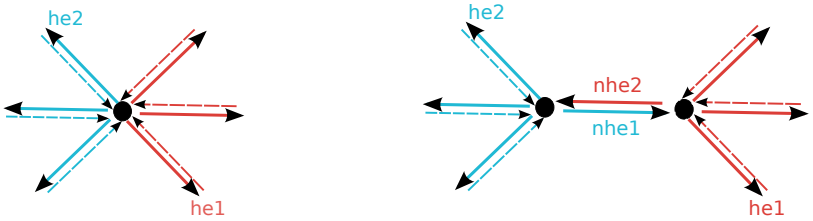
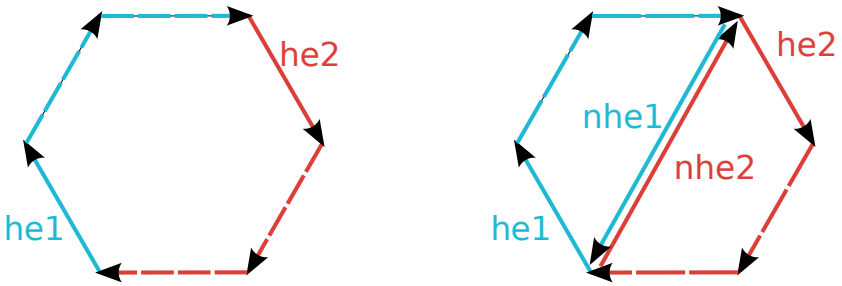


Figura 2.8: Operador *mvfs*.



(a) (b)

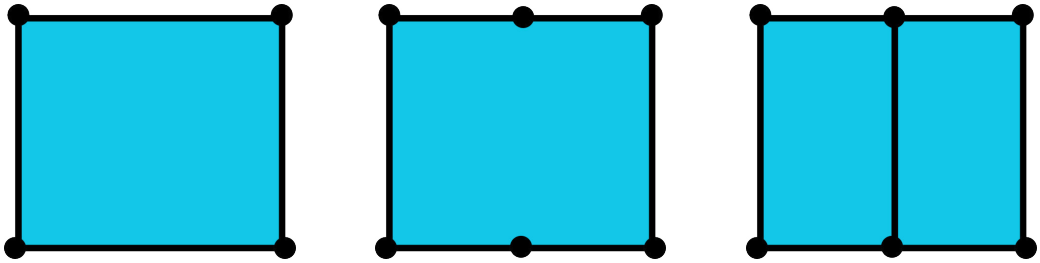
Figura 2.9: Operação *mev*.



(a) (b)

Figura 2.10: Operação *mef*.

De especial interesse para a implementação da operação *split*, necessária à criação de modelos utilizando a gramática descrita na Seção 2.1, são os operadores *mev* e *mef*. A Figura 2.11 mostra os passos para implementar uma operação *split* utilizando operadores de Euler.



(a) Face incial. (b) Após a aplicação de 2 *mev*s. (c) Aplicação do operador *mef*.

Figura 2.11: Operação split implementada utilizando os operadores de Euler.

3 TRABALHOS RELACIONADOS

Neste capítulo, é apresentada uma visão geral de técnicas de modelagem procedural de construções, quer elas usem gramáticas ou não, que, de algum modo, aplicam técnicas de otimização para explorar o espaço de construções procedurais. Por ser uma área nova de pesquisa, ainda não existem muitos trabalhos nessa linha. Os trabalhos discutidos foram classificados da seguinte maneira:

Aprendizado de Gramática Técnicas de aprendizado de gramática buscam, a partir de uma série de gramáticas determinísticas, inferir uma gramática probabilística capaz de gerar a mesma saída que as gramáticas da entrada, além de outras. Espera-se com essa gramática mais genérica explorar a linguagem definida pelas gramáticas da entrada, podendo derivar modelos parecidos com os apresentados.

Exploração de *designs* Técnicas deste tipo realizam buscas no espaço de modelos procedurais para criar variações de modelos dados como entrada que mantenham aspectos desejáveis.

Modelagem Procedural Inversa Técnicas de modelagem procedural inversa buscam obter uma representação procedural para um conjunto de modelos. A idéia é, dado um modelo, inferir uma gramática, geralmente determinística, capaz de derivar o mesmo modelo.

Foram deixados de fora deste texto técnicas que realizam a exploração de *layouts* por amostragem probabilística de regras procedurais e métodos que otimizam construções sujeitas a simulações físicas.

3.1 Aprendizado de Gramática

O trabalho de Talton et al. (2012) recebe uma série de modelos criados manualmente e induz uma gramática probabilística capaz de criar novos tipos de modelos, diferentes dos fornecidos na entrada. Para tanto, o trabalho faz uso de uma técnica chamada de *Bayesian Model Merging*, já aplicada com sucesso para resolver o problema de indução de gramáticas para processamento de linguagem natural.

A técnica proposta no artigo foi aplicada para a criação de modelos geométricos baseados em grafos de cena, ou seja, o modelo é formado por uma árvore de primitivas e transformações geométricas aplicadas de maneira hierárquica entre os nós desta árvore. O algoritmo proposto pode ser resumidamente explicado pelos seguintes passos:

1. Criar uma gramática G, tal que cada um dos n modelos da entrada possam ser gerados por G com probabilidade $1/n$.

2. Aplicar Cadeias de Markov Monte Carlo para explorar um grande banco de regras que possam ser adicionadas a G para torná-la mais genérica.

(a) Merge: dados dois símbolos não terminais $X \rightarrow X_0, \ldots, X_n$ e $Y \rightarrow Y_0, \ldots, Y_n$, criar um novo símbolo não terminal $Z \rightarrow X_0, \ldots, X_n, Y_0, \ldots, Y_n$. De modo que uma referência a X ou a Y será substituída por uma referência a Z.

(b) Split: oposto do Merge, dado um símbolo não-terminal $X \rightarrow X_0, \ldots, X_n$, dividí-lo em um ponto aleatório i tal que dois novos símbolos não-terminais sejam criados $X_a \rightarrow X_0, \ldots, Xi$ e $X_b \rightarrow X_{i+1} \ldots, X_n$ de modo que, onde antes havia uma referência a X, passa a existir uma referencia a X_a ou X_b, escolhido de maneira aleatória.

3. Avaliar nova gramática criada pelo passo 2 usando inferencia Bayesiana. Calcular a probabilidade dessa nova gramática gerar os modelos da entrada bem como o seu tamanho, sendo priorizadas gramáticas menores, ou seja, com menor número de regras.

4. Voltar para o passo 2 até que um número máximo de passos já tenha sido executado ou uma gramática bem avaliada tenha sido obtida.

A técnica descrita por Martinović e Van Gool (2013) é muito similar ao método discutido acima. A principal diferença entre estes trabalhos está no tipo de gramática utilizada e os modelos gerados, já que este trabalho utilizou *Split Grammars* com o objetivo de modelar fachadas de construções. O mesmo algoritmo descrito na seção anterior se aplica para a explicação deste método, com uma notável diferença: a entrada para o método são imagens de fachadas, então para obter as gramáticas capazes de gerar os modelos iniciais, é necessário realizar uma etapa de *parsing* no espaço da imagem. Além disto, o algoritmo proposto é mais próximo da técnica originalmente usada para indução de gramáticas de processamento de linguagem natural pois não utiliza a operação split, explicada acima.

A Figura 3.1 mostra o resultado obtido por Martinović e Van Gool (2013) aplicado a fachadas reais. A técnica foi aplicada em duas das fachadas dessa imagem, sendo possível ver os diferentes *layouts* obtidos.

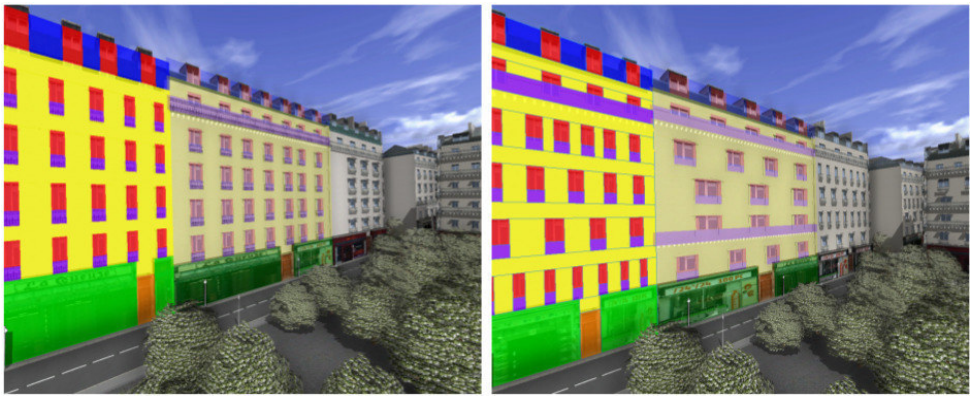


Figura 3.1: Exemplo de resultado da técnica de Martinović e Van Gool (2013).

3.2 Exploração de *designs*

Bao et al. (2013) apresenta um método para caracterizar o espaço de construções procedurais que satisfazem certos critérios definidos pelo usuário. Neste trabalho, as construções são representadas como a união de caixas, possivelmente sobrepostas, $\bigcup R_i$. Cada caixa R_i é representada por um vetor de 6 números: $[l_i, w_i, h_i, x_i, y_i, \theta_i]$, onde $(l_i, w_i e h_i)$ representam, respectivamente, o comprimento, a largura e a altura da caixa, (x_i, y_i) representa a sua posição e θ_i sua orientação. Logo, uma construção pode ser vista como um vetor $\Gamma \in R^d (d = 6n)$. A técnica proposta no artigo funciona da seguinte maneira:

1. Uma população inicial de boas construções, ou seja, modelos que satisfazem uma série de restrições escolhidas pelo usuário, é gerada utilizando *simulated annealing*. Além de gerar proceduralmente os modelos iniciais, existe a possibilidade de o usuário definir manualmente o conjunto inicial de construções.

2. Dado um modelo representado por Γ_i, aplicam-se perturbações na forma de vetores $v \in R^d, \|v\| = 1$, a fim de se criarem variações $\Gamma_j = \Gamma_i + v$. Tais vetores são escolhidos de modo que eles não violem as restrições impostas aos modelos. Dadas as variações, o espaço local de construções de Γ_i é formado.

3. É construído um grafo ligando os espaços locais criados no passo anterior, tal que as arestas conectam os modelos cujas distâncias são menores.

Apesar do método levar a resultados interessantes, a representação dos modelos é demasiadamente simplificada, não necessitando de gramáticas para representar as construções. O conjunto de caixas sobrepostas representa apenas a massa de construções, deixando de lado a representação das fachadas.

A Figura 3.2 mostra o tipo de modelo utilizado por Bao et al. (2013). É fácil notar as diferentes caixas sobrepostas formando o modelo. Já na Figura 3.3 é mostrado a evolução dos modelos ao longo das iterações do algoritmo de otimização.

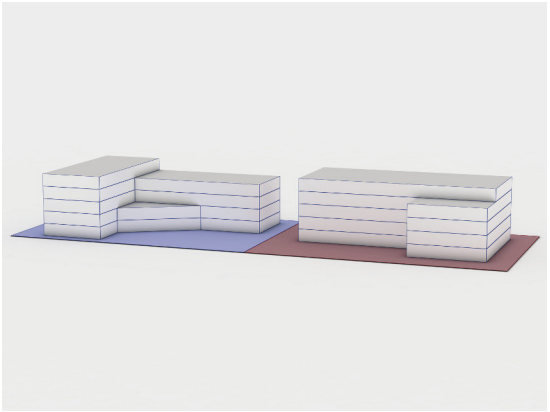


Figura 3.2: Exemplo de construções utilizadas por Bao et al. (2013).

Um dos passos propostos por (SIMON et al., 2011) envolve a exploração do espaço de gramáticas, de modo semelhante ao descrito acima. Porém, foram utilizadas *Split Grammars*.

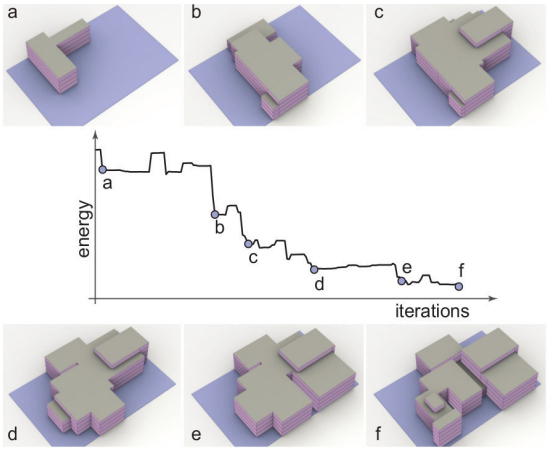


Figura 3.3: Otimização aplicada por Bao et al. (2013) ao longo das iterações.

Para tanto, as regras são modeladas como vetores de d dimensões e, assim como a técnica de (BAO et al., 2013), estes vetores são perturbados numa direção aleatória. Para guiar a busca é utilizada uma função objetivo que calcula a energia da gramática utilizando a imagem da fachada por ela gerada. A Figura 3.4 mostra a evolução dos modelos de fachada ao longo do processo de minimização de energia aplicado. Nota-se a semelhança com o método de (BAO et al., 2013).

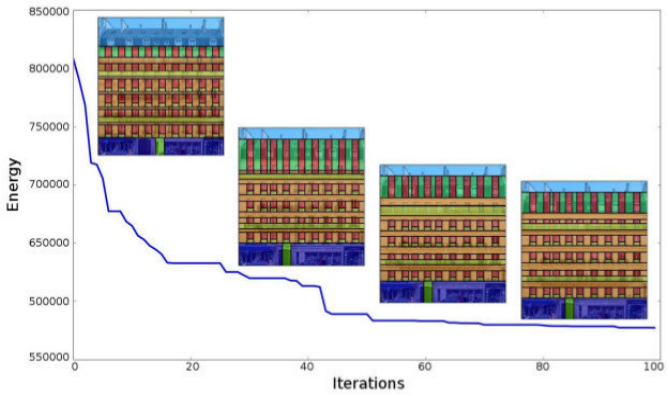


Figura 3.4: Otimização aplicada por (SIMON et al., 2011) evoluindo os elementos das fachadas.

3.3 Modelagem Procedural Inversa

O trabalho apresentado por (ZHANG et al., 2013) busca uma representação procedural de fachadas a partir de imagens segmentadas. Motivado por teorias sobre percepção cognitiva, é apresentado uma nova forma de estruturar fachadas, utilizando camadas além dos *splits*. No trabalho é apresentada uma ferramenta para realizar manualmente a segmentação das imagens em elementos atômicos da fachada, como janelas e portas. Em seguida, estes elementos são agrupados em regiões retangulares de modo que elementos geometricamente iguais pertençam à mesma região. Por fim, o método busca uma representação hierarquica ótima dos grupos de elementos, buscando uma decomposição da fachada que maximize a simetria das divisões de toda a região da fachada entre os agrupamentos de elementos. Para resolver o problema de otimização foi utilizado um algoritmo genético, cuja função objetivo busca caracterizar simetria reflexiva

entre os grupos de elementos. O artigo apresenta argumentos relacionados à percepção visual para motivar o uso da simetria, mostrando que é uma característica presente em construções reais. Este trabalho, entretanto, faz uso apenas de simetria reflexiva, deixando de explorar a simetria translacional, facilmente observada em projetos arquitetônicos.

A Figura 3.5 mostra o processo resumido acima. Na imagem da esquerda percebe-se a imagem de entrada a segmentação da mesma em regiões retangulares que representam os elementos da fachada. Na imagem da direita é possível ver os *splits* e *layers* escolhidos pelo algoritmo, de forma maximizar a simetria.

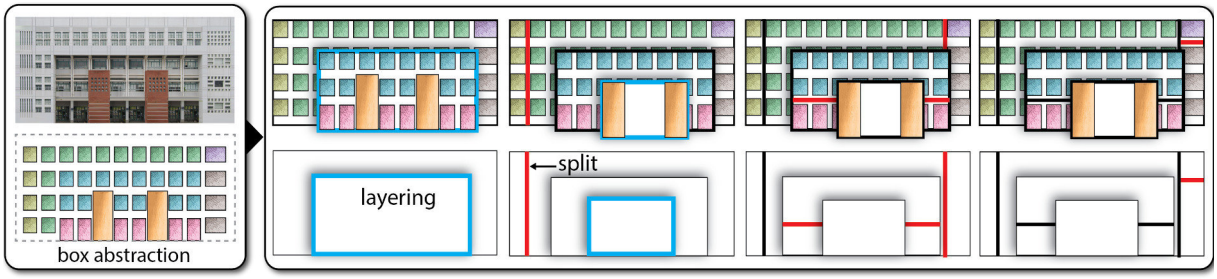


Figura 3.5: Visualização do processo aplicado por (ZHANG et al., 2013).

Outro trabalho que propõe uma solução ao problema de modelagem procedural inversa é apresentado por (WU et al., 2014). Inspirado em técnicas de processamento de linguagem natural, o problema é modelado como uma busca pela menor gramática. A cada regra é atribuído um custo que depende do seu tipo e tamanho. Para calcular o custo de uma gramática, somam-se os custos de todas as regras. Para resolver a otimização foi utilizado programação dinâmica aproximada. Esta técnica é um pouco similar à técnica anterior: a entrada para o algoritmo são imagens segmentadas descrevendo os elementos terminais da fachada e a busca pelas divisões também utiliza uma medida de simetria para achar os melhores *splits*. A saída desta técnica é uma *Split Grammar* descrevendo a fachada passada como entrada.

A Figura 3.6 mostra o resultado da aplicação do método. As regras que descrevem uma parte selecionada da fachada são mostradas, assim como a imagem de entrada e as regiões segmentadas dos elementos da fachada.

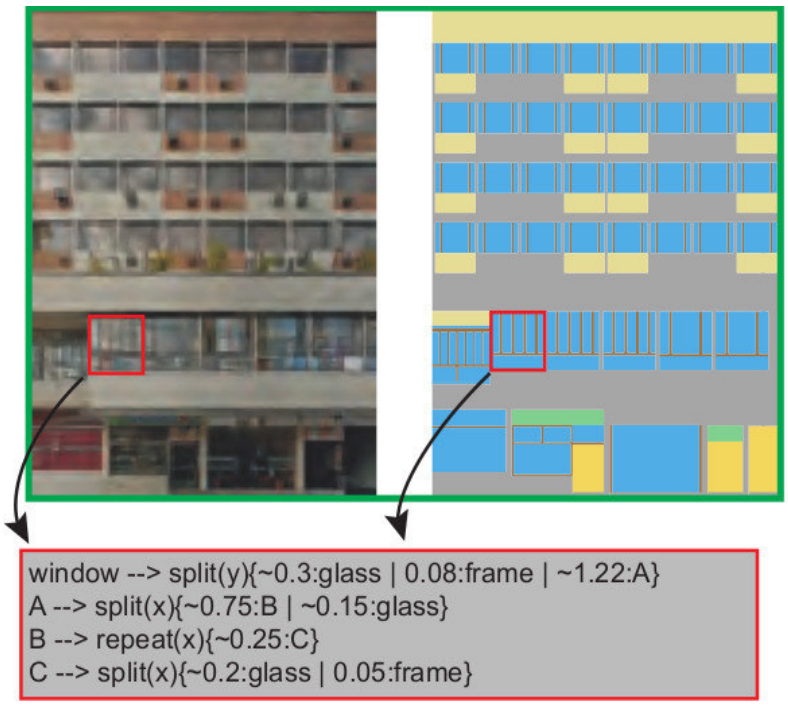


Figura 3.6: Resultado do método de (WU et al., 2014).

4 TÉCNICA PROPOSTA

Neste capítulo é apresentada a técnica proposta para otimização de *Split Grammars* usando algoritmos genéticos. Conforme descrito no Capítulo 2, os principais procedimentos realizados pelo algoritmo genético são a criação da população inicial, o crossover, a mutação e a aplicação da função *fitness* para avaliar os indivíduos da população. Neste capítulo, será apresentado como estes procedimentos foram adaptados para serem usados em *Split Grammars*.

4.1 Visão Geral

Para evoluir *Split Grammars*, os indivíduos da população serão gramáticas, tais quais as demonstradas no Capítulo 2, ou seja, uma lista de regras, sendo uma delas a regra inicial. As operações do algoritmo genético atuarão sobre as regras destas gramáticas, como será explicado nas próximas seções.

A entrada para o algoritmo de otimização é formada pela população inicial (pop), o número de indivíduos desta população ($popsize$), o número máximo de iterações ($numiter$) e o valor de *fitness* que se pretende alcançar ($fmax$) . O algoritmo será executado até que uma das seguintes condições seja satisfeita: o número máximo de iterações for alcançado ou o melhor indivíduo da população possua *fitness* maior que ou igual a $fmax$.

A cada iteração $popsize/2$ novos indivíduos são criados pela aplicação de *Crossovers* e adicionados à população. Dessa população de $3popsize/2$ indivíduos, apenas os $popsize$ melhores, ou seja, os que possuem o maior *fitness*, passarão para a próxima iteração e os outros $popsize/2$ piores indivíduos serão descartados. Cada indivíduo criado pela aplicação do *Crossover* está sujeito a sofrer uma mutação com probabilidade $mprob$.

O Algoritmo 4.1 apresenta o pseudo-código do método descrito acima. Nas próximas seções serão detalhados os procedimentos para a criação da população inicial, o *Crossover*, a *Mutação* e, por fim, o cálculo do *fitness*.

4.2 População incial

A população inicial é gerada de maneira aleatória. O algoritmo recebe quatro valores ($xmin, ymin, xmax, ymax$), que definem os pontos mínimo e máximo da região retangular da fachada, e usa-os para calcular certas medidas que serão utilizadas para definir as regras da gramática, como o número e a altura do andares e o tamanho das divisões dentro de um andar.

A função começa definindo a altura (h), que será um valor aleatório no invervalo $[3.0, 5.0]$. O número de andares ($nfloors$) é dado por: $nfloors = \lfloor (ymax - ymin)/h \rfloor$. Em seguida, $nfloors + 1$ regras são criadas: a regra inicial $Facade$, que dividirá a região original da fachada no eixo Y em $nfloors$ partes, e as regras $Floor_1, Floor_2, ..., Floor_{nfloors}$. A regra $Facade$ terá a seguinte forma:

Algorithm 4.1 Evolução da população

```
1:  function EVOLUI(pop, popsize, numiter, fmax)
2:      for i ← 1 até popsize do
3:          pop[i].fitness ← Fitness()
4:      for i ← 1 até numiter do
5:          for j = 1 até popsize/2 do
6:              Indiv a ← SELECIONA_INDIV(pop)
7:              Indiv b ← SELECIONA_INDIV(pop)
8:              Indiv novoa, novob ← Crossover(a, b)
9:              if prob < mprob then
10:                 MUTATE(novoa)
11:             if prob < mprob then
12:                 MUTATE(novob)
13:             novoa.fitness ← Fitness()
14:             novob.fitness ← Fitness()
15:             ADICIONA_INDIV(a, b)
16:         ORDENA_DECRESCENTE(pop)
17:         REMOVE_INDIV(pop[popsize + 1], ..., pop[3popsize/2])
18:         if pop[0].fitness ≥ fmax then return fim
19:     return fim
```

$$Facade \rightarrow split(Y)\{s_1 : Floor_1, \ldots, s_n : Floor_n floors\}. \tag{4.1}$$

Para gerar a regra $Floor_i$ de cada andar, é sorteado um número inteiro aleatório no intervalo $[1,3]$ de divisões no eixo X, cada uma de tamanho aleatório, de modo que a soma de todos estes pedaços seja igual a $xmax - xmin$. Cada uma dessas partes será então dividida em partes menores, com regras terminais para paredes e janelas. Na regra do primeiro andar, existe uma pequena diferença em relação às outras, já que alguma das divisões deverá conter, obrigatoriamente, um símbolo terminal de semântica *porta*. A forma da regra $Floor_i$ é a seguinte:

$$Floor_i \rightarrow split(X)\{s_1 : Div_1, \ldots, s_j : Div_j\}. \tag{4.2}$$

A Figura 4.1 apresenta dois exemplos de modelos gerados por gramáticas criadas aleatoriamente, de acordo com os passos descritos nesta seção.

4.3 Crossover

O procedimento *Crossover* recebe duas gramáticas G_1 e G_2 e retorna duas novas gramáticas identificadas por $NewG_1$ e $NewG_2$. O algoritmo começa chamando a função *SELECIONA_REGRA*, escolhendo uma regra R de G_1 e uma regra S de G_2 que serão trocadas para a criação das novas gramáticas. Este procedimento deve garantir que as regras R e S levem à derivação de símbolos terminais iguais. Esta restrição é importante, pois impossibilita que sejam

Algorithm 4.2 População Inicial

```
1: function POPULAÇÃO_INICIAL(xmin, ymin, xmax, ymax)
2:     Grammar g
3:     h ← RANDOM()
4:     nfloors ← (ymax − ymin)/h
5:     REGRA_INICIAL(g, nfloors, "Facade")
6:     for i ← 1 até nfloors do
7:         REGRA_ANDAR(i)
       return g
```

Algorithm 4.3 Seleção das regras que serão trocadas

```
1: function SELECIONA_REGRA(g1, g2)
2:     r ← REGRA_ALEATORIA(g1)
3:     Conjunto cr ← Símbolos terminais alcançados por r
4:     Conjunto candidatas ← ∅
5:     for Regra i ∈ g1 do
6:         Conjunto ci ← Símbolos terminais alcançados por i
7:         if cr = ci then
8:             ADICIONA_REGRA(candidatas, i)
9:     s ← REGRA_ALEATORIA(candidatas)
10:    return r, s
```

criadas fachadas indesejáveis, como prédios com portas de entrada em um andar diferente do térreo.

Como definido no Capítulo 2, nas regras da forma $LHS \rightarrow RHS$, cada símbolo não terminal LHS está associado a um conjunto de símbolos terminais ou não-terminais RHS, chamados de sucessores. Esta relação entre os símbolos da gramática impõe implicitamente uma *hierarquia* entre as regras da seguinte forma: os símbolos terminais ou não que aparecem em RHS serão filhos do símbolo que aparece em LHS em alguma regra da gramática. Desse modo, é fácil construir uma árvore relacionando os símbolos da gramática hierarquicamente.

Para definir o conjunto de símbolos terminais que podem ser alcançados a partir de uma dada regra R, basta que se tome os símbolos das folhas da sub-árvore da gramática cujo nó raiz é R. O conjunto de símbolos terminais alcançados por R representa, efetivamente, as regiões retangulares que são adicionadas à fachada devido a aplicação da regra cujo LHS é R.

A função *SELECIONA_REGRA* começa escolhendo uma regra aleatória R de G_1 e constrói um conjunto de valores semânticos de R. Em seguida, para cada regra $I \in G_2$, é construído o seu conjunto de valores semânticos e, se este conjunto for equivalente ao conjunto de R, I é adicionada à lista de regras candidatas a serem trocadas por R. Por fim, a regra S será algum dos elementos da lista de regras candidatas, escolhido aleatoriamente. O Algoritmo 4.3 descreve este procedimento.

Após a chamada à função *SELECIONA_REGRA*, já é possível construir as novas gramáticas que serão retornadas pela operação *Crossover*. Uma vez que a gramática pode ser vista como uma árvore, a troca de regras entre duas gramáticas significa permutar todo o ramo

(a)

(b)

Figura 4.1: Exemplos de modelos gerados pelas gramáticas da população inicial. As divisões em cada andar podem ser claramente observadas.

cujo nó raiz é a regra que se pretende trocar. Seja V_1 o conjunto de regras de G_2 que podem ser alcançadas por S e V_2 o conjunto de regras de G_1 que podem ser alcançadas por R, as novas gramáticas serão $NewG_1 = G_1 - R \cup V_1$ e $NewG_2 = G_2 - S \cup V_2$. O Algoritmo 4.4 descreve a operação *Crossover*.

A Figura 4.2 mostra um exemplo de aplicação da função *Crossover*. As regras marcadas por círculos pontilhados da Figura 4.2a serão trocadas entre si, assumindo que é possível, resultando nas novas gramáticas da Figura 4.2b.

4.4 Mutação

A operação de mutação é trivial. Dada uma gramática G, é escolhida uma regra $r \in G$ aleatoriamente e o tamanho de alguma das divisões da operação *split* desta regra é modificado,

Algorithm 4.4 Crossover

1: **function** CROSSOVER($g1, g2$)
2: $r, s \leftarrow$ SELECIONA_REGRA($g1, g2$)
3: $v1 \leftarrow$ Regras de $g2$ que podem ser alcançadas por s
4: $v2 \leftarrow$ Regras de $g1$ que podem ser alcançadas por r
5: $newg1 \leftarrow g1 - r \cup v1$
6: $newg2 \leftarrow g2 - s \cup v2$
7: **return** $newg1, newg2$

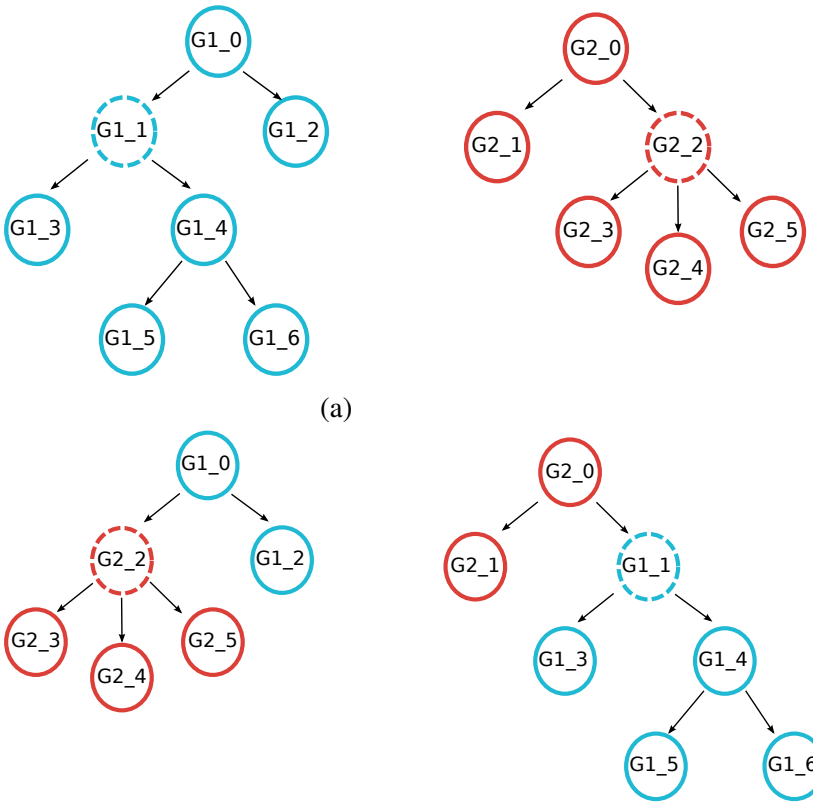


Figura 4.2: Crossover entre duas gramáticas G1 e G2. A regra G1_1 é trocada pela regra G2_2.

sendo incrementado ou decrementado por uma fração do valor anterior.

4.5 Fitness

A função de avaliação dos indivíduos da população é feita tomando-se amostras de modelos que podem ser gerados pela gramática e calculando diferentes medidas sobre estes modelos. Avaliar a lista de regras que compõe a gramática diretamente é uma tarefa inviável. Informações básicas sobre fachadas, como número de andares e janelas, não podem ser obtidas apenas olhando para a gramática sem que fortes limitações no formato das regras fossem impostas. Tais limitações ainda se refletiriam na forma como é feita a operação *Crossover*, pois ele teria que garantir a consistência das novas gramáticas criadas.

No problema de otimização de fachadas, a função de avaliação deve ser capaz de retirar informações importantes do modelo para avaliá-lo. Como já descrito anteriormente, o modelo é representado por uma árvore em que os nós folhas representam os blocos que compõe a fachada, como janelas, paredes, portas, varandas, etc. Dada a lista de nós folhas, o número de andares do modelo é obtido e, para cada andar, a lista de nós terminais que pertencem a ele. Com estas informações, uma fachada pode ser avaliada calculando-se medidas sobre as regiões geométricas que formam os andares das fachadas.

Para criar esta lista de andares, a ideia é encontrar retas que não cortem nenhum dos polígonos que formam a fachada assim, duas retas consecutivas delimitam um andar. Com a lista de retas que formam candidatos a andar, marcamos quais caracteres terminais pertencem a qual andar e, por fim, excluímos da lista os candidatos a andar que não possuem janelas ou portas, já que estes são bons marcadores para a classificação dos andares da fachada.

No Capítulo 5 serão apresentadas algumas aplicações do método proposto com a descrição das funções de avaliação usadas.

5 EXEMPLOS E RESULTADOS

Neste capítulo são apresentados exemplos de aplicações da técnica descrita no Capítulo 4, além dos resultados obtidos.

Conforme discutido na Seção 1.3, problemas que envolvem a exploração de *layouts* podem ser modelados como problemas de otimização ou de amostragem probabilística (BAO et al., 2013). O método apresentado neste trabalho pode ser empregado para explorar o espaço de fachadas procedurais modeladas por *Split Grammars*, tratando o problema como uma busca guiada por uma função que seleciona características desejáveis nos modelos. A técnica proposta foi aplicada ao problema de maximização de simetria e uma variação deste problema, chamada de simetria multimodal, mas poderia ter sido utilizada na busca por gramáticas que geram fachadas ótimas de acordo com outras funções, como o gasto com materiais para a construção de fachadas ou a maximização da iluminação natural do ambiente interno, entre outros. As gramáticas resultantes desta busca podem ser utilizadas por um usuário para a rápida prototipação de ambientes urbanos ou cenários virtuais, editando as regras geradas a fim de se obter o *layout* final desejado, sem ter o tedioso trabalho de escrever todas elas manualmente.

Na Seção 5.1 será apresentado como maximizar a simetria em fachadas. Na Seção 5.2 esta ideia será estendida para permitir a criação automática de gramáticas que modelem fachadas mais complexas.

5.1 Simetria

Existem bons motivos para aplicar o método proposto no problema de maximização de simetria em fachadas. Além das razões apresentadas por (ZHANG et al., 2013), já discutidas no Capítulo 3, existe uma grande diferença visual entre uma fachada simétrica e uma assimétrica, o que é bom para identificar que a aplicação do método proposto leva a bons resultados. A Figura 5.1 mostra um exemplo de uma construção real cuja fachada possui a característica de ser simétrica.

Partindo de uma população inicial de gramáticas que geram fachadas totalmente assimétricas, o que é obtido pela aplicação das funções descritas na Seção 4.2, é fácil notar, visualmente, quando uma gramática passa a gerar modelos simétricos, resultantes da combinação adequada de regras das gramáticas da população inicial.

Para medir a simetria, são consideradas apenas a forma e quantidade das janelas que formam a fachada. Utilizando o método descrito na Seção 4.5 para extrair informações sobre os andares, a fachada é analisada sob três aspectos e um total de 5 quesitos, como explicado a seguir:

- Geral: Conta-se quantos pares de janelas são iguais na fachada, dividindo-se este valor pelo número total de pares. Se todas as janelas da fachada possuem a mesma área, este quesito terá valor igual a 1, assim como se todas as janelas forem diferentes, o valor será 0.

Figura 5.1: Fachada frontal de Tianjin University of Technology, Tianjin, China (*http://wikipedia.org/*).

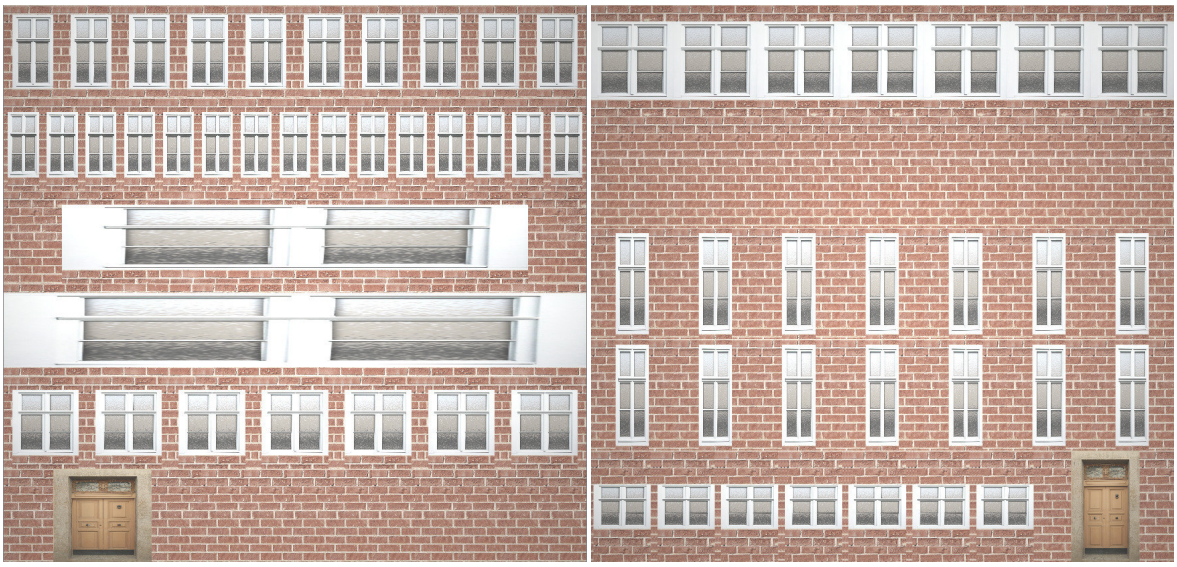
- Inter-andar: Comparam-se os andares, par a par, contando quantos possuem a mesma área ocupada por janelas e quantos possuem um número igual de janelas. A ideia é verificar a presença de simetria translacional. Assim como no item anterior, estes dois quesitos são transformados para que seus valores pertençam ao intervalo $[0, 1]$, e, se ambos forem iguais à 1, significa que tanto a soma das áreas das janelas quanto o número de janelas é igual em todos os andares.

- Intra-andar: Cada andar é analisado individualmente, dividindo-o ao meio em duas partes de mesmo tamanho. É verificado se a soma das áreas e o número de janelas de uma parte são iguais aos da outra. A ideia é constatar a presença de simetria reflexiva. Buscar simetria reflexiva em cada andar por vez, ao invés de analisar toda a estrutura de uma só vez captura melhor pequenas variações no *layout* da fachada, o que é bom para guiar o algoritmo genético durante a busca. Novamente, estes dois quesitos são normalizados para terem valores entre 0 e 1.

Por fim, é feita uma soma ponderada dos quesitos apresentados para a definição do valor final do *fitness* da gramática. O peso de cada membro que compõe a função objetivo tem um grande impacto no tipo de resultado que é obtido; assim, pesos maiores devem ser atribuídos a características mais desejáveis.

A Figura 5.2a mostra um exemplo do tipo de fachada obtida pelo algoritmo quando o peso associado às características da área das janelas é muitas vezes maior que o peso associado ao número de janelas, o que leva ao surgimento de andares em que a soma das áreas das janelas é igual, mas o número de janelas diferente. Já na Figura 5.2b, é mostrada a situação oposta: pesos para manter o mesmo número de janelas entre andares diferentes são maiores que os pesos associados a selecionar janelas de mesma área. É fácil ver que, neste caso, o método seleciona

andares com o mesmo número de janelas, mesmo que elas não tenham as mesma dimensões. Portanto, para que a busca por fachadas simétricas tenha sucesso, é importante uma escolha adequada dos pesos que modificam a contribuição dos quesitos apresentados anteriormente.



(a) Prioriza que os andares tenham a mesma área ocupada pro janelas.

(b) Prioriza que os andares possuam o mesmo número de janelas, dando pouca importância às areas das janelas.

Figura 5.2: Exemplo do uso de pesos com valores discrepantes. Em ambos os casos o algoritmo executou o mesmo número de iterações.

A Figura 5.3 mostra a evolução do *fitness* do algoritmo genético ao longo das iterações. As setas apontam para imagens que mostram as fachadas em algumas iterações específicas. Note que o *fitness* começa bastante baixo e sobe rapidamente, chegando a ficar muitas vezes maior que o valor inicial ($0,3$ até $1,8$). Note como as características da fachada vão ficando mais simétricas e uniformes, com os andares ficando cada vez mais semelhantes entre si.

A Figura 5.4 mostra a evolução de uma fachada ao longo das iterações. É fácil perceber como as trocas entre as regras da gramática levam à emergência da simetria, como desejado. É interessante notar, por exemplo, como na 5.4c um andar que tinha apenas uma janela passa a ficar com o mesmo número de janelas dos outros andares (Figura 5.4d), resultado de uma troca de regras que melhorou o valor do *fitness*.

Percebe-se que a aplicação do método leva a obtenção de gramáticas que derivam modelos realísticos, como mostra a Figura 5.5a. Esta fachada apresenta características próximas das observadas, por exemplo, em prédios de hotéis, devido ao número de janelas e à regularidade da disposição destas nos andares. Já a Figura 5.5b mostra uma fachada com uma única grande janela no térreo, característico de construções que possuem apartamentos residenciais nos andares superiores e algum tipo de loja ou comércio no térreo.

Devido à natureza estocástica do método, diferentes execuções do algoritmo levam à geração de gramáticas que derivam fachadas levemente distintas, como mostra a Figura 5.6. A exploração do espaço de modelos procedurais se torna possível devido a esta característica,

inerente ao método, que amostra o espaço de boas construções selecionando modelos que possuem características desejáveis.

Gramáticas que geram fachadas levemente diferentes podem ser empregadas para o projeto de ruas ou até mesmo de bairros inteiros. Estes tipos de ambientes urbanos são, geralmente, povoados por construções parecidas, cujo estilo é definido por aspectos sociais e econômicos. Para a modelagem de cenários virtuais, pode ser apresentado a um usuário diferentes modelos que possuam características desejáveis e, a partir deles, guiar o processo artístico de modelagem.

5.2 Simetria Multimodal

O modelo de simetria apresentado na seção anterior pode ser facilmente estendido para gerar fachadas mais complexas, formadas por partes assimétricas entre si, mas simétricas se analisadas individualmente. Assim como acontece com a simetria, vários prédios e fachadas projetados por arquitetos apresentam essa característica. Um bom exemplo de construção que possui esse tipo de característica é mostrado na Figura 5.7. A este tipo de característica foi dado o nome de simetria multimodal.

Para modelar uma função avaliação capaz de captruar a simetria multimodal, são aplicados os mesmos parâmetros da seção anterior, adicionando variáveis para definir o número de *clusters* que devem ser formados na fachada. Além disso, é póssivel definir quantos andares com o mesmo número de janelas são necessários para que se forme um *cluster* válido. Como mais variáveis são adicionadas ao problema de maximização de simetria, a busca por uma solução boa se torna mais lenta se comparada ao problema anterior, necessitando, assim, de um número maior de iterações em média.

A função objetivo para a simetria multimodal começa contando quantos blocos (*clusters*) de andares possuem o mesmo número de janelas, penalizando quando este valor é diferente do desejado. Depois, as mesmas medidas utilizadas para a simetria são aplicadas para o cálculo do *fitness*, porém serão considerados cada cluster individualmente, ao invés da fachada inteira. Por fim, o *fitness* de cada cluster é acumulado em uma variável, e dividido pelo número de blocos encontrados.

A Figura 5.8 mostra a busca por gramáticas capazes de gerar fachadas com 3 *clusters*. Após 150 gerações, chega-se a uma gramática que possui a característica desejada. O gráfico mostra a evolução do *fitness* ao longo das iterações. As setas apontam para imagens que mostram as fachadas em certas gerações específicas, e mostram a emergência da característica desejada.

Toda a discussão levantada na seção anterior também é válida para a busca por modelos que apresentem simetria multimodal. Novamente, é constatado que o método leva à geração de gramáticas que derivam modelos levemente diferentes e que lembram construções reais. Na Figura 5.9 são apresentados exemplos destes modelos. Nesse caso, a fim de ilustração, o método faz uma busca por fachadas que possuam dois *clusters*, cada um formado por pelo menos dois andares. A Figura 5.9a apresenta dois *clusters* válidos, o primeiro com cinco janelas em cada andar e o segundo, ocupando os últimos andares, com três elementos. A Figura 5.9b

também apresenta dois *clusters*: o primeiro, formado pelos dois primeiros andares e o segundo pelo restante. Pode-se notar que o primeiro cluster não obteve sucesso na maximização da simetria da área inter-andar, mas teve uma boa nota nos quesitos intra-andar e relacionados ao número de janelas.

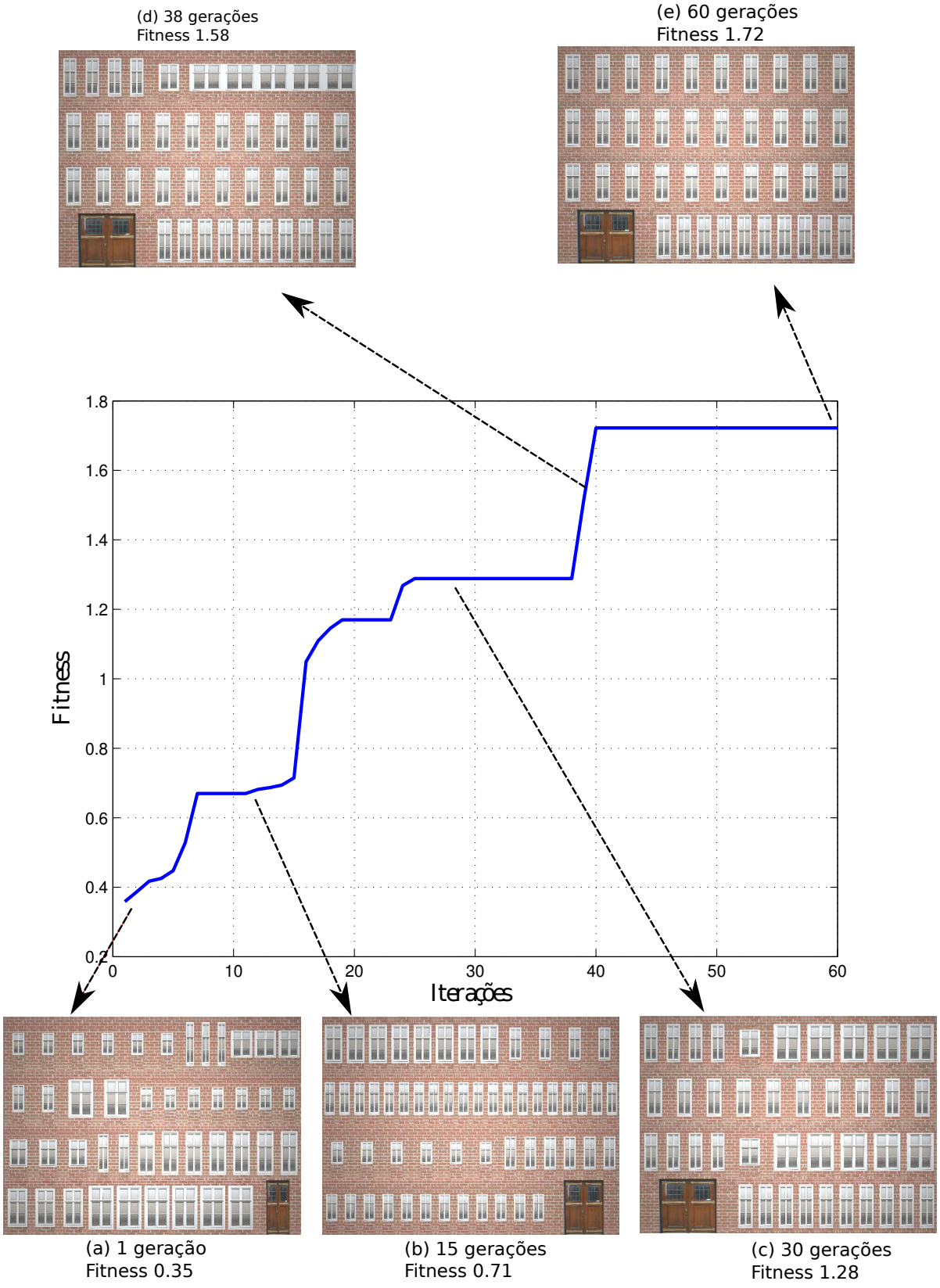


Figura 5.3: Evolução da mediana do *fitness* ao longo das iterações. Amostras da gramática foram tomadas e os modelos gerados por elas são indicados na imagem.

(a) (b)

(c) (d)

Figura 5.4: Evolução da gramática ao longo das gerações.

(a) Construção semelhante a um hotel.

(b) A imagem lembra construções comuns em centros comerciais, em que uma loja ocupa o piso do prédio e apartamentos residenciais ocupam os andares superiores.

Figura 5.5: Exemplos de construções obtidas pela maximização da simetria.

(a) (b) (c)

Figura 5.6: Variações de modelos obtidos pela aplicação do método. Esse tipo de característica é uma das vantagens obtidas pelo uso de algoritmo genético, o que permite amostrar modelos parecido com pouco trabalho.

Figura 5.7: Fachada do Palácio do Anhagabaú, sede da prefeitura de São Paulo (*http://wikipedia.org/*).

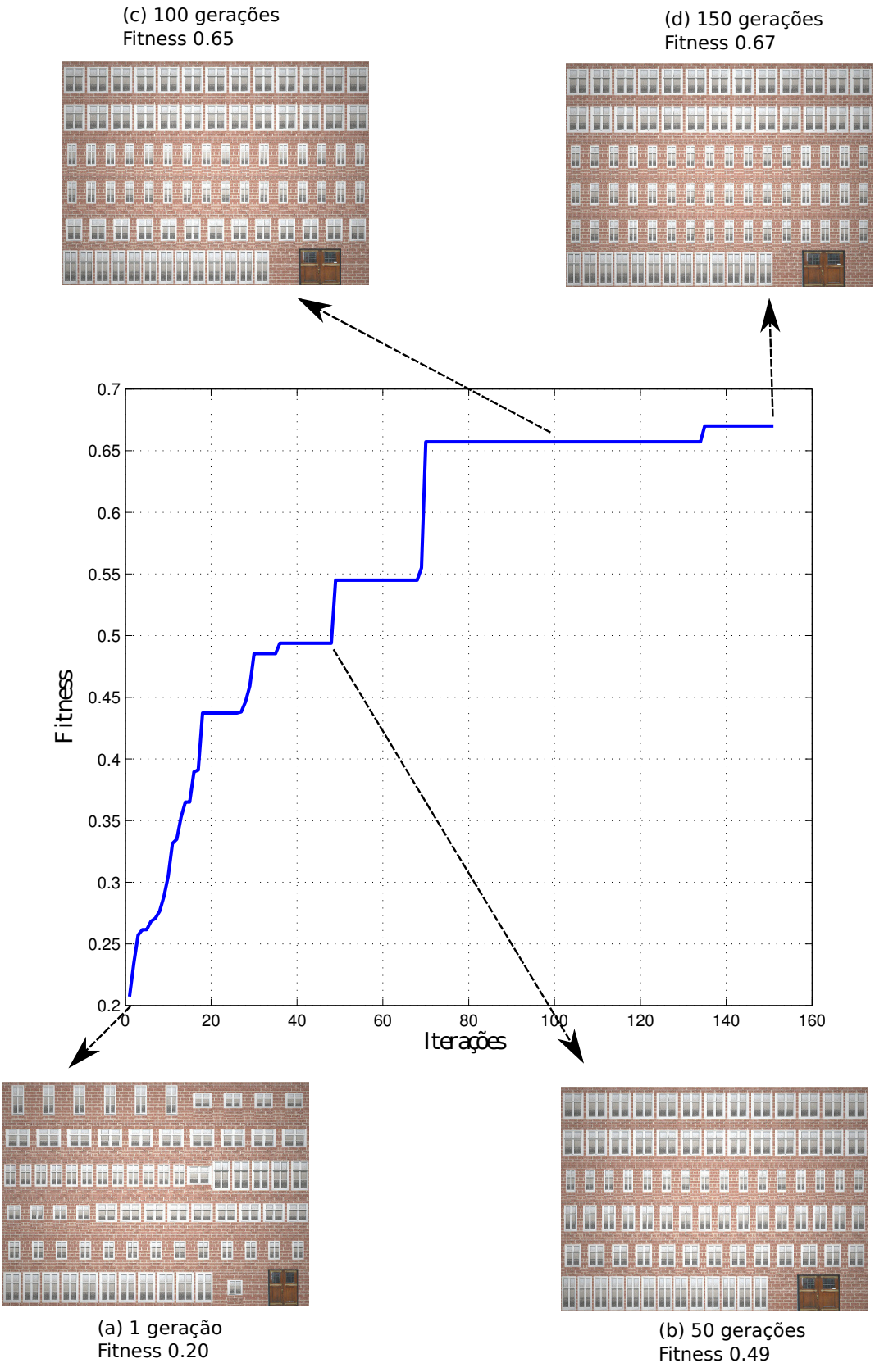


Figura 5.8: Evolução da mediana do *fitness* ao longo das iterações. Amostras da gramática foram tomadas e os modelos gerados por elas são indicados na imagem.

(a) (b)

Figura 5.9: Exemplos de modelos obtidos pela busca por simetria multimodal. As texturas das janelas são alternadas para representar mais realisticamente as construções.

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 Conclusões do Trabalho

Este trabalho desenvolveu uma técnica para evoluir *Split Grammars* utilizando Algoritmo Genético (AG), com o objetivo de resolver problemas que envolvem a exploração do espaço de modelos procedurais de fachadas de construções, modelando-os como problemas de otimização.

O método foi concebido evidenciando uma característica comum em uma grande quantidade de construções reais: a simetria. Dessa forma, o algoritmo evolutivo buscou otimizar dois aspectos na simetria:

- Simetria total: A fachada é considerada um bloco simétrico, de modo que os andares possuem características iguais entre si, como número, tamanho e posicionamento das janelas.
- Simetrial multimodal: Considera-se a existência de agrupamentos onde a simetria é maior intra do que inter agrupamentos. Ou seja, certos andares possuem padrões parecidos quando estão dentro de um mesmo grupo, mas diferentes ao serem comparados com elementos de outros grupos.

Inicialmente, no intuito de modelar o problema de otimização de fachadas, realizou-se uma revisão bibliográfica em modelagem procedural, principalmente nas técnicas empregadas na geração de construções, inferência de gramáticas, exploração de designs e algoritmos genéticos. Esse levantamento foi demonstrado nos Capítulos 2 e 3 e, com base nessas informações, a proposta de empregar o algoritmo evolutivo foi concebida, implementada e testada, conforme leitura dos Capítulos 4 e 5.

Levando em consideração a maximização da simetria nas fachadas, foi possível inferir gramáticas que geram modelos realisticamente plausíveis. O método proposto apresenta a vantagem de utilizar uma gramática de modelagem de fachadas (*Split Grammar*) já bem estabelecida no meio acadêmico, sem a necessidade de criar uma nova representação para construções procedurais, como fazem (BAO et al., 2013; ZHANG et al., 2013). Além disso, Algoritmo Genético já foi utilizado com sucesso para evoluir gramáticas utilizadas em processamento de linguagem natural, o que leva a crer que a utilização desta técnica com *Split Grammars* é promissora. Os experimentos realizados comprovaram a capacidade da técnica de gerar fachadas de vários estilos, com tamanhos e características diferentes, abrangendo uma ampla variedade de geometrias possíveis.

Um aspecto importante da técnica está na definição de uma boa modelagem matemática que caracterize as soluções candidatas, ou seja, a função *fitness* do AG. Como estas funções, geralmente, dependem de muitos parâmetros, a escolha apropriada de pesos tem também um grande impacto no resultado final da aplicação do método, bem como na sua eficiência. Pode-se citar, ainda, como outro aspecto importante da técnica proposta o fato de que, certas vezes,

não se chega perto do valor ótimo para o problema que se pretende resolver e, mesmo após várias iterações, uma gramática que não leva à construção de fachadas satisfatórias é retornada. Variações conhecidas de AG, como adaptatividade e paralelismo, poderiam ajudar a refinar a busca por boas soluções.

6.2 Trabalhos Futuros

Durante a implementação das técnicas apresentadas nesta dissertação, surgiu a necessidade de ampliação da proposta inicial, com a possibilidade de trabalhos futuros relacionados às seguintes ideias:

- A criação manual de gramáticas para modelagem procedural não é uma tarefa simples. Em geral, as gramáticas possuem muitas regras diferentes com parâmetros variados. Uma possibilidade de trabalho futuro é investigar como criar gramáticas tão boas quanto às escritas por humanos. Uma ferramenta capaz de concebê-las pode ser muito útil para arquitetos e urbanistas, além de outros profissionais que trabalham com *CAAD (Computer-Aided Architectural Design)*. Esse objetivo poderia ser alcançado empregando algum método de aprendizagem de máquina que encontrasse os padrões nas gramáticas escritas por humanos e depois tentasse classificar as gramáticas inferidas pelo AG.
- Utilizar outros eixos de simetria para buscar outros tipos de prédios e até modelos não simétricos. Apesar de uma grande quantidade de construções apresentarem os dois tipos de simetria aqui classificados, ainda existem outros (por exemplo, por coluna ao invés de andar) que não foram explorados.
- Otimização de construções envolvendo simulações físicas, como a técnica apresentada por (GAGNE; ANDERSEN, 2010), que busca encontrar o tamanho ótimo de janelas, de modo a maximizar a iluminação natural dos cômodos.
- Expandir a gramática adicionando novas regras para gerar, por exemplo, varandas, colunas, lojas e outras características, para contemplar uma maior capacidade de geração de modelos pela técnica proposta.

REFERÊNCIAS BIBLIOGRÁFICAS

BAO, F. et al. Generating and exploring good building layouts. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 32, n. 4, p. 122:1–122:10, jul. 2013. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/2461912.2461977>.

GAGNE, J.; ANDERSEN, M. Multi-objective façade optimization for daylighting design using a genetic algorithm. In: *Proceedings of SimBuild 2010-4th National Conference of IBPSA-USA*. [S.l.: s.n.], 2010.

KUNZE, A. et al. A conceptual participatory design framework for urban planning: The case study workshop 'world cup 2014 urban scenarios', porto alegre, brazil. p. pp. 895–903, 2011.

MANTYLA, M. *Introduction to Solid Modeling*. New York, NY, USA: W. H. Freeman & Co., 1988. ISBN 0-88175-108-1.

MARTINOVIć, A.; Van Gool, L. Bayesian grammar learning for inverse procedural modeling. In: *CVPR*. [S.l.: s.n.], 2013.

MITCHELL, M. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262631857.

MüLLER, P. et al. Procedural modeling of buildings. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 25, n. 3, p. 614–623, jul. 2006. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/1141911.1141931>.

MUSIALSKI, P. et al. A Survey of Urban Reconstruction. *Computer Graphics Forum*, Early View, maio 2013. ISSN 01677055. Disponível em: <http://doi.wiley.com/10.1111/cgf.12077>.

PARISH, Y. I. H.; MüLLER, P. Procedural modeling of cities. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 2001. (SIGGRAPH '01), p. 301–308. ISBN 1-58113-374-X. Disponível em: <http://doi.acm.org/10.1145/383259.383292>.

PROCEDURAL. *CityEngine*. 2008. Disponível em: <www.procedural.com>.

SIMON, L. et al. Random exploration of the procedural space for single-view 3d modeling of buildings. *Int. J. Comput. Vision*, Kluwer Academic Publishers, Hingham, MA, USA, v. 93, n. 2, p. 253–271, jun. 2011. ISSN 0920-5691. Disponível em: <http://dx.doi.org/10.1007/s11263-010-0370-6>.

TALTON, J. et al. Learning design patterns with bayesian grammar induction. In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2012. (UIST '12), p. 63–74. ISBN 978-1-4503-1580-7. Disponível em: <http://doi.acm.org/10.1145/2380116.2380127>.

VANEGAS, C. A. et al. Modelling the Appearance and Behaviour of Urban Spaces. *Computer Graphics Forum*, Blackwell Publishing Ltd, v. 29, n. 1, p. 25–42, 2010. ISSN 0167-7055. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2009.01535.x>.

WONKA, P. et al. Instant architecture. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 22, n. 3, p. 669–677, jul. 2003. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145-/882262.882324>.

WU, F. et al. Inverse procedural modeling of facade layouts. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 33, n. 4, p. 121:1–121:10, jul. 2014. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/2601097.2601162>.

ZHANG, H. et al. Layered analysis of irregular facades via symmetry maximization. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 32, n. 4, p. 121:1–121:13, jul. 2013. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/2461912.2461923>.