PEDRO RIBEIRO DE ANDRADE NETO

BUSCA DE PADRÕES EM SUBDIVISÕES PLANARES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: André Luiz Pires Guedes

CURITIBA

2004

Agradecimentos

A Deus, por mais esta etapa da minha vida.

Ao meu orientador, pelas conversas e conselhos que me guiaram no decorrer deste trabalho, e por entender os momentos em que eu podia, e os que eu não podia, me dedicar ao mestrado.

Ao professor Paulo Justiniano, pela oportunidade de trabalhar no projeto SAUDAVEL, que não tem relações diretas com este trabalho, mas que me proporcionou estabilidade e conforto para terminar o mestrado.

Ao Departamento de Informática da UFPR, pela oportunidade como professor substituto, e em especial ao professor Alexandre Direne, pelos valiosos conselhos e por sempre encontrar um tempinho para conversar.

Aos professores Hélio Pedrini, Cristina Duarte e Jair Donadelli, pelo aprendizado dentro e fora da sala de aula.

Aos meus pais e familiares, pelos ensinamentos, carinho, apoio incondicional, e por compreenderem a longa distância.

Aos amigos que aqui fiz: André, Bia, Cássio, David, Eduardo, Felipe, José Augusto, Juliano, Leslie, Marcos e Paulo, pela amizade e pelas boas horas de descontração.

A Conce, Edson, Milena, Jamile e Talita, pela recepção e hospitalidade no difícil começo.

A Viviane, que me deu força e confiança para terminar este trabalho.

Resumo

O sub-isomorfismo de grafos é uma abordagem muito utilizada para solucionar problemas de busca de padrões, mas este é um problema NP-completo. Desta forma, deve-se investir em pesquisa para encontrar soluções aproximadas, ou que funcionem em casos especiais do problema. Subdivisões planares podem ser consideradas um caso especial de grafos, pois, além dos vértices e arestas, existe uma topologia mais rígida quanto à ordem das arestas, surgindo o conceito de face. Este trabalho apresenta um algoritmo linear para busca de padrões em subdivisões planares. Os padrões a serem buscados também são considerados subdivisões e, portanto, este é um problema de sub-isomorfismo. O algoritmo apresentado baseia-se em uma representação híbrida entre o dual e o grafo de regiões adjacentes (RAG) para representar os padrões, de forma a não ter qualquer custo adicional de armazenamento. Então, os padrões são procurados na subdivisão de busca, utilizando um algoritmo de crescimento de regiões. Este trabalho também realiza um estudo comparativo das estruturas de dados mais utilizadas para armazenamento de subdivisões planares.

Abstract

Graph sub-isomorphism is a very used approach to solving pattern search problems, but this is a NP-complete problem. This way, it is necessary to invest in research of approximate solutions, or in special cases of the problem. Planar subdivisions can be considered as a special case of graphs, because, in addition to nodes and edges, there is a more rigid topology in relation to the order of the edges, arising to the concept of face. This work presents a linear algorithm for pattern search in planar subdivisions. The patterns to be searched are also considered subdivisions, and therefore it is a sub-isomorphism problem. The presented algorithm is based on a hybrid approach between the dual and the region adjacency graph (RAG) to represent the patterns, saving additional storage costs. Thus, the patterns are looked over the search subdivision, using an algorithm of region growing. This work also performs a comparative study of the data structures commonly used for storage of planar subdivisions.

Sumário

1 Introdução **1**

2 Subdivisões Planares e Estruturas de Dados **4**

2.1 Conceitos Básicos 5

2.2 Operações em Subdivisões Planares 5

2.2.1 Acesso aleatório 5

2.2.2 Acesso ao perímetro 6

2.2.3 Acesso à vizinhança 6

2.2.4 Subdivisão planar dual 7

2.2.5 Criação e remoção de elementos 8

2.3 Representações Geradas a Partir de Topologias 9

2.4 Construção de Subdivisões Planares 11

2.5 Complexidades de Tempo e Espaço em Subdivisões Planares 12

2.6 Estruturas de Dados para Subdivisões Planares 14

2.6.1 A estrutura *Winged Edge* 14

2.6.2 A estrutura DCEL 15

2.6.3 Outras estruturas 16

2.7 Conclusões 20

3 Sub-isomorfismo de Subdivisões Planares **21**

3.1 Isomorfismo e Sub-isomorfismo 21

3.2 Algoritmos Exatos 22

3.3 Algoritmos Aproximados 23

3.4 Algoritmos Algébricos 23

3.5 Busca Indexada 23

3.6 Crescimento de Regiões 24

4 Busca de Padrões em Subdivisões Planares **25**

4.1 RAG versus Dual 25

4.2 Crescimento de Regiões 27

4.3 Algoritmo para Sub-isomorfismo 31

4.4 A Estrutura de Dados DCEL 34
4.5 Complexidade do Algoritmo 34
4.6 Subdivisões Espelhadas 35
4.6.1 Busca em árvore binária 36
4.6.2 Pré-processamento do padrão 36
4.7 Implementação e Testes 36
4.7.1 Subdivisões de teste 37
4.7.2 Resultados das buscas 38
4.8 Considerações Finais 39

5 Conclusões **40**

Referências Bibliográficas **42**

Lista de Figuras

1.1 Exemplo de busca de padrões em desenhos de arquitetura [25] 2

2.1 Perímetros dos três elementos 6
2.2 Algoritmo para percorrer o perímetro de um elemento 7
2.3 Vizinhança dos três elementos 7
2.4 Algoritmo para percorrer a vizinhança de um elemento 8
2.5 Exemplo de uma subdivisão e sua representação dual 8
2.6 Restrições da definição de adjacência para regiões de uma imagem [33] 9
2.7 Exemplo de um RAG 10
2.8 Uma subdivisão planar e os seus múltiplos níveis de RAG 11
2.9 Os três elementos e suas relações 12
2.10 Relações indireta (a) e inversa (b) 13
2.11 Ponteiros da estrutura *Winged Edge* 15
2.12 Ponteiros da estrutura DCEL 16
2.13 Ponteiros das estruturas *Half-edge* (a) e *Vertex(Face)-edge* (b) 17
2.14 Ponteiros da Estrutura de Dados Simétrica 18
2.15 Ponteiros da estrutura Δ 18
2.16 Ponteiros da estrutura *Face-edge-vertex* 19

3.1 Grafos de subdivisões isomórficos não implicam nas subdivisões serem equivalentes 22
3.2 Crescimento de regiões [25] 24

4.1 Duas arestas não adjacentes podem ter o mesmo perímetro de faces 26
4.2 Algoritmo para crescimento de regiões 27
4.3 Variáveis do algoritmo de crescimento de regiões 28
4.4 Atualização das variáveis no algoritmo de crescimento de regiões 29
4.5 Exemplo de um crescimento de regiões 30
4.6 Algoritmo para sub-isomorfismo de subdivisões planares 31
4.7 Exemplo geração da lista L 32
4.8 Exemplo do algoritmo de isomorfismo 33
4.9 Necessidade da busca para as duas combinações de vértices 34
4.10 Duas subdivisões que podem ser isomórficas a partir de um espelhamento 36

4.11 Exemplo de subdivisão planar gerada de forma aleatória 37
4.12 Exemplos de padrões a serem buscados 38
4.13 Resultado das buscas 38
4.14 Tempos obtidos com a busca de padrões 39

Lista de Tabelas

2.1 Complexidade de armazenamento de cada uma das nove relações [47] 13
2.2 Resumo das características das B-reps 20

Capítulo 1

Introdução

Grafos com atributos associados aos seus vértices e arestas são amplamente utilizados para representar estruturas visuais complexas, em aplicações de visão computacional e reconhecimento de padrões. O uso de grafos nestas aplicações está relacionado com a existência de algoritmos eficientes para a solução de alguns problemas, podendo-se citar caminho mínimo, geração de árvore e buscas.

A comparação de grafos é uma estratégia muito utilizada para classificação de imagens, como por exemplo em aplicações CAD (*Computer Aided Design*) e robótica. Para isto, são utilizados algoritmos cujo objetivo é gerar um mapeamento que preserva as relações de adjacência entre os elementos de um grafo. Este mapeamento recebe o nome de *isomorfismo*.

Um dos fatores em destaque no uso de isomorfismo em visão computacional é a sua complexidade computacional. Esta é uma dificuldade inerente ao problema do isomorfismo. Algoritmos de força bruta requerem tempo fatorial $O(n!)$ para um grafo com n vértices, e o sub-isomorfismo de grafos é provado ser NP-completo [27]. Assim, são necessárias estratégias específicas para cada caso, com o objetivo de tornar o problema tratável. Muitas vezes são utilizados algoritmos aproximados, como redes neurais e algoritmos genéticos, mas eles podem encontrar um máximo local e não retornar a solução exata para o problema.

Uma das aplicações do isomorfismo é a busca de padrões em imagens segmentadas, como por exemplo em desenhos de arquitetura. Um exemplo pode ser visto na Figura 1.1. Nota-se que tanto os padrões a serem buscados quanto as imagens segmentadas possuem algumas características especiais, que os diferencia dos grafos padrões (vértices e arestas). São elas:

- Existe uma ordem na topologia associada. Assim, as arestas de um vértice estão ordenadas, seguindo uma certa direção;
- Havendo ordem na topologia, surge o conceito de face, que é um ciclo que não contém qualquer aresta no seu espaço interno;

- Quase a totalidade dos padrões a serem procurados contém, em cada vértice, pelo menos duas arestas. Por exemplo, em desenhos de arquitetura são procurados padrões como portas, mesas, etc., que são sempre estruturas neste formato. Desta forma, os vértices das imagens segmentadas que possuem apenas uma aresta podem ser descartados.

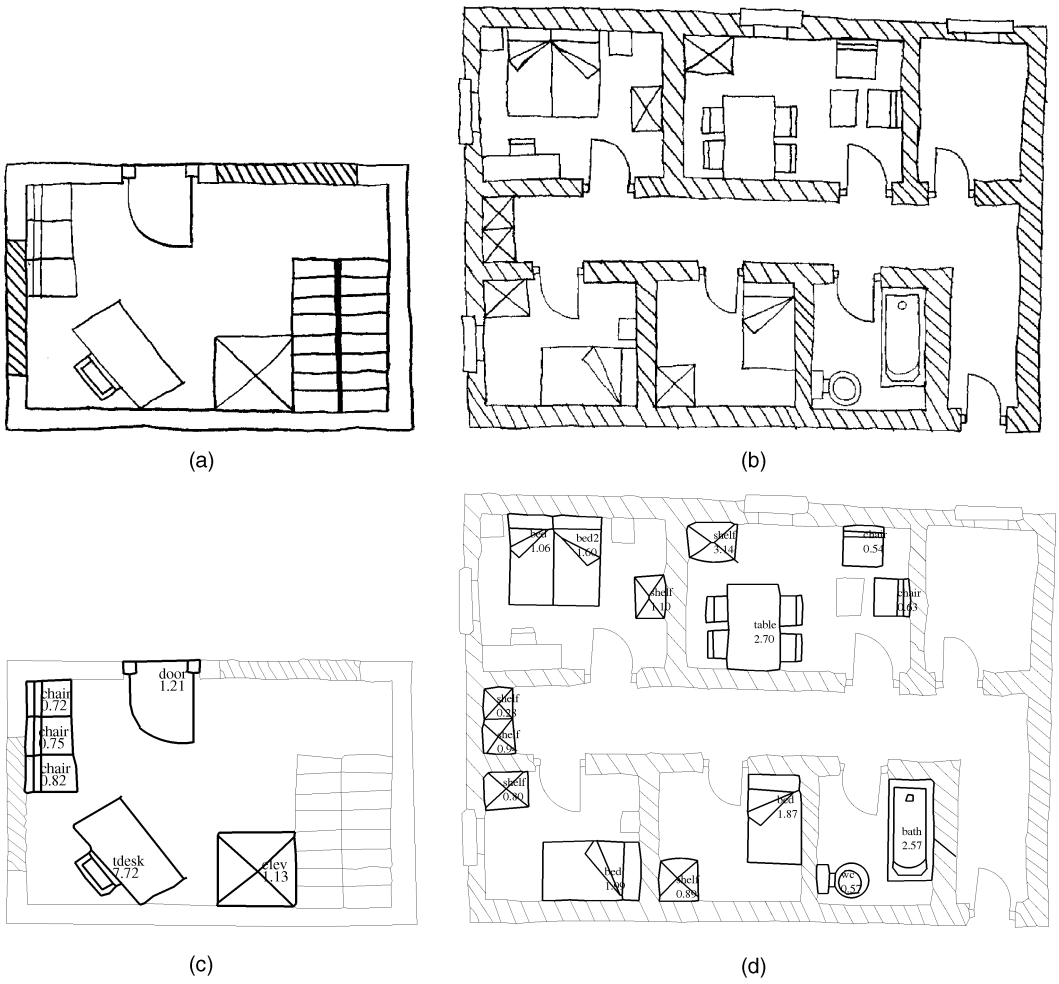


Figura 1.1: Exemplo de busca de padrões em desenhos de arquitetura [25]

Utilizando estas informações pode-se definir uma *subdivisão planar*, que é um particionamento de um plano em regiões denominadas *faces*. As faces são limitadas por segmentos de retas, as arestas, e cujos extremos são os vértices. Como os padrões a serem procurados também têm vértices, arestas e faces, eles podem ser considerados subdivisões planares, e então a sua busca é um problema de sub-isomorfismo.

Uma das representações muito utilizadas em processamento de imagens é o grafo de regiões adjacentes, ou RAG (*Region Adjacency Graph*). Essa representação consiste em agrupar *pixels* vizinhos e com características parecidas em uma mesma região, que será um vértice do grafo. As arestas conectando dois vértices indicam que as respectivas regiões possuem *pixels* vizinhos.

O objetivo deste trabalho é investigar o uso de RAGs no sub-isomorfismo de subdivisões planares. O uso de RAGs é comparado com o dual, e também com os vários níveis de RAGs, propostos neste

trabalho. Também é proposta uma representação híbrida entre o dual e o RAG para ser utilizada no sub-isomorfismo, aproveitando as vantagens de cada uma das representações.

O foco deste trabalho está na complexidade de tempo do problema. Desta forma, é apresentado um algoritmo linear para o sub-isomorfismo topológico de subdivisões planares. Também é objetivo deste trabalho realizar um estudo comparativo das estruturas de dados para representação de subdivisões planares, e o seu suporte às representações de RAG e dual.

Esta dissertação está dividida da seguinte forma. No Capítulo 2 são descritas características das subdivisões planares e são apresentadas e comparadas algumas estruturas de dados para o seu armazenamento. O Capítulo 3 descreve o problema do (sub-)isomorfismo, mostrando brevemente várias abordagens para a solução do problema. Um algoritmo linear para sub-isomorfismo de subdivisões planares é apresentado no Capítulo 4. Neste Capítulo também são descritas a implementação e os testes realizados, que comprovam a eficiência do algoritmo. Finalmente, o Capítulo 5 contém as conclusões e trabalhos futuros.

Capítulo 2

Subdivisões Planares e Estruturas de Dados

Em visão computacional, tipicamente as aplicações manipulam características perceptíveis, que são extraídas de imagens. Uma abordagem para obtenção de informações a partir de uma imagem é a extração de *vértices* e *arestas*, gerando uma representação com as seguintes características:

- Todo vértice está associado a um ponto único no espaço;
- Todo cruzamento de duas arestas torna-se um vértice;
- As arestas são linhas conectando os vértices, ou seja, esta representação tem um desenho bem definido no plano.

Uma *subdivisão planar* é um particionamento de um plano em regiões fechadas denominadas *faces*. As faces são limitadas por segmentos de retas, as arestas, cujos extremos são os vértices. Informações de adjacência entre os componentes de uma subdivisão planar compõem a *topologia* da subdivisão, e as descrições tais como a localização dos vértices e o formato das arestas compõem a *geometria*.

A representação de imagens utilizando subdivisões planares possui dois objetivos. Primeiro, dar suporte à execução de algoritmos de busca que facilitem a localização dos seus componentes, com a elaboração de estruturas de dados. Segundo, esta representação objetiva a simplificação da imagem, facilitando o seu desenho e representação.

Subdivisões planares são muito utilizadas em visão computacional para representação da realidade. Como aplicações para subdivisões planares pode-se citar CAD (*computer aided design*), modelagem e descrição de superfícies e objetos 3D.

Este Capítulo tem como objetivo descrever esta representação, mostrando suas características e operações. Também são apresentadas algumas representações que podem ser construídas a partir destas

topologias, que são os grafos de regiões adjacentes e seus vários níveis. Ao final do Capítulo, são descritas e comparadas as estruturas de dados mais utilizadas em memória primária para processamento destas informações.

2.1 Conceitos Básicos

Uma subdivisão planar é composta por um vetor de vértices V, um vetor de arestas A e um vetor de faces F. Para generalização, uma letra maiúscula representa um vetor, e uma letra minúscula representa um determinado elemento de um vetor, por exemplo $a \in A$. Dado um vetor X, o seu tamanho é representado por $|X|$. Um *elemento* de uma subdivisão planar pode ser um vértice, uma aresta ou uma face, e o *tipo* de um elemento é o vetor no qual ele está contido. Cada elemento pertencente ao conjunto X possui um identificador único, um valor inteiro entre 1 e $|X|$, inclusive.

Neste trabalho, são consideradas apenas partições finitas do plano. Também é assumido que nenhuma aresta passa por qualquer vértice além dos seus vértices limites, e cada vértice está no limiar de pelo menos duas arestas. Também é considerado que cada aresta separa duas, e somente duas, faces. Neste texto, freqüentemente será utilizada a palavra *subdivisão* ao invés de *subdivisão planar*, para simplificação.

2.2 Operações em Subdivisões Planares

Subdivisões são abstrações que manipulam essencialmente a topologia. Alguns operadores básicos são definidos para acesso e modificação de subdivisões, e eles podem ser divididos nas seguintes categorias:

- *acesso aleatório*,
- *acesso ao perímetro*,
- *acesso à vizinhança*,
- *subdivisão planar dual*, e
- *criação e remoção de elementos.*

O acesso ao perímetro e à vizinhança juntos constituem as *relações de adjacência* de uma subdivisão planar. Cada uma das cinco categorias define operadores para manipulação de subdivisões, que são descritos a seguir.

2.2.1 Acesso aleatório

O acesso aleatório se refere a como obter informações de um determinado elemento pertencente a um dado vetor, a partir do seu identificador único. Por exemplo, quando se deseja buscar características associadas a vértices, deve-se ter um acesso aleatório rápido a eles. Para o acesso aleatório é definido o operador de índice X_i, representando o elemento do vetor $X \in \{V, A, F\}$ cujo identificador único possui valor i.

2.2.2 Acesso ao perímetro

O perímetro define os limites geométricos e topológicos de um determinado elemento, sendo representado por uma lista circular ordenada, contendo elementos dos dois outros tipos, alternadamente. Estas listas podem ser percorridas nos dois sentidos, que correspondem aos sentidos horário e anti-horário na subdivisão. Para simplificação, o perímetro pode ser dividido nos dois sub-perímetros, que juntos constituem o perímetro original, percorrendo-se os elementos das duas listas alternadamente.

Os perímetros dos três elementos são mostrados na Figura 2.1. As arestas têm no seu perímetro apenas dois vértices e duas faces. Vértices e faces podem ser contornados por qualquer número de elementos dos seus dois tipos complementares.

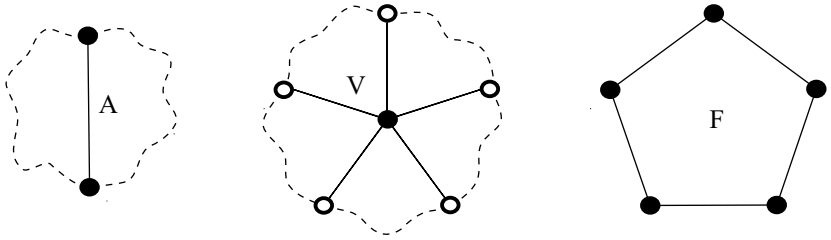


Figura 2.1: Perímetros dos três elementos

Para definição do perímetro, são necessários seis operadores básicos, chamados de $p(x, y)$, dois para cada par de vetores distintos. Dados x e y, pertencentes a vetores diferentes, com y pertencente ao perímetro de x, estes operadores retornam o próximo elemento do mesmo tipo que y, no sentido horário em torno do elemento x. O operador inverso, chamado de $p^{-1}(x, y)$, percorre o perímetro de x no sentido anti-horário.

A Figura 2.2 mostra um algoritmo que percorre todos os elementos de um tipo que formam o perímetro de um elemento x. Este algoritmo retorna uma lista contendo todos estes elementos.

2.2.3 Acesso à vizinhança

O acesso à vizinhança envolve elementos pertencentes a um mesmo vetor, e está baseado no compartilhamento de perímetros. Duas arestas são vizinhas se elas contêm um vértice e uma face em comum nos seus perímetros. Desta forma, cada aresta possui exatamente quatro vizinhos, como mostrado na Figura 2.3. Dois vértices são vizinhos se compartilham uma aresta e, conseqüentemente, duas faces. Duas faces são vizinhas se têm uma aresta em comum e, portanto, dois vértices. A vizinhança dos vértices e faces também são mostradas na Figura 2.3. Note que os vértices e as faces diferem das arestas, pois possuem número variado de vizinhos. O *grau* de um elemento é o número de vizinhos deste elemento.

Para a representação da vizinhança são necessários três novos operadores básicos, que são os $v(x, y)$. Esta primitiva retorna o próximo vizinho do elemento x no sentido horário a partir de um elemento y adjacente a x. A sua inversa $v^{-1}(x, y)$ percorre a vizinhança no sentido anti-horário de x.

Algoritmo para percorrer o perímetro

Entrada: $x \in X$, $y \in Y$, com $X, Y \in \{V, A, F\}$, $X \neq Y$ e y pertencendo ao perímetro de x

Saída: lista com todos os elementos do perímetro de x que sejam do mesmo tipo de y

1. $z \leftarrow y$, $Lista \leftarrow vazio$
2. repita
   (a) insira z no final de $Lista$
   (b) $z \leftarrow p(x, z)$

   enquanto $z \neq y$
3. retorne $Lista$

Figura 2.2: Algoritmo para percorrer o perímetro de um elemento

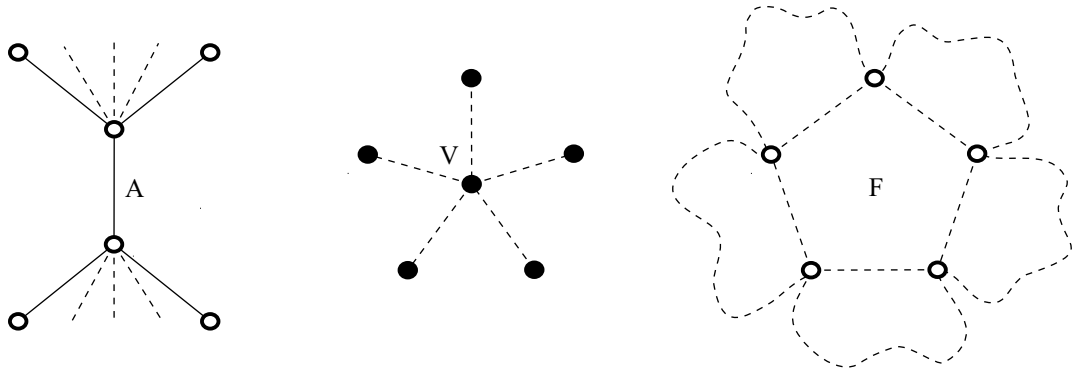


Figura 2.3: Vizinhança dos três elementos

A Figura 2.4 mostra um algoritmo para percorrer a vizinhança de um determinado elemento a partir de um vizinho. Este algoritmo retorna uma lista contendo estes elementos, da mesma forma que o algoritmo anterior.

2.2.4 Subdivisão planar dual

Dada uma subdivisão planar S, a sua representação dual pode ser definida. A subdivisão dual $D(S)$ contém vértices que correspondem às faces de S, e faces que correspondem aos seus vértices. Dois vértices são conectados por uma aresta em $D(S)$ se as correspondentes faces em S contêm uma aresta separando-as. Desta forma, para cada aresta unindo dois vértices em S, existe uma no dual separando as respectivas faces. Portanto, o dual contém o mesmo número de arestas que a subdivisão original, e o número de vértices e faces são trocados.

A Figura 2.5 mostra um exemplo de construção da subdivisão dual. Os círculos preenchidos repre-

Algoritmo para percorrer a vizinhança

Entrada: $x \in X$, $y \in Y$, com $X, Y \in \{V, A, F\}$ e y adjacente a x

Saída: lista com todos os elementos da vizinhança de x

1. $z \leftarrow y$, $Lista \leftarrow vazio$
2. repita
   (a) insira z no final de $Lista$
   (b) $z \leftarrow v(x, z)$

   enquanto $z \neq y$
3. retorne $Lista$

Figura 2.4: Algoritmo para percorrer a vizinhança de um elemento

sentam vértices na subdivisão original, e os não preenchidos são as faces. As arestas representadas por linhas retas pertencem à subdivisão original, e as tracejadas são as pertencentes ao dual. Nesta figura, cada aresta da subdivisão cruza com a sua correspondente no dual.

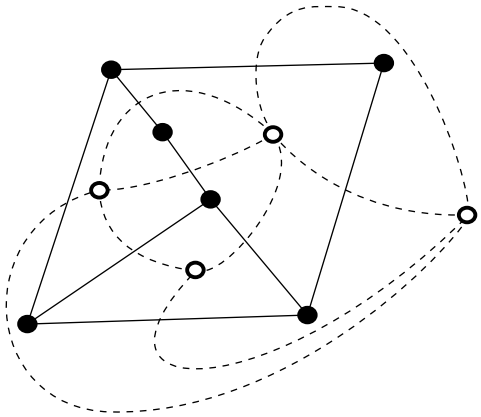


Figura 2.5: Exemplo de uma subdivisão e sua representação dual

2.2.5 Criação e remoção de elementos

A criação e remoção de elementos consiste na alteração da estrutura de dados, modificando a sua topologia, de forma a manter a subdivisão consistente. Estes operadores são destinados a estruturas de dados dinâmicas.

Operadores de criação e remoção de elementos não são utilizados neste trabalho, pois os algoritmos aqui descritos têm como parâmetros apenas estruturas de dados estáticas. Algumas estruturas de dados para representação de subdivisões, a serem apresentadas na Seção 2.6, suportam operadores para este

fim. Alguns deles são baseados na álgebra Euleriana, sendo chamados de operadores Eulerianos. Este trabalho não fornece informações sobre o seu funcionamento.

2.3 Representações Geradas a Partir de Topologias

Em uma imagem, uma região conexa de uma determinada cor pode ser descrita completamente pelo seu contorno, e geralmente esta representação consome bem menos espaço que sua descrição em termos de *pixels*. Portanto, operações sobre tais imagens podem ser facilitados se o *grafo de regiões adjacentes*, ou RAG (*Region Adjacency Graph*) estiver disponível [33]. A representação através do RAG consiste em extrair um grafo da imagem, de forma que cada vértice deste grafo corresponda a uma determinada região da imagem, e dois vértices estão conectados por uma aresta se as suas respectivas regiões forem adjacentes.

O conceito de adjacência deve ser definido cuidadosamente para imagens discretas, de forma a fazer com que o RAG seja sempre planar. Em particular, quando quatro regiões se encontram em um ponto, as duas conexões diagonais não são permitidas. Por exemplo, na Figura 2.6 apenas uma das duas arestas mais escuras que se cruzam pode existir no RAG.

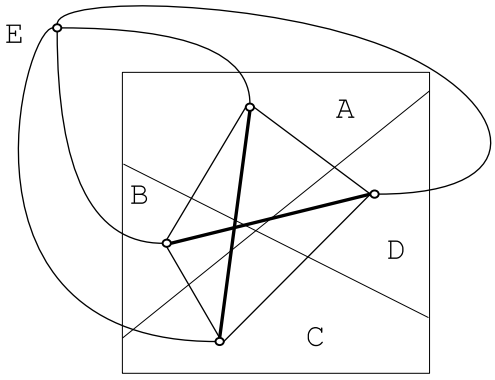


Figura 2.6: Restrições da definição de adjacência para regiões de uma imagem [33]

Segundo Lládos et al., o mesmo conceito de RAGs extraídos de imagens pode ser utilizado em grafos extraídos de imagens, de forma a reproduzir o seu desenho [25]. Assim, RAGs passam a ser uma representação de grafos em dois níveis, como mostrado na Figura 2.7. O primeiro nível contém o grafo representado na sua forma original, com vértices e arestas. O segundo nível é representado em termos das suas regiões adjacentes. Os vértices de um RAG representam as faces, ou seja, os minímos ciclos sem arestas no seu espaço interno. As faces, por sua vez, representam vértices do primeiro nível do grafo.

No RAG, um vértice de grau um corresponde a uma face completamente contornada por uma outra, ou que está contornada por uma outra e pela face externa (que não é representada no RAG). Vértices do RAG com grau elevado geralmente correspondem a faces grandes no grafo original, também chamadas de *globais*. Um *vértice de corte* de um RAG é um vértice c tal que existem dois vértices a e b no grafo com a propriedade que todos os caminhos de a para b passam por c. Faces de transição no grafo têm um

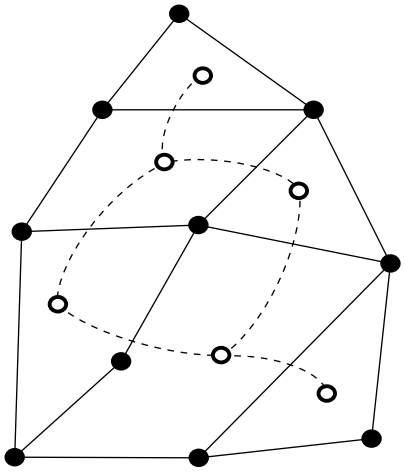


Figura 2.7: Exemplo de um RAG

grau baixo, e podem estar conectados a dois vértices de grau alto. Será utilizado $R(G)$ para representar o RAG de um dado grafo G.

Esta representação também pode ser utilizada em subdivisões planares. Note que existem semelhanças entre a geração de um RAG e um dual em uma subdivisão. Quando um RAG é gerado, algumas informações da subdivisão são perdidas, ao contrário do dual, que preserva toda a representação, pois o dual de um dual é a subdivisão original. As informações perdidas na criação do RAG são:

- a face externa da subdivisão é removida, portanto, todas as arestas da borda são perdidas, e
- no RAG, todas as arestas que compartilham o mesmo perímetro de faces são substituídas por apenas uma única aresta. Desta forma, todos os vértices de grau dois são removidos, e suas arestas são unidas.

A partir de um RAG $R(S)$, o seu RAG $R(R(S))$ pode ser gerado. O RAG de um RAG não é a subdivisão original, pois o número de vértices diminui com a sua criação, diferente do dual. Então, para cada subdivisão planar, podem ser gerados RAGs de RAGs até encontrar uma subdivisão que seja uma árvore, porque, com a ausência de ciclos, não é possível gerar um próximo nível de RAG, pois este não conterá vértices, gerando uma subdivisão vazia.

A Figura 2.8 contém um exemplo de uma subdivisão planar e os seus vários níveis de RAG. Os círculos preenchidos são os vértices da estrutura original, e os não preenchidos são as faces. Esta figura possui três subdivisões, que são a subdivisão original (acima), e o seu primeiro e o segundo nível de RAG (no meio e abaixo, respectivamente). Note que a aresta do segundo nível de RAG corresponde a uma aresta da subdivisão original, e os seus dois vértices também têm correspondentes. Portanto, os vários níveis podem ser armazenados em uma mesma estrutura de dados.

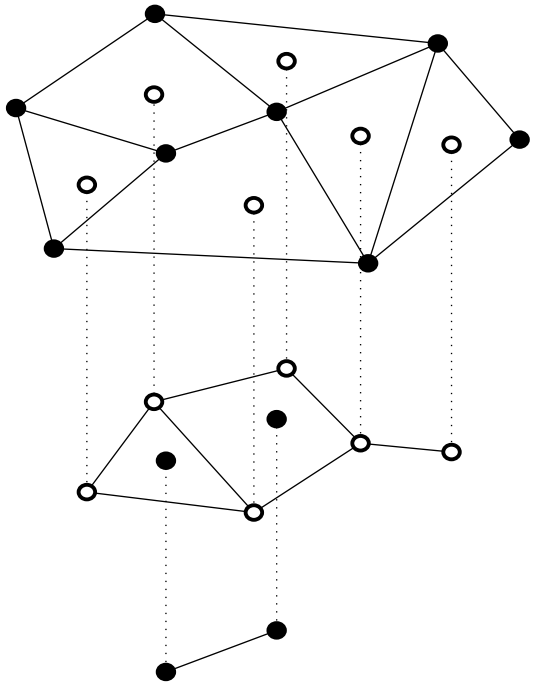


Figura 2.8: Uma subdivisão planar e os seus múltiplos níveis de RAG

2.4 Construção de Subdivisões Planares

Subdivisões planares podem ser originadas tanto de imagens quanto de modelos da computação gráfica. Para extrair subdivisões de uma imagem são necessários dois processos: *vetorização da imagem* e *estabelecimento de ordem na subdivisão planar*.

A vetorização da imagem consiste na extração de segmentos de reta, as arestas. O cruzamento de duas arestas define os seus limites, representados pelos vértices. Existem vários métodos para a extração dos segmentos, e cada um deles possui vantagens e desvantagens. Estes métodos têm como argumentos uma série de parâmetros e limiares, e um importante fator na robustez desses algoritmos é justamente a minimização do número de argumentos [40]. Os principais critérios para a escolha de um método de vetorização e os paradigmas mais utilizados podem ser encontrados em [41]. O resultado da vetorização são os vértices, com a sua localização espacial, e as arestas incidentes nestes vértices.

A subdivisão planar é então construída com o resultado da vetorização, gerando as faces e a topologia dos elementos. Um algoritmo intuitivo para resolver este problema consiste em ordenar as arestas incidentes em cada vértice através dos seus ângulos. Em casos como, por exemplo, a estrela com $n-1$ arestas ($K_{1,n-1}$), este algoritmo gasta tempo $\Theta(n \log n)$. Knuth mostrou que este algoritmo consome tempo $\Omega(n \log n)$ [22].

Kirkpatrick propõe um algoritmo para estabelecimento de ordem na subdivisão com complexidade $O(n + \log(\lambda(G)))$, onde $\lambda(G)$ é o número de atribuições topologicamente distintas do grafo G no plano [21]. Este algoritmo primeiramente define a ordem das arestas incidentes nos vértices cujos graus são limitados por uma constante. Então, aproveitando a estrutura combinatória do grafo, são deduzidas as ordens relativas aos vértices de maior grau, a partir das ordens já estabelecidas dos seus vizinhos de

menor grau.

2.5 Complexidades de Tempo e Espaço em Subdivisões Planares

Existem várias estruturas de dados para armazenamento de subdivisões. A comparação entre estas estruturas de dados é realizada baseando-se em dois fatores: o espaço necessário para armazenar a estrutura de dados, e o tempo para os acessos aleatórios, do perímetro e da vizinhança. O espaço utilizado é medido com relação ao número de arestas da subdivisão, e o tempo gasto é calculado através da soma dos tempos necessários para cada uma das operações básicas.

A Figura 2.9 mostra os três elementos e as suas nove relações. Uma seta indica que o elemento de origem armazena explicitamente elementos do destino, ou seja, ele tem *ponteiros* para o destino.

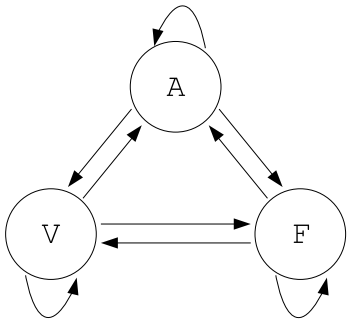


Figura 2.9: Os três elementos e suas relações

Cada elemento de um vetor tem acesso seqüencial armazenado implicitamente em uma estrutura de dados se existir uma seta com origem no vetor. O acesso à vizinhança é representado por uma seta na qual a origem e o destino são o mesmo elemento, enquanto que o acesso ao perímetro é representado por setas entre elementos de tipos diferentes.

O armazenamento implícito de uma relação de adjacência implica na realização da sua consulta através da *relação direta*, que é o acesso direto à estrutura de dados. Uma estrutura de dados perfeita para qualquer aplicação teria tempo de acesso $O(1)$ para cada uma das operações básicas. Mas existe um contrapeso, que é justamente o espaço consumido pela estrutura de dados, que está diretamente relacionado com o número de redundâncias armazenado.

Segundo Woo, o armazenamento implícito de cada uma das nove operações básicas de perímetro e vizinhança tem um custo fixo, mostrado na Tabela 2.1 [47]. Nesta Tabela, assim como no restante do texto, XY_i significa o número de elementos do vetor X adjacentes ao elemento Y_i. Woo mostrou que os limites de armazenamento para estas estruturas de dados é sempre linear, sendo $4|A|$ o limite inferior e $20|A|$ o limite superior. Ele também infere que, para o tempo, o limite inferior é $9k$, com k representando tempo de acesso constante, e o superior é $7|A| + 2k$, ou seja, duas operações com tempo constante e sete com tempo $O(|A|)$.

O acesso aleatório não costuma ser tratado nessas estruturas de dados, por dois motivos. O primeiro é que as estruturas de dados, como serão vistas na próxima Seção, costumam ter ponteiros com origem em

Tabela 2.1: Complexidade de armazenamento de cada uma das nove relações [47]

Relação	Soma	Armazenamento
$V \to V$	$\sum_i^{\|V\|} VV_i$	$2\|A\|$
$V \to A$	$\sum_i^{\|V\|} AV_i$	$2\|A\|$
$V \to F$	$\sum_i^{\|V\|} FV_i$	$2\|A\|$
$A \to V$	$\sum_i^{\|A\|} VA_i$	$2\|A\|$
$A \to A$	$\sum_i^{\|A\|} AA_i$	$4\|A\|$
$A \to F$	$\sum_i^{\|A\|} FA_i$	$2\|A\|$
$F \to V$	$\sum_i^{\|F\|} VF_i$	$2\|A\|$
$F \to A$	$\sum_i^{\|F\|} AF_i$	$2\|A\|$
$F \to F$	$\sum_i^{\|F\|} FF_i$	$2\|A\|$

todos os elementos, implementando o acesso aleatório implicitamente. O segundo motivo é que, caso a estrutura de dados não possua este tipo de acesso, pode ser adicionado um vetor com esta representação para o elemento, e isto adiciona espaço $O(|A|)$ à estrutura de dados.

Uma estrutura de dados que não for capaz de realizar todas as operações utilizando a relação direta necessita de alguma outra forma. As duas opções existentes são relação *indireta* e a relação *inversa*. A relação indireta consiste em utilizar duas (ou mais, contanto que este número seja constante) operações sucessivamente, para suprir uma operação inexistente. A relação inversa consiste em, dada uma operação desejada, realizar uma busca em uma relação cujo sentido é o inverso desta operação.

Por exemplo, considere a estrutura de dados mostrada na Figura 2.10. Uma face tem ponteiros para todas as suas arestas, e cada aresta tem ponteiros para os seus dois vértices. As linhas tracejadas representam duas consultas a serem realizadas na estrutura de dados. A consulta (a) corresponde à seguinte pergunta: "dada uma face, encontre todos os vértices do seu perímetro." Claramente esta consulta pode ser realizada *indiretamente* utilizando $F \to A$ e então $A \to V$. Já a consulta (b) corresponde a uma relação *inversa*. Se a relação $V \to F$ existisse, a relação $A \to F$ poderia ser computada indiretamente através de $A \to V$ e então $V \to F$. A impossibilidade de utilizar uma relação indireta leva à necessidade de se buscar a solução executando uma busca invertida na relação $F \to A$, procurando as faces cujo conjunto de arestas do seu perímetro contém a aresta escolhida. Esta consulta consome tempo $O(|A|)$.

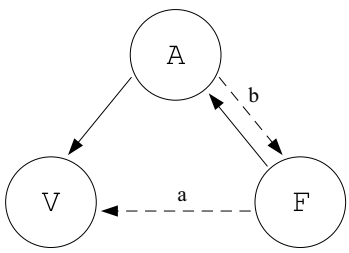


Figura 2.10: Relações indireta (a) e inversa (b)

Uma representação é capaz de armazenar o seu dual implicitamente se os vértices e as faces armazenam informações de forma simétrica. Mas note que, mesmo que a estrutura de dados não suporte o dual implicitamente, pode-se encapsular as consultas, bastando apenas trocar os vértices com faces para utilizar o dual [32].

2.6 Estruturas de Dados para Subdivisões Planares

As estruturas de dados utilizadas para representar subdivisões planares são chamadas de *estruturas para representar fronteiras* (*boundary data structures*), também chamadas de *B-rep*. Algumas destas estruturas de dados são capazes de representar informações mais complexas do que subdivisões planares, como por exemplo objetos 3-D. Para isto, elas necessitam de outras entidades em adição aos três vetores básicos, como por exemplo *corpos*, *cavidades* e *loops*. Neste trabalho, estas informações adicionais não são descritas.

Neste texto, uma B-rep representa apenas vértices, arestas e faces. As estruturas de dados aqui descritas têm como objetivo a representação de apenas informações topológicas das subdivisões planares, pois a representação de geometria a partir destas estruturas é trivial.

2.6.1 A estrutura *Winged Edge*

A primeira estrutura de dados para armazenamento de topologia em visão computacional baseando-se nas arestas foi proposta por Baumgart, e esta recebe o nome de *Winged Edge* [4, 5]. Arestas são mais facilmente detectadas em imagens que vértices e faces. Qualquer *pixel* pode ser um vértice, e faces são estruturas mais complexas. Uma estrutura com a topologia armazenada nas arestas possui as seguintes propriedades:

1. Outras representações são facilmente mostradas serem equivalentes em tempo linear, ou seja, transformações entre representações podem ser realizadas em tempo linear no tamanho da subdivisão planar [14, 30];

2. Esta representação se estende a problemas mais gerais como, por exemplo, a modelagem de superfícies poliédricas [5];

3. Algoritmos que têm como entrada um grafo planar desenhado em uma superfície podem ser descritos de maneira natural e eficiente em termos desta representação. Exemplos incluem algoritmos para triangulação, pré-processamento de subdivisões planares para localização rápida de pontos e regularização [14, 20, 24, 35].

Portanto, é viável o armazenamento de informações topológicas explicitamente nas arestas.

A *Winged Edge* aproveita esta característica, de forma que cada aresta desta estrutura armazena toda sua vizinhança e seu perímetro, e os vértices e as faces armazenam apenas um ponteiro para uma das suas arestas incidentes. Os seus ponteiros estão ilustrados na Figura 2.11. Os pesos contidos nas setas indicam a quantidade de ponteiros armazenados para esta relação. Por exemplo, $V \xrightarrow{1} A$ significa que cada vértice tem um ponteiro para apenas uma aresta.

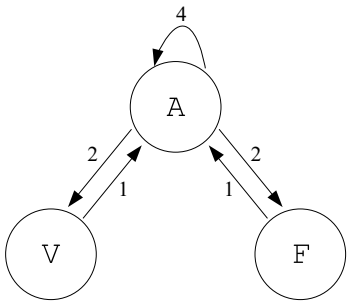


Figura 2.11: Ponteiros da estrutura *Winged Edge*

Para armazenar esta estrutura de dados é necessário espaço $8|A| + |V| + |F| = 8|A| + |A| = 9|A|$. Todas as operações básicas na *Winged Edge* têm tempo de acesso constante por transição, pois todas as operações indiretas realizadas utilizam ponteiros das arestas, que têm tamanho constante. As operações baseadas em vértices e faces consomem tempo igual ao tamanho do perímetro na procura do elemento de referência, antes de executar a transição.

2.6.2 A estrutura DCEL

Uma das mais simples estruturas de dados para representação de subdivisões planares é a DCEL (*Doubly-connected-edge-list*) [30, 35]. Seu nome é devido ao fato de que cada aresta está conectada a apenas outras duas na estrutura de dados, ao invés de quatro, como na *Winged Edge*. Outra diferença é que os ponteiros com origem nos vértices e faces não existem. Então, esta estrutura de dados é composta apenas por uma tabela contendo seis colunas, sendo quatro de informações ($V1$, $V2$, $F1$ e $F2$) e duas de ponteiros ($P1$ e $P2$). Cada linha da tabela representa uma aresta, com seus dois vértices, origem ($V1$) e destino ($V2$), e suas duas faces, esquerda ($F1$) e direita ($F2$). Como as arestas necessitam de uma orientação e uma direção, existem os índices $P1$ e $P2$. O índice $P1$ indica a primeira aresta circulando no sentido anti-horário em torno de $V1$, que é também a primeira aresta no sentido horário em torno de $F1$. Similarmente, $P2$ é utilizada como referência para $V2$ e $F2$. Um diagrama dos ponteiros da estrutura DCEL pode ser encontrado na Figura 2.12.

Esta estrutura quase não é citada na literatura pelas outras estruturas de dados propostas após ela. Mais ainda, ela contradiz a idéia de Woo, mostrada na Tabela 2.1, a qual são necessárias $4|A|$ para armazenar a vizinhança das arestas, pois na DCEL são utilizadas apenas $2|A|$.

As limitações da DCEL estão relacionadas justamente com a sua falta de redundância. Desta forma, as faces e os vértices não podem ser percorridos nos dois sentidos em tempo constante por transição. Apenas um sentido é permitido para os dois, e estes elementos são percorridos em sentidos opostos. Então, é necessário percorrer uma volta em torno da face para poder encontrar a próxima aresta no sentido anti-horário de uma face.

Esta estrutura de dados consome espaço $2|A|$ para cada um dos três pares de ponteiros armazenados nas arestas, totalizando $6|A|$. Os tempos de acesso da DCEL são os seguintes:

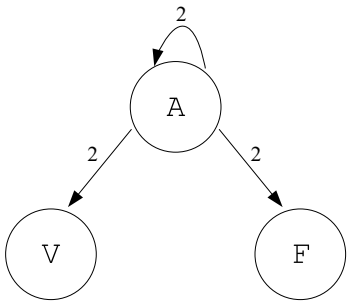


Figura 2.12: Ponteiros da estrutura DCEL

- $O(1)$ para todas as transições de perímetro e vizinhança das arestas, desde que sejam percorridas em apenas um sentido;
- $O(1)$ para acesso aleatório às arestas, pois cada uma é representada por uma linha da tabela;
- $O(|A|)$ para acessos com origem em vértices e faces. Como não é armazenada qualquer informação sobre estes elementos, uma busca invertida na tabela é necessária para encontrar uma aresta que tenha incidência em um destes elementos.

Para uma maior agilidade na recuperação das informações é necessária a criação de redundâncias, permitindo que todas as operações sejam realizadas de forma eficiente. Podem ser adicionados vetores de vértices e faces com ponteiros a uma das suas arestas, como na *Winged Edge*, para melhorar os acessos aleatórios.

2.6.3 Outras estruturas

Existem várias outras estruturas propostas, algumas são tentativas de melhorar a *Winged Edge*, outras apresentam novas abordagens, mas todas têm como objetivo central obter um tempo de acesso ótimo, minimizando o espaço gasto. Elas são citadas a seguir, e estão ordenadas pelas suas datas de publicação.

Split edges

Em estruturas baseadas nas arestas, toda aresta tem a função de representar tanto os limites das faces quanto a conexão entre duas faces. Na estrutura *Winged Edge*, uma aresta representa estas duas características em um único campo. Como cada aresta está na transversal de duas faces, é necessária a verificação da transversal de cada aresta (esquerda para direita ou o contrário). Para evitar esta verificação, surgiu o conceito de separação de arestas em metades (*split*), inicialmente proposto por Eastman [9]. Desta forma, cada metade está conectada a apenas uma face e um vértice.

Exitem duas variações da *split*, e os dois casos podem ser visualizados na Figura 2.13. A primeira conserva as arestas, e cada metade contém ponteiros para as suas respectivas arestas e para as metades anterior e posterior. Dois autores propuseram esta representação, Mantyla e Kalay, com as estruturas de dados *Half-edge* e a *Hybrid-edge*, respectivamente [19, 26]. A segunda variação exclui completamente o

conceito de aresta, com cada metade apontando para seus vizinhos e a sua respectiva metade irmã, nas estruturas de dados *Vertex-edge* e *Face-edge*, propostas por Weiler [44] e também por Wilson [45].

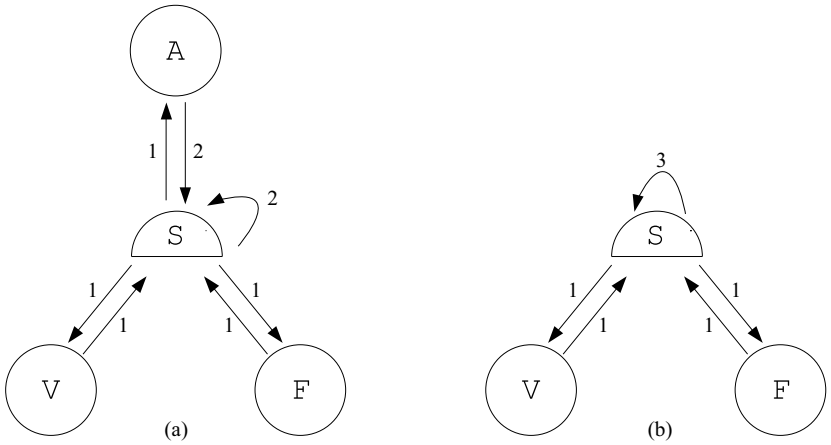


Figura 2.13: Ponteiros das estruturas *Half-edge* (a) e *Vertex(Face)-edge* (b)

Por estas estruturas de dados armazenarem uma entidade a mais, elas consomem muita memória. A *Half-edge* consome $5|A|$ para cada metade, mais $2|A|+|V|+|F|$, totalizando $13|A|$. A *Vertex-edge* consome o mesmo espaço para cada metade, mas elimina os dois ponteiros das arestas, consumindo espaço $11|A|$. Estas estruturas de dados armazenam o dual sem qualquer custo adicional, e tem os mesmos tempos de acesso que a *Winged Edge*, pois elas apenas evitam uma verificação que gasta tempo constante (a verificação da transversal).

Quad Edge

A estrutura de dados *quad-edge*, proposta por Guibas e Stolfi, particiona as arestas em grupos de oito [14]. Esta estrutura de dados é capaz de representar, simultaneamente, a subdivisão, o grafo dual e o grafo espelhado. Cada um dos oito grupos consiste em quatro arestas orientadas para o grafo e quatro para o seu dual. Mas, como o vizinho de uma aresta no sentido horário de um vértice é equivalente à próxima aresta no sentido anti-horário de uma das suas faces, então apenas quatro ponteiros são suficientes para representar a topologia das arestas. Portanto, os seus ponteiros são os mesmos da estrutura *Winged Edge* (Figura 2.11).

Estrutura de Dados Simétrica

A Estrutura de Dados Simétrica (*Symmetric Data Structure*, ou *SDS*) foi proposta por Woo e Wolter [48], e uma análise combinatorial da mesma foi estudada por Woo [47]. Uma versão chinesa desta estrutura foi proposta por Jiaguang et al. [17]. Os ponteiros desta estrutura podem ser visualizados na Figura 2.14.

Esta estrutura abandona a representação de toda a topologia nas arestas. É proposto o armazenamento dos perímetros de cada elemento, com as arestas armazenando apenas os seu perímetro, e vértices e faces armazenam todas as suas arestas.

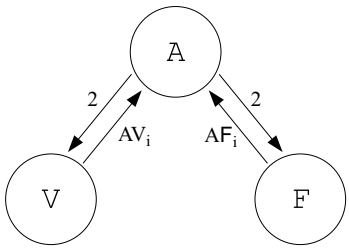


Figura 2.14: Ponteiros da Estrutura de Dados Simétrica

O espaço necessário para armazenar esta estrutura é $4|A| + AV_i + AF_i = 4|A| + 4|A| = 8|A|$, ou seja, $|A|$ a menos que a *Winged Edge*. O autor infere que seu tempo de acesso é mais rápido que o da estrutura *Winged Edge*, pelo fato que sua estrutura de dados armazena quatro das operações básicas implicitamente, enquanto que, segundo ele, a *Winged Edge* armazena apenas três, pois os ponteiros dos vértices e faces não armazenam toda a vizinhança. Mas, neste trabalho, esta idéia não é válida, pois os operadores definidos acessam apenas uma aresta por vez. Aqui é considerado que uma estrutura de dados é suficiente se ela possuir transições com gasto constante de tempo. Desta forma, mesmo com os armazenamentos $V \rightarrow A$ e $F \rightarrow A$ implícitos, é necessário percorrer todo o perímetro para encontrar a próxima aresta.

Estrutura de Dados Universal

A Estrutura de Dados Universal, também chamada de UDS (*Universal Data Structure*), foi proposta por Ala [1]. O objetivo da UDS é ser uma generalização completa para esquemas de representação de fronteiras. Qualquer estrutura de dados para a representação de fronteiras pode ser expressa como um caso especial desta estrutura de dados.

Para a representação de subdivisões planares, Ala apresenta a estrutura Δ, cujos ponteiros podem ser visualizados na Figura 2.15. Esta é uma estrutura de dados circular, na qual as operações básicas consomem entre uma e três transições de ponteiros, aproveitando o acesso indireto. Mas note que todas as operações de vizinhança e a operação $V \rightarrow A$ consomem tempo $O(VF_i^2)$, pois estas utilizam duas consultas que tem tamanho variado. Outra desvantagem da estrutura é a impossibilidade de se representar o dual, por ela ser circular. Esta estrutura tem armazenamento $6|A|$, igualando-se à estrutura DCEL.

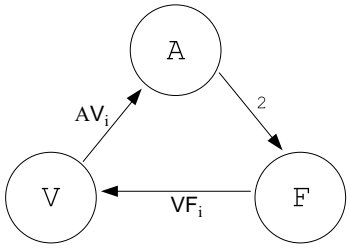


Figura 2.15: Ponteiros da estrutura Δ

Ala também apresenta uma estrutura chamada Δ-reversa, na qual todos os ponteiros da estrutura Δ são invertidos. Esta estrutura consome o mesmo espaço e tem as suas operações realizadas com o mesmo gasto de tempo que a estrutura Δ.

O autor estudou o comportamento de B-reps quando é necessário uso de memória virtual, encontrando anomalias no desempenho de estruturas de dados [2]. Ele mostrou que as estruturas de dados com maior armazenamento implícito não apresentam bom desempenho, por causa do alto uso de paginação apresentado por estas estruturas. As que têm menos ponteiros, como a estrutura Δ, possuem uma menor paginação e, conseqüentemente, um maior desempenho.

Face-edge-vertex

A estrutura de dados *Face-edge-vertex* foi proposta por Ni, e os seus ponteiros podem ser visualizados na Figura 2.16 [31, 32]. A estrutura *Face-edge-vertex* é uma simplificação da Estrutura de Dados Simétrica, eliminando os ponteiros dos vértices.

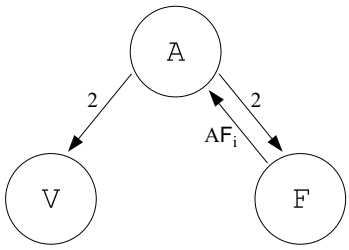


Figura 2.16: Ponteiros da estrutura *Face-edge-vertex*

Como esta estrutura de dados não armazena qualquer ponteiro com origem nos vértices, é necessário utilizar a relação inversa para computar as quatro operações que se originam de vértices. Mas o autor mostrou que existe uma classe de problemas na qual esta estrutura de dados é suficiente, pois quase não se utiliza estas operações. Ni também inferiu que não existe qualquer estrutura de dados ótima que armazene mais do que quatro das nove operações de adjacência.

Álgebra de Incidência

A álgebra de incidência consiste no armazenamento de três elementos adjacentes em uma tupla [13]. Desta forma, são necessárias apenas três operações básicas, fixando dois elementos para se encontrar o outro elemento adjacente aos dois. Assim, não é necessário percorrer uma lista de adjacentes para encontrar o próximo elemento. Estas três operações são necessárias e suficientes para descrever qualquer outra operação, inclusive as operações apresentadas neste Capítulo.

O seu armazenamento pode ser realizado utilizando uma tabela. Cada posição da tabela armazena o resultado das três operações para uma dada tupla, e um ponteiro para o seu respectivo vértice. Também é necessário um vetor para cada elemento, cujas posições apontam para uma tupla da tabela. Como cada aresta pode ter quatro combinações com os seus vértices e faces, são necessárias $4|A|$ colunas na tabela,

totalizando $4|A| \times 4 + 2|A| = 18|A|$ de armazenamento.

Esta estrutura de dados pode ser melhorada, de mesma forma que a DCEL que utiliza os ponteiros em apenas um sentido. Assim, o número de posições da tabela fica reduzido pela metade, e a estrutura de dados consome espaço $10|A|$.

2.7 Conclusões

Neste Capítulo foram descritas subdivisões planares, e foram apresentadas as estruturas de dados mais conhecidas na literatura para a sua representação. A idéia proposta por Woo na qual todas as B-reps que armazenam a vizinhança das arestas implicitamente necessitam de espaço $4|A|$ não é verdadeira em todos os casos, pois a estrutura DCEL utiliza espaço $2|A|$ e é capaz de realizar esta operação, embora em apenas um sentido.

A Tabela 2.2 contém um resumo das B-reps descritas neste Capítulo, indicando os seus gastos de tempo e espaço. A Tabela também indica se a estrutura é capaz de representar o seu dual sem qualquer custo adiconal. Os tempos gastos para acesso à adjacência são calculados com base no tempo necessário para realizar uma única transição. Desta forma, de nada adianta ter todos os elementos adjacentes armazenados implicitamente se, para fazer qualquer processamento com estes dados é necessário percorrer a estrutura. Portanto, as seis operações tendo como origem vértices e faces não consomem tempo constante, a menos que o número de elementos adjacentes esteja limitado por uma constante, ou se a estrutura de dados estiver armazenada em uma tabela *hash*, como na Álgebra de Incidência. Desta forma, se a estrutura de dados proporcionar uma transição em tempo constante então ela é suficiente.

Tabela 2.2: Resumo das características das B-reps

Estrutura de Dados	Ref.	Espaço	Tempo Aleatório	Tempo Perímetro[a]	Tempo Vizinhança[a]	Dual?
Winged Edge	[4, 5]	$9\|A\|$	$3k$	$4AV_i + 2k$	$2AV_i + k$	Sim
DCEL[b]	[30, 35]	$6\|A\|$	$2\|A\| + k$	$4\|A\| + 2k$	$2\|A\| + k$	Sim
Face(Vertex)-edge	[44, 45]	$13\|A\|$	$3k$	$4AV_i + 2k$	$2AV_i + k$	Sim
Half(Hybrid)-edge	[19, 26]	$11\|A\|$	$3k$	$4AV_i + 2k$	$2AV_i + k$	Sim
SDS	[47, 48]	$8\|A\|$	$3k$	$4AV_i + 2k$	$2AV_i + k$	Sim
UDS – Δ	[1, 2]	$6\|A\|$	$3k$	$AV_i^2 + 4AV_i + k$	$3AV_i^2$	Não
Face-edge-vertex	[31, 32]	$6\|A\|$	$\|A\| + 2k$	$2\|A\| + 2AV_i + 2k$	$\|A\| + 2AV_i$	Não
Álgebra de Incidência[c]	[13]	$18\|A\|$	$3k$	$6k$	$3k$	Sim

[a]Uma vez que AV_i, AF_i, FF_i, VV_i, FV_i e VF_i possuem a mesma complexidade, foi escolhido AV_i como unidade [48].
[b]A DCEL é capaz de acessar as adjacências de um elemento em apenas um sentido.
[c]Como estas estruturas de dados trabalham com tuplas, não é possível manipular apenas um elemento.

Neste Capítulo também foi mostrada a geração de RAGs a partir de subdivisões planares. Devido à remoção de informações na representação de um RAG, foi proposta a repetição deste processo até existir apenas uma árvore, chamado de vários níveis de RAG.

Capítulo 3

Sub-isomorfismo de Subdivisões Planares

Uma subdivisão planar pode ser considerada um grafo planar desenhado em uma superfície, sem cruzamento de arestas. Desta forma, algoritmos baseados em grafos podem ser utilizados para processamento de subdivisões, como por exemplo algoritmos de busca e de isomorfismo.

Isomorfismo é um mapeamento que preserva nos conjuntos as relações entre os seus elementos. Dois grafos que contêm o mesmo número de vértices conectados da mesma forma são ditos isomórficos. O problema do sub-isomorfismo consiste no mesmo mapeamento, mas de grafos com tamanhos não necessariamente iguais, procurando-se por conjuntos de vértices no grafo maior que possuam as mesmas relações do grafo menor.

Este Capítulo tem por objetivo mostrar algumas soluções adotadas ao problema do (sub-)isomorfismo em grafos. São descritos tanto algoritmos exatos e aproximados, quanto alguns casos especiais, onde o problema pode ser resolvido em tempo polinomial. Este Capítulo está baseado em [25].

3.1 Isomorfismo e Sub-isomorfismo

Formalmente, um grafo G é um par (V, A), com V sendo um conjunto de vértices e A um conjunto de arestas. Cada aresta é representada por um par $\{u, v\}$, com $u, v \in V$. Dois grafos G_α e G_β possuindo vértices $V(G_\alpha) = V(G_\beta) = V_n = \{1, 2, \ldots, n\}$ e arestas $A(G_\alpha)$ e $A(G_\beta)$ são ditos isomórficos se existe uma permutação p de V_n, tal que $\{u, v\} \in A(G_\alpha)$ se, e somente se, $\{p(u), p(v)\} \in A(G_\beta)$. Atualmente o problema do isomorfismo entre grafos não é provado ser P ou NP-completo.

Um outro problema baseado no mapeamento de grafos é o sub-isomorfismo. Dados os grafos G_α e G_β, deseja-se encontrar uma permutação dos vértices de G_α que seja isomórfica a um sub-conjunto $H \subseteq V(G_\beta)$. O isomorfismo pode então ser considerado um caso especial do sub-isomorfismo, onde o

número de vértices dos dois grafos é o mesmo. Por este motivo, o problema do sub-isomorfismo também é chamado de isomorfismo de sub-grafos. Este problema é provado ser NP-Completo [27].

Os grafos de duas subdivisões equivalentes S e S' são obviamentes isomórficos. O contrário não é necessariamente verdadeiro: se S e S' possuem grafos isomórficos, isto não implica nas subdivisões serem equivalentes. A Figura 3.1 mostra um exemplo. Os grafos gerados das subdivisões são isomórficos, mas as subdivisões não são equivalentes, pois as duas faces internas da subdivisão da esquerda são separadas por duas arestas, enquanto que as faces da subdivisão da direita são separadas por apenas uma.

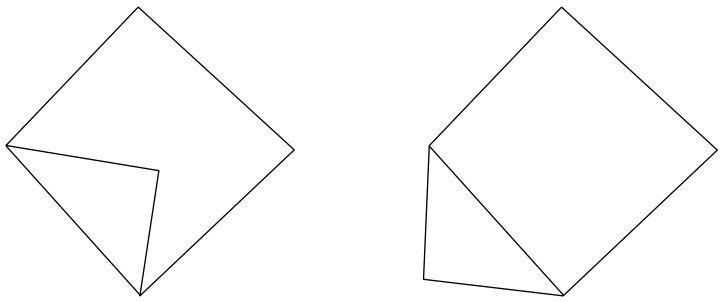


Figura 3.1: Grafos de subdivisões isomórficos não implicam nas subdivisões serem equivalentes

Este exemplo demonstra como os conjuntos de vértices e arestas não contêm informações suficientes para caracterizar a equivalência entre duas subdivisões. Portanto, alguns dos algoritmos descritos neste Capítulo devem passar por um processo de verificação após o isomorfismo.

3.2 Algoritmos Exatos

O isomorfismo para grafos não é provado ser P ou NP-completo, mas em alguns casos especiais ele é mostrado ser polinomial. Hopcroft mostrou um algoritmo linear para resolver o problema do isomorfismo em grafos planares [16]. Mitchell et al. propuseram algoritmos lineares para grafos em que todos os seus vértices pertencem à face externa (*outerplanar graphs*) [28].

Os algoritmos clássicos de sub-isomorfismo consistem em computar um mapeamento vértice a vértice incremental, executando uma busca em profundidade em uma árvore. A cada nível da árvore, um conjunto de mapeamento dos vértices é escolhido baseado em todas as instâncias anteriores.

Ullman aumentou a eficiência deste algoritmo, realizando um procedimento de *lookahead*, chamado de *forward checking*, de forma a rejeitar mapeamentos incompatíveis o mais cedo possível [42]. Métodos para relaxamento discreto são parecidos com a abordagem proposta por Ullman [15, 29]. Estes métodos reduzem o número de possíveis mapeamentos, verificando consistências em termos do número de arestas incidentes nos vértices, antes de avançar para o próximo nó da árvore de busca. Ambas abordagens reduzem o tempo de busca, mas nenhuma destas soluções consegue reduzir a complexidade de tempo do problema.

Uma outra abordagem para resolver o problema do sub-isomorfismo baseia-se no grafo de associação (*association graph*). O grafo de associação é construído baseando-se nos dois grafos de entrada para o

sub-isomorfismo, de tal forma que a solução do problema torna-se uma busca do maior clique no grafo de associação. A desvantagem desta solução está na dificuldade de se encontrar o clique, que também é um problema NP-completo. Entretando, em certas condições, este pode ser um método eficiente, como mostrado por Pelillo et al., que realiza sub-isomorfismo em grafos que podem ser organizados de forma hierárquica [34].

3.3 Algoritmos Aproximados

Algoritmos aproximados, também chamados de algoritmos de otimização contínua, têm como vantagem a obtenção da solução do sub-isomorfismo em tempo polinomial. Entretanto, a função de busca pode estabilizar em um máximo local, e não encontrar a solução ótima. Existem três técnicas aproximadas utilizadas em sub-isomorfismo. São elas:

Relaxamento probabilístico. Nesta técnica, o mapeamento entre os vértices não tem uma formulação binária, pois é definido em termos de uma função de probabilidade, que é iterativamente melhorada por um procedimento de relaxamento [7, 10, 12, 46].

Redes neurais. Os vértices de uma rede neural podem representar um mapeamento vértice-a-vértice, e os pesos das conexões da rede representam uma medida de compatibilidade entre os mapeamentos correspondentes [23, 38, 39]. A rede é programada de forma a minimizar uma função de energia (custo), que é definida em termos da compatibilidade entre os mapeamentos. O maior problema das redes neurais é que o procedimento de minimização é altamente dependente da inicialização da rede.

Algoritmos genéticos. Vetores de genes são definidos para representar o mapeamento entre o padrão e a entrada [8, 11, 18]. Estes vetores de solução são combinadas com operadores genéticos para encontrar a solução.

3.4 Algoritmos Algébricos

Grafos com peso são um tipo especial de grafos, que têm pesos associados às suas arestas. Um grafo com pesos G pode ser representado por uma matriz de adjacência $M_A(G)$, onde uma posição (i, j) contém o peso associado à aresta (v_i, v_j). Uma abordagem analítica pode então ser utilizada para resolver o problema de comparação dos grafos. Soluções interessantes baseadas em manipulação algébrica da matriz de adjacência foram propostas por Umeyama, que é chamada *eingendecomposition* [43], e Almohamad e Duffuaa, que utilizam programação linear [3]. Estes métodos funcionam apenas quando ambos o modelo e o grafo de entrada têm o mesmo número de vértices.

3.5 Busca Indexada

A comparação de grafos também é utilizada em recuperação de imagens baseando-se em conteúdo. Quando os conceitos de uma imagem podem ser representados por estruturas na forma de grafos, o

problema de procurar em uma base de dados por imagem que contém um determinado objeto pode ser resolvido através de uma busca indexada nesta base de dados. Sossa e Horaud propuseram o uso do polinômio do segundo imanantal da matriz Laplaciana de um grafo com o objetivo de criar um código *hash* para qualquer grafo [37]. Segundo os autores, o tamanho do grafo é inversamente proporcional ao erro da tabela.

Bunke e Messmer propuseram uma abordagem utilizando árvore de decisão [6]. Esta solução está organizada em termos das diferentes permutações das matrizes de adjacência dos grafos na base de dados. A busca indexada encontra a solução do problema em tempo polinomial, mas geralmente requer complexidade de tempo exponencial para a compilação da base de dados.

3.6 Crescimento de Regiões

Lládos et al. realizaram sub-isomorfismo em imagens segmentadas utilizando RAGs, com um algoritmo chamado de crescimento de regiões (*region growing*) [25]. Este algoritmo está baseado no sub-isomorfismo dos RAGs de grafos extraídos de imagens. Durante a iteração, o algoritmo realiza o crescimento de uma região nos dois grafos, de forma aproximada, percorrendo todo o perímetro da região. A Figura 3.2 mostra o funcionamento deste algoritmo. Foram definidos operadores para a comparação aproximada, e o seu algoritmo está baseado em *strings*.

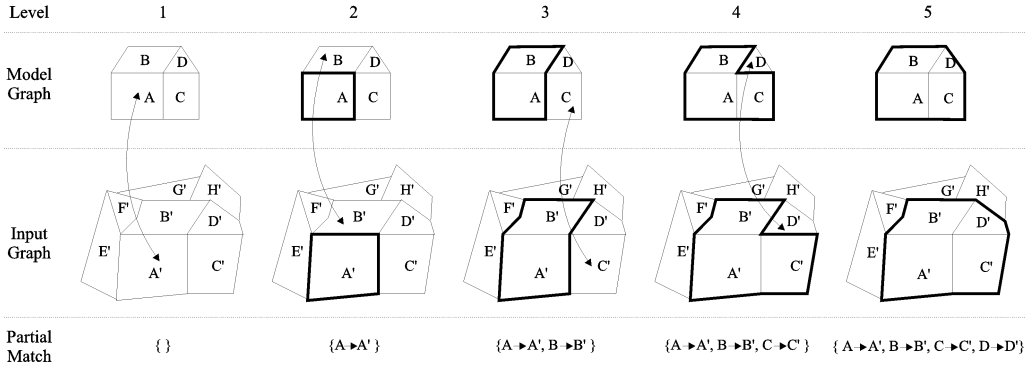


Figura 3.2: Crescimento de regiões [25]

Os autores concluiram que o algoritmo consome tempo exponencial, embora os resultados obtidos nos testes indicassem tempo pseudo-polinomial. O bom desempenho do algoritmo proposto foi justificado pela redução do número de vértices quando o RAG é gerado. Não foi proposta qualquer estrutura de dados para o armazenamento do grafo e do seu RAG.

Capítulo 4

Busca de Padrões em Subdivisões Planares

Subdivisões planares possuem características específicas se comparadas com grafos contendo apenas vértices e arestas. Este Capítulo tem como objetivo apresentar um algoritmo para sub-isomorfismo de subdivisões planares.

O algoritmo proposto é uma variação do algoritmo apresentado por Lládos et al., notando-se as diferenças entre um grafo com atributos, representando os segmentos de uma imagem, e uma subdivisão planar. Para facilitar a compreensão do algoritmo, este foi dividido em duas partes. A primeira parte consiste na definição de um procedimento para crescimento de uma região. A segunda etapa, que é o algoritmo propriamente dito, executa uma busca por sub-isomorfismos na subdivisão, utilizando o crescimento de regiões.

O algoritmo para sub-isomorfismo proposto neste Capítulo recebe como entrada subdivisões representadas utilizando a estrutura de dados DCEL. As justificativas da escolha da DCEL serão explicadas após a descrição do algoritmo. Antes de apresentar o algoritmo, é realizada uma discussão sobre o uso ou não do RAG como representação para resolver o problema.

4.1 RAG versus Dual

Lládos et al. propõem realizar o sub-isomorfismo utilizando RAGs. Seu algoritmo recebe como argumento apenas os segmentos da imagem, e então o RAG é construído, executando um algoritmo parecido com a geração de ordem em uma subdivisão. Estes dois algoritmos (construção do RAG e geração de ordem na subdivisão) têm mesma complexidade de tempo, uma vez que todas as faces devem ser percorridas.

O armazenamento de um RAG na mesma estrutura de dados da subdivisão original não é trivial. Quando duas ou mais arestas compartilham um mesmo perímetro de faces, elas têm que ser transformadas

em apenas uma no RAG, de forma que as relações de adjacência da subdivisão se conservem. Este problema seria simples se todas estas arestas sempre formassem um caminho, pois dado uma aresta que pertence ao caminho na subdivisão, esta seria representada pelas metades das duas arestas nos extremos do caminho no RAG. A não trivialidade deste problema se dá ao fato destas arestas não serem necessariamente adjacentes, como mostrado na Figura 4.1. Nesta figura, as duas arestas com maior espessura não são adjacentes, mas irão se transformar em apenas uma no RAG. Desta forma, o armazenamento do RAG na mesma estrutura de dados implica em perda de desempenho, pois tem-se que verificar, para cada aresta, se existe alguma outra com o mesmo perímetro de faces, e então tratar os seus casos. Assim, a perda de desempenho da estrutura de dados seria proporcional ao número de arestas removidas para gerar o RAG.

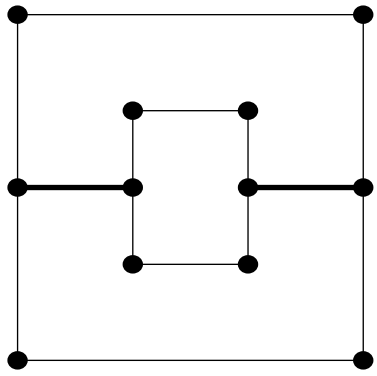


Figura 4.1: Duas arestas não adjacentes podem ter o mesmo perímetro de faces

Analisando a quantidade de arestas removidas com a geração do RAG, pode-se perceber que este valor nem sempre é significativo, pelo fato de este número ser muito dependente da subdivisão utilizada. Por exemplo, em uma malha triangular, este procedimento remove apenas as arestas cuja face externa está no seu perímetro e, portanto, não reduz consideravelmente o tempo do algoritmo subseqüente. Desta forma, a grande vantagem de se utilizar RAGs com o objetivo de redução da subdivisão é a remoção da face externa. Assim, a representação dos vários níveis de RAG também é descartada, pois a sua representação implica em mais problemas do que representar simplesmente o RAG.

No caso de subdivisões planares, pode ser considerado que o algoritmo de geração de ordem já foi executado anteriormente e, portanto, já existem faces. Mas estas faces não são necessariamente as mesmas do RAG, porque todas as arestas são conservadas.

Para evitar o armazenamento da subdivisão e do seu RAG em estruturas de dados diferentes, e por esta representação não ter vantagens significativas no quesito tempo, este trabalho utiliza uma estrutura híbrida entre o dual e o RAG para representação das regiões, aproveitando as vantagens das duas representações. Esta representação apresenta as mesmas características do dual, com duas pequenas diferenças. A primeira é que a face externa do dual deve ser marcada com um valor especial, aqui chamado de NULO. A segunda observação refere-se às arestas: uma aresta pertence ao dual se, e somente se, as duas faces do seu perímetro têm valor diferente de NULO. Desta forma, esta representação contém apenas as regiões

internas de uma subdivisão planar. Dada uma subdivisão S, esta representação tem o nome de $D_R(S)$.

4.2 Crescimento de Regiões

Uma vez definida a representação a ser utilizada pelo algoritmo, esta Seção descreve a primeira parte do algoritmo, que consiste no crescimento de uma região. Este algoritmo representa uma função, que será utilizada pelo algoritmo que efetua o sub-isomorfismo propriamente dito. Apesar do algoritmo aqui proposto realizar o crescimento de faces em uma subdivisão, o nome *crescimento de regiões* foi escolhido para este algoritmo porque este está baseado no algoritmo proposto por Lládos et al. A Figura 4.2 mostra os passos executados pelo algoritmo.

Algoritmo para Crescimento de Regiões

Entrada: $\mathrm{T} = \{\mathrm{T}_A, \mathrm{T}_V, \mathrm{T}_F\}$, a_p e v_p.

Saída: Se o crescimento da região foi realizado com sucesso (T é ou não válido).

1. $a_s \leftarrow \mathrm{T}_A[a_p]$, $v_s \leftarrow \mathrm{T}_V[v_p]$

   $f_p \leftarrow$ face do lado direito de v_p em a_p, $f_s \leftarrow$ face do lado direito de v_s em a_s

   $aloop_p \leftarrow a_p$, $aloop_s \leftarrow a_s$

   $of_p \leftarrow p(a_p, f_p)$, $of_s \leftarrow p(a_s, f_s)$

2. se $f_p = \mathrm{NULO}$ retorna verdadeiro, senão se $f_s = \mathrm{NULO}$ retorna falso

3. se $\mathrm{T}_F[f_p] \neq \mathrm{NULO}$ retorna $\mathrm{T}_F[f_p] = f_s$, senão $\mathrm{T}_F[f_p] \leftarrow f_s$

4. repita

   (a) se $\mathrm{T}_V[v_p] \notin \{\mathrm{NULO}, v_s\}$ retorna falso senão $\mathrm{T}_V[v_p] \leftarrow v_s$

   (b) se $\mathrm{T}_A[aloop_p] \notin \{\mathrm{NULO}, aloop_s\}$ retorna falso senão $\mathrm{T}_A[aloop_p] \leftarrow aloop_s$

   (c) se $of_p \neq \mathrm{NULO}$ e $\mathrm{T}_F[of_p] \notin \{\mathrm{NULO}, of_s\}$ retorna falso

   (d) $v_p \leftarrow p(aloop_p, v_p)$, $v_s \leftarrow p(aloop_s, v_s)$

   (e) $aloop_p \leftarrow v(aloop_p, v_p)$, $aloop_s \leftarrow v(aloop_s, v_s)$

   (f) $of_p \leftarrow p(aloop_p, f_p)$, $of_s \leftarrow p(aloop_s, f_s)$

   enquanto $aloop_p \neq a_p$ e $aloop_s \neq a_s$

5. retorna $aloop_p = a_p$ e $aloop_s = a_s$

Figura 4.2: Algoritmo para crescimento de regiões

O algoritmo de crescimento de regiões recebe três argumentos. O primeiro é T, uma variável composta por três tabelas (T_A, T_V e T_F), onde são armazenadas as referências de cada um dos elementos do padrão

a ser procurado na subdivisão de busca. Um elemento x que ainda não tem uma referência na subdivisão de busca tem o seu valor marcado como NULO em T_x (note que NULO é o mesmo valor utilizado para indicar uma face externa, mas não causa confusão, pois nos dois casos deseja-se representar a ausência de um valor). Neste algoritmo, T é um argumento recebido por referência, portanto, caso os seus valores sejam alterados, a variável passada como argumento para este algoritmo também será alterada.

Os outros dois argumentos para o crescimento de regiões são uma aresta a_p, indicando de onde será iniciado o crescimento da face, e um vértice v_p, estabelecendo o sentido da busca, que será de v_p para $p(a_p, v_p)$, percorrendo a face à direita de a_p neste sentido. O algoritmo considera que os elementos a_p e v_p já possuem referência em T antes do início da sua execução.

Dados os argumentos T, a_p, e v_p, este algoritmo possui dois objetivos. Primeiro, verificar se é possível realizar o isomorfismo para todos os elementos adjacentes à face a direita de a_p, baseado em um isomorfismo parcial contido em T. Segundo, atualizar T com as novas referências encontradas ao percorrer a face. Este algoritmo retorna um valor booleano, que indica se foi ou não possível realizar o crescimento da região, ou seja, se as modificações requeridas em T foram efetuadas com sucesso.

Primeiramente, o algoritmo inicializa algumas variáveis. Por convenção, f_p representa a face do lado direito em relação a a_p no sentido de v_p para $p(a_p, v_p)$. A variável of_p armazena a outra face de a_p em relação a f_p, ou seja, $p(a_p, f_p)$. $aloop_p$ representa uma aresta que inicialmente tem o valor de a_p, e será utilizada para percorrer a face. Para cada uma destas variáveis, e também para os argumentos a_p e v_p, existe uma variável que armazena a referência ao seu elemento na subdivisão de busca, que serão utilizadas para verificar e atualizar T. Estas variáveis possuem um índice s indicando que são da subdivisão de busca. As variáveis do algoritmo podem ser visualizadas na Figura 4.3, lembrando que para cada uma delas existe uma irmã com um índice s armazenando a sua referência na subdivisão de busca.

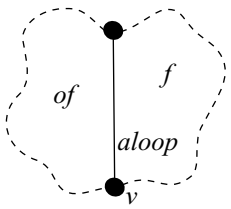


Figura 4.3: Variáveis do algoritmo de crescimento de regiões

Os passos 2 e 3 verificam algumas características das faces que se pretende percorrer. Se a face do padrão for a face externa (NULO), então o algoritmo termina retornando verdadeiro. A face externa não pode ser comparada com outras faces, pois, na maioria das vezes, esta corresponde a mais de uma face na subdivisão de busca. O mesmo acontece com a face externa da subdivisão de busca. Como esta não pode ser percorrida por não ter uma face no padrão com o seu tamanho, se o algoritmo encontrar esta situação, este é encerrado retornando falso. O único caso onde a comparação das faces externas é útil ocorre quando as duas subdivisões têm o mesmo tamanho, ou seja, no isomorfismo. Se a subdivisão de busca não possuir uma face externa, a condição que verifica o valor da sua face externa pode ser removida

do algoritmo.

O passo 3 verifica se T_F armazena algum valor para f_p, o que implica nesta face já ter sido percorrida anteriormente. Caso esta situação ocorra, o algoritmo retorna se a referência da face que se pretende percorrer é justamente o valor marcado em T_F. Se esta face não tiver sido percorrida anteriormente, o algoritmo atribui f_s a $\mathrm{T}_F[f_p]$, e então começa a percorrer f_p.

A repetição do algoritmo é representada pelo passo 4. Esta etapa está dividida em seis passos, sendo os três primeiros de verificação e atualização do isomorfismo (a,b,c), e os outros três de atualização de variáveis, de forma a percorrer a face f_p (d,e,f). Os três primeiros passos verificam se as entradas em T são válidas para poder atualizá-las, averiguando se o elemento já tem alguma referência, e se este valor é diferente do que foi encontrado. Uma vez que o algoritmo encontre um novo valor para um elemento já referenciado, isto implica na falha do isomorfismo. Caso contrário, o valor encontrado passa a ser a nova referência do elemento, e o campo do elemento em T é alterado com este valor. Note que T não é alterado para as faces no passo (c), porque o algoritmo supõe que uma face marcada implica nela já ter sido percorrida anteriormente. Portanto, as referências das faces só podem ser modificadas uma vez para cada execução deste algoritmo, e apenas para a face que se pretende percorrer, o que é representado pelo passo 3, fora da repetição.

Caso o algoritmo não encontre uma falha nos passos de verificação e atualização da tabela, as variáveis são atualizadas nos próximos três passos. Os valores de $aloop_p$, v_p e of_p (assim como o das variáveis da subdivisão de busca) são modificados com os próximos valores no sentido horário em torno de f_p, como mostrado na Figura 4.4. Os novos valores de v_p e of_p são calculados utilizando o perímetro da face, e o valor de $aloop_p$ é atualizado utilizando a vizinhança de v_p, pois percorrer v no sentido anti-horário é o mesmo que percorrer f_p no sentido horário. Desta forma, este algoritmo sempre percorre a face em apenas um sentido.

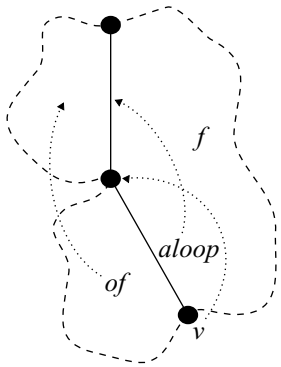


Figura 4.4: Atualização das variáveis no algoritmo de crescimento de regiões

A iteração percorre as duas faces, f_p e f_s, até que pelo menos uma das variáveis que represente suas arestas volte a ter o mesmo valor inicial. Se o algoritmo conseguir terminar a repetição, o isomorfismo é possível com os elementos das duas faces, contanto que elas tenham sido percorridas por completo, pois ambas devem ter a mesma quantidade de elementos adjacentes. Ao final do algoritmo, T contém as

referências dos elementos adjacentes à face percorrida atualizados.

Um exemplo de crescimento de uma região pode ser visualizado na Figura 4.5. O desenho do topo mostra as variáveis antes da iteração, e os dois outros mostram as atualizações das variáveis, nos seus respectivos passos (setas tracejadas). Continuando o algoritmo a partir do último desenho, ainda existirão mais duas iterações, até que $aloop_p$ volte a ser igual a a_p. Então o algoritmo retornará falso, pois as duas faces não possuem o mesmo tamanho ($aloop_s \neq a_s$). Note que o algoritmo também verifica as entradas de T, mas neste exemplo pode ser considerado que apenas as entradas de a_p e v_p possuem algum valor associado.

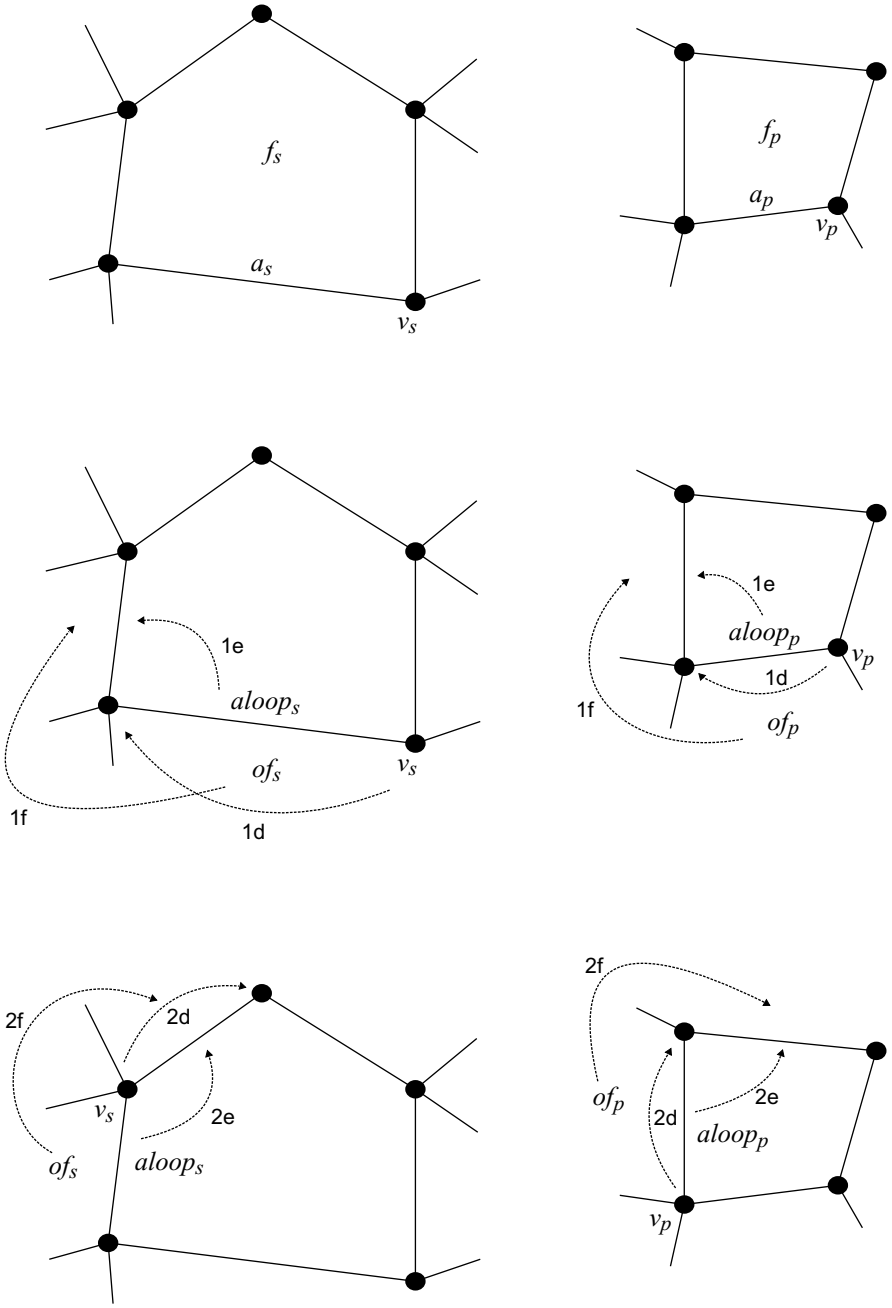


Figura 4.5: Exemplo de um crescimento de regiões

4.3 Algoritmo para Sub-isomorfismo

Uma vez que o algoritmo para crescimento de regiões está definido, agora será descrita a parte principal do algoritmo, que executa o sub-isomorfismo propriamente dito. Dadas duas subdivisões S e P, onde P é o padrão a ser procurado, e S representa a subdivisão onde será executada a busca, o algoritmo utiliza o crescimento de regiões para calcular o sub-isomorfismo. Os passos realizados pelo algoritmo são mostrados na Figura 4.6.

Algoritmo para sub-isomorfismo de subdivisões planares

Entrada: Um padrão P e uma subdivisão de busca S

Saída: Uma lista *Resp* com os sub-isomorfismos encontrados

1. gere $D_R(P)$, $Resp \leftarrow$ vazio
2. se $D_R(P)$ não possui arestas, inicie uma lista L com uma aresta $a \in P$, senão escolha uma aresta $a \in D_R(P)$ para iniciar a lista
3. se $D_R(P)$ tem arestas, execute um algoritmo de busca em largura, a partir de a em $D_R(P)$, inserindo todas as arestas encontradas no final da lista L
4. para cada aresta $a_s \in S$ e para $v \in \{V1, V2\}$ faça
   (a) inicie cada posição de T com NULO
   (b) isomórfico $\leftarrow$ verdadeiro
   (c) $a_p \leftarrow$ topo de L
   (d) T$[a_p] \leftarrow a_s$, T$[a_p[v]] \leftarrow a_s[V1]$, T$[p(a_p, v)] \leftarrow a_s[V2]$
   (e) enquanto não atingir o final de L e isomórfico repita
      i. se não cresce a região(T, $a_p, a_p[V1]$) isomórfico $\leftarrow$ falso
      ii. se não cresce a região(T, $a_p, a_p[V2]$) isomórfico $\leftarrow$ falso
      iii. $a_p \leftarrow$ próximo da lista L
   (f) se isomórfico insira T em *Resp*
5. retorne *Resp*

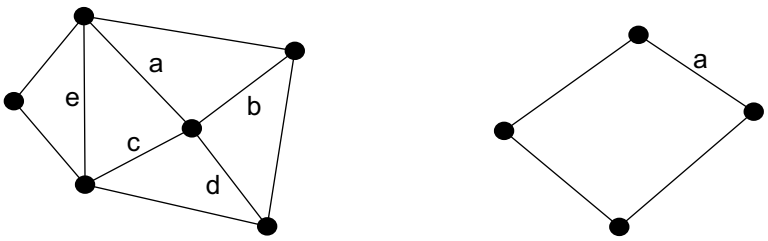
Figura 4.6: Algoritmo para sub-isomorfismo de subdivisões planares

No primeiro passo do algoritmo, $D_R(P)$ é gerado. Depois uma lista *Resp*, que armazenará todos os sub-isomorfismos encontrados durante a busca, é iniciada com vazio. Então existem duas possibilidades para o conjunto de arestas de $D_R(P)$, a serem verificadas e executadas pelo algoritmo nos passos 2 e 3, que são:

$|\mathbf{A_{D_R(P)}}| \neq \mathbf{0}$. O padrão contém pelo menos duas faces internas, pois os seus respectivos vértices serão unidos por uma aresta em $D_R(P)$. Então uma das arestas de $D_R(P)$ deve ser escolhida para iniciar a busca. Desta forma, o algoritmo executa uma busca em largura, procurando por arestas em $D_R(P)$, no passo 3. A ordem de arestas a terem suas regiões preenchidas deve ser a mesma ordem das arestas encontradas na busca, pois, uma vez que uma face de uma dada aresta tenha sido percorrida, os vizinhos da aresta já possuem referência em T, e podem ter as suas faces percorridas (note que, quando o crescimento de regiões for executado para um vizinho de uma aresta, uma das suas duas faces já terá sido percorrida). Desta forma, não existe diferença entre a busca em profundidade ou largura, pois as duas garantem que a sequência de crescimento de regiões é válida.

$|\mathbf{A_{D_R(P)}}| = \mathbf{0}$. O padrão P possui apenas duas faces, uma interna e outra externa. Desta forma, o crescimento de apenas uma região é suficiente para resolver o problema. Portanto, qualquer aresta do padrão pode ser escolhida. Assim, o passo 3 é abortado, porque o $D_R(P)$ não contém arestas a serem procuradas.

Por exemplo, na Figura 4.7 existem dois padrões, e são mostrados os resultados dos passos 2 e 3 utilizando estes padrões. Note que, nestes passos, a subdivisão de busca ainda não é utilizada.



(a) L armazena a sequência $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$

(b) L contém apenas o elemento a, ou qualquer um dos outros

Figura 4.7: Exemplo geração da lista L

Os três primeiros passos do algoritmo são de preparação de variáveis para serem utilizadas no passo 4, que realiza a busca dos padrões na subdivisão de busca. Cada aresta da subdivisão pode ser equivalente à primeira aresta da lista L, e estas arestas podem ser equivalentes apenas nas duas combinações possíveis de vértices. O passos (a) e (b) preparam as últimas variáveis antes do início do isomorfismo, que são (a) a inicialização de todas as posições de T com NULO e (b) a atribuição de verdadeiro à variável *isomórfico*, ou seja, existe um isomorfismo até que se prove o contrário. O passo (c) pega a primeira aresta de L, para em (d) atribuir os primeiros valores à T, que são a equivalência entre a_p e a_s, e uma combinação dos seus vértices. Daí a necessidade de o passo 4 ser repetido para cada uma das $2|A_S|$ possibilidades, que são os dois sentidos de cada uma das arestas de S.

No passo (e), a_p percorre toda a lista L, e o crescimento de regiões é chamado para os dois sentidos de a_p. Se qualquer um destes crescimentos de regiões falhar, então não é possível o sub-isomorfismo

utilizando a configuração inicial (a_s e uma combinação de vértices). Mas, se todas as regiões forem percorridas com sucesso, existe um sub-isomorfismo, e T é inserida na lista *Resp*. Finalmente, no passo 5, *Resp* é retornada com todos os sub-isomorfismos encontrados pelo algoritmo.

A Figura 4.8 mostra um exemplo de execução do segundo algoritmo. Note que, para este padrão, o algoritmo de crescimento de regiões é executado 10 vezes, mas em apenas 5 vezes são as regiões são realmente percorridas, representadas por linhas contínuas. Nas outras 5 vezes, as linhas tracejadas, o algoritmo de crescimento é abortado pelo fato da face já ter sido percorrida anteriormente. Assim, comprova-se a idéia de que, uma vez que a primeira aresta é encaixada na subdivisão de busca, não existe qualquer tempo combinatório, pois são necessárias apenas verificações, e estas dependem do tamanho do padrão, e não da subdivisão de busca.

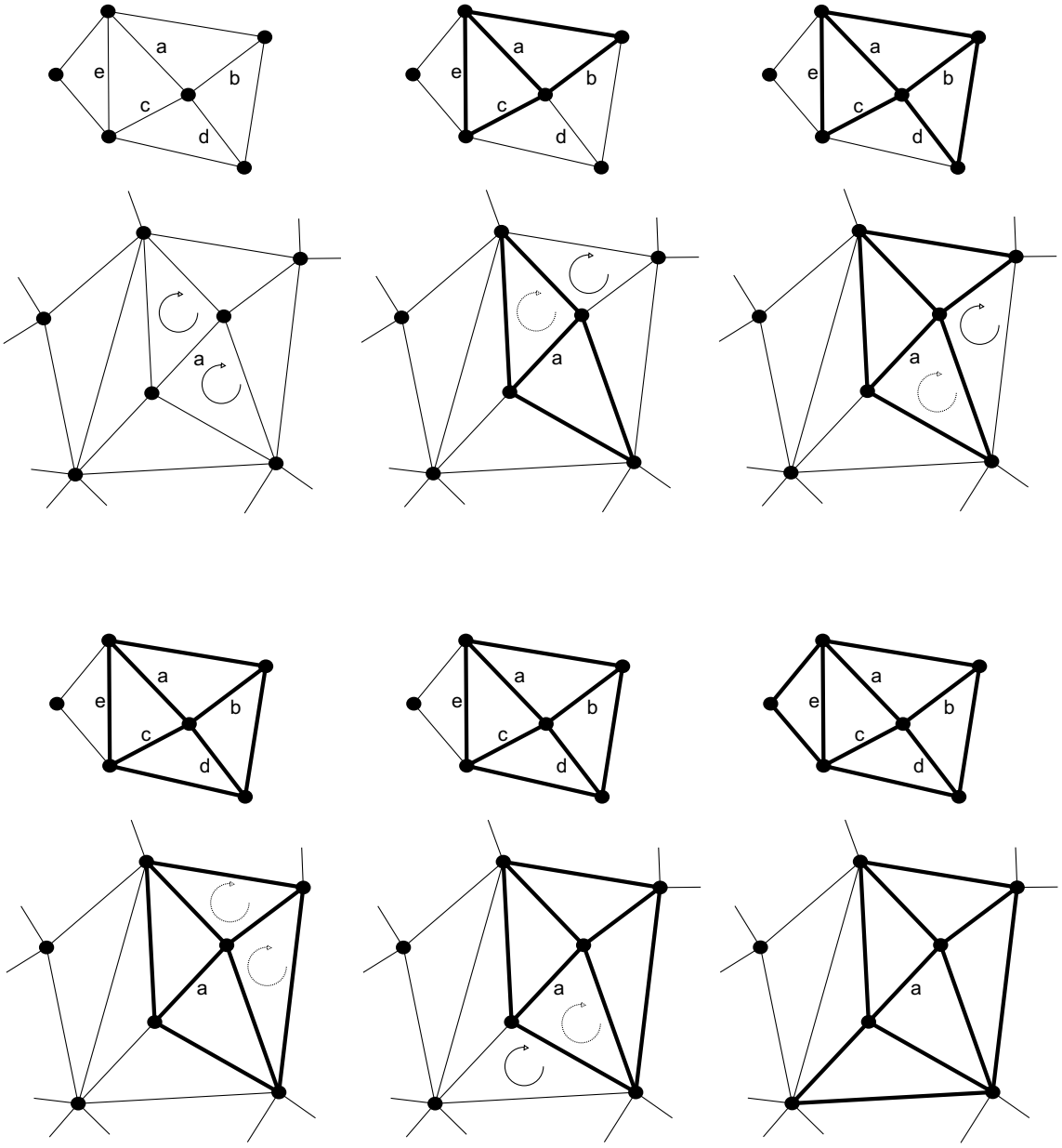


Figura 4.8: Exemplo do algoritmo de isomorfismo

Percebe-se também que, se os dois vértices de a fossem invertidos, o algoritmo também encontraria um isomorfismo, como mostrado na Figura 4.9.

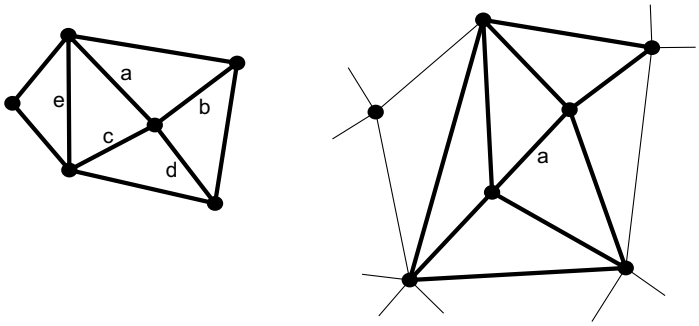


Figura 4.9: Necessidade da busca para as duas combinações de vértices

4.4 A Estrutura de Dados DCEL

A estrutura de dados DCEL foi escolhida como base para este algoritmo. Quatro motivos justificam a preferência pela DCEL. São eles:

1. A DCEL é uma estrutura de dados simples, com armazenamento de $6|A|$, e pode ser representada por uma tabela. Desta forma, o acesso ao dual é realizado em tempo constante, o que facilita a sua implementação;

2. O isomorfismo está baseado na comparação de arestas. Vértices e faces são utilizados apenas para verificação dos elementos do perímetro de uma aresta, portanto, não é necessário o acesso aleatório nem da adjacência a partir destes elementos;

3. O acesso ao perímetro é necessário somente em um sentido, pois o algoritmo percorre as faces apenas no sentido horário, utilizando a vizinhança das arestas baseando-se nos vértices.

4. Para gerar a tabela contendo as referências, é necessária a quantidade de arestas da subdivisão, e este valor é justamente o número de linhas da estrutura DCEL. O número de vértices e faces não pode ser obtido facilmente desta estrutura. Para resolver este problema, pode-se assumir que o algoritmo para gerar ordem na superfície retorna o número de faces encontradas, e então o número de vértices pode ser deduzido por Euler.

4.5 Complexidade do Algoritmo

Para análise da complexidade do algoritmo proposto, primeiro será analisada a segunda parte do algoritmo, e depois a primeira. Na segunda parte do algoritmo, o que consome mais tempo é a iteração que é realizada utilizando cada aresta da subdivisão S (passo 4), pois os outros passos não são repetidos, e D_R é gerado em tempo constante. Neste passo do algoritmo, existem dois trechos de código que requerem

mais processamento. O primeiro é o passo (a), que inicia todas as posições de T com NULO, e consome tempo $2|A_P|$, que é aproximadamente o tamanho de T ($|V| + |F| = |A| + 2$). O segundo trecho é o passo (e), que realiza o crescimento de regiões para cada uma das faces do perímetro das arestas contidas em L. Então a complexidade deste passo depende do crescimento de regiões.

A primeira parte do algoritmo percorre as arestas de duas faces, uma do padrão e uma da subdivisão, simultaneamente, até que uma das duas faces seja percorrida por completo. Como a estrutura DCEL consome tempo constante para todas as operações utilizadas no algoritmo, este procedimento consome tempo igual ao tamanho do menor perímetro entre as duas faces. Pode-se considerar, então, que este tempo é limitado pelo perímetro da face do padrão.

Nota-se que cada face do padrão é percorrida apenas uma vez, pois o algoritmo de crescimento verifica se a face já foi percorrida anteriormente, antes de iniciar a sua iteração. Como cada face é percorrida apenas uma vez, então o algoritmo passa por cada aresta do padrão no máximo duas vezes, uma vez para cada face do seu perímetro. Desta forma, o passo 4(e) do segundo algoritmo consome tempo limitado por $2|A_P|$. Portanto, para cada repetição do passo 4 do algoritmo de sub-isomorfismo é consumido tempo $4|A_P| = O(|A_P|)$.

Como o algoritmo de sub-isomorfismo realiza duas repetições para cada aresta da subdivisão de busca, o tempo total é $2|A_P| \times O(|A_S|) = O(|A_P||A_S|)$. O padrão a ser buscado pode ser considerado bem menor que a subdivisão de busca, e portanto pode-se afirmar que o tempo gasto pelo algoritmo é linear. No caso do isomorfismo, como os dois possuem o mesmo número de arestas ($|A_P| = |A_S|$), o algoritmo passa a ter complexidade de tempo $O(|A_S|^2)$.

À medida que é possibilitada a busca de padrões com características diferentes de subdivisões (duas ou mais arestas por vértice), se aproximando de padrões no formato de árvores, o problema vai se tornando cada vez mais combinatório. Isto acontece porque a ausência de faces faz com que a topologia seja mais dispersa, possibilitando combinações diferentes de um mesmo padrão.

4.6 Subdivisões Espelhadas

O algoritmo apresentado suporta sub-isomorfismo de subdivisões invariante quanto a rotações. Mas, em certas aplicações, pode-se desejar encontrar subdivisões como mostrado na Figura 4.10. Estas duas subdivisões são diferentes porque, se a face que possui quatro vizinhos distintos for percorrida no sentido anti-horário, o sub-isomorfismo se comportará de forma diferente se comparado com o sentido horário, pois os vizinhos não possuem as mesmas características. Mas estas duas subdivisões são isomórficas se for aplicada uma transformação de espelhamento em uma delas.

Para suportar o isomorfismo invariante a estas transformações por espelhamento, duas abordagens podem ser adotadas, que são a *busca em árvore binária* ou o *pré-processamento do padrão*. Estas abordagens são comentadas a seguir.

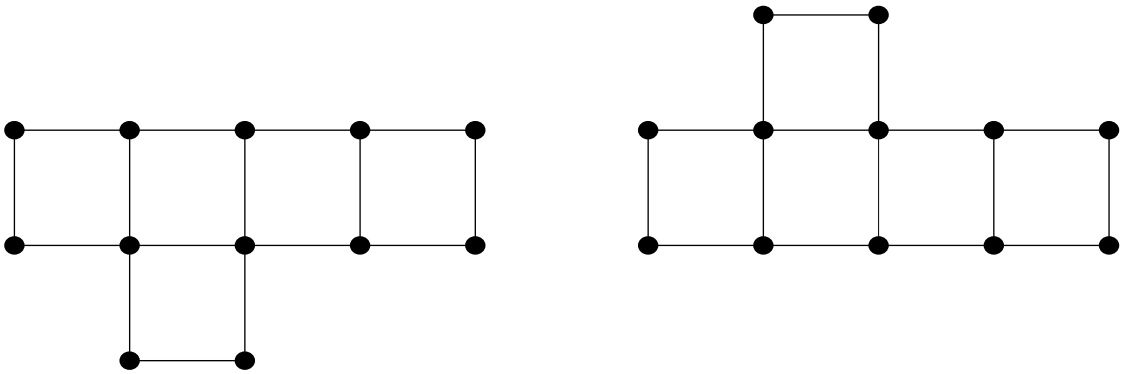


Figura 4.10: Duas subdivisões que podem ser isomórficas a partir de um espelhamento

4.6.1 Busca em árvore binária

A solução de busca em árvore binária funciona da seguinte forma: cada vez que o algoritmo localizar uma face que se percorrida em sentidos opostos forem encontradas subdivisões diferentes, este deverá executar uma busca em árvore. A esquerda de um nó da árvore indica que o algoritmo deverá crescer a região percorrendo as duas faces no mesmo sentido. Depois, encontrando ou não o isomorfismo, o algoritmo deverá realizar novamente o crescimento, mas percorrendo as duas faces em sentidos contrários, no lado direito do nó da árvore.

Para suportar esta busca em árvore, são necessárias mudanças na estrutura de dados, com o objetivo de ter acesso rápido nas duas direções. Tendo em vista a economia de espaço, o algoritmo pode percorrer os dois sentidos no padrão, ao invés da subdivisão de busca. Desta forma, o padrão buscado deverá ter acesso à vizinhança das arestas nos dois sentidos, podendo ser utilizada a *Winged Edge* para o seu armazenamento. A estrutura de dados para a subdivisão de busca não precisa ser alterada.

4.6.2 Pré-processamento do padrão

Dado um padrão, a identificação de suas transformações espelhadas pode ser realizada percorrendo-se as faces nos dois sentidos e verificando se elas apresentam características diferentes. Então, pode-se realizar um pré-processamento no padrão, de forma a gerar todas as suas combinações possíveis. Assim, o algoritmo não requer modificações, mas deverá processar cada um dos padrões gerados. Da mesma forma que na busca em árvore, o padrão deverá estar armazenado utilizando *Winged Edge*, mas os resultados do pré-processamento podem estar armazenados em estrutura DCEL.

4.7 Implementação e Testes

O algoritmo proposto nas Seções 4.2 e 4.3 foi implementado utilizando a linguagem C++. O programa utilizado para gerar arquivos contendo a descrição da subdivisão, a ser descrito na próxima Subseção, não retorna as faces. Então, também foi implementado um algoritmo para geração de ordem em subdivisões planares, como descrito no Capítulo 2. Desta forma, o programa implementado tem duas funcionalidades:

1. Dados arquivos de entrada contendo a descrição de vértices e arestas de uma subdivisão, o programa estabelece ordem na subdivisão planar, e armazena estes dados em um outro arquivo, já no formato de uma tabela, representando uma estrutura DCEL;

2. Dados dois arquivos contendo subdivisões planares, um representando a subdivisão de busca e o outro o padrão, o programa realiza o sub-isomorfismo do padrão na subdivisão, salvando o resultado em um terceiro arquivo.

4.7.1 Subdivisões de teste

Para gerar subdivisões de testes, foi utilizado o programa *triangle* [36]. Este programa é capaz de construir uma malha triangular dado um polígono envolvente e uma resolução, gerando arquivos com a descrição de vértices e arestas. A partir das malhas geradas, são removidas arestas aleatoriamente, mas de forma a garantir que cada vértice, ao final do processamento, possua pelo menos duas arestas no seu perímetro, mantendo a estrutura de dados fechada. Foi utilizada uma probabilidade de 15% de remoção de uma aresta (se ela puder ser removida), gerando subdivisões planares como a mostrada na Figura 4.11. Esta imagem foi gerada utilizando o programa *showme*, disponível juntamente com o programa *triangle*. Desta forma, as subdivisões planares geradas para os testes estão sempre envolvidas por um retângulo. Mas esta não é uma limitação do algoritmo, que também funciona para subdivisões sem face externa, desde que a subdivisão esteja sobre uma superfície.

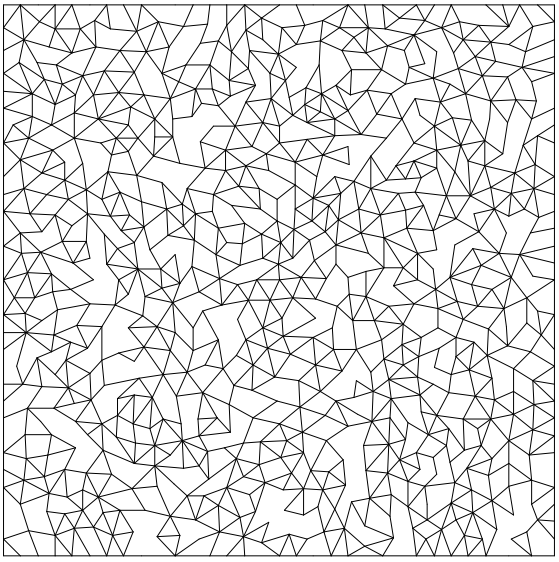
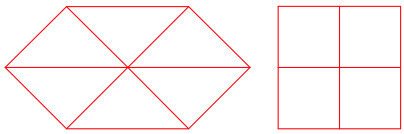


Figura 4.11: Exemplo de subdivisão planar gerada de forma aleatória

Os padrões gerados para teste devem ter uma face externa, caso contrário não existe qualquer sub-isomorfismo. Não foi encontrado qualquer programa gráfico para gerar arquivos de padrões no formato do programa *triangle*, portanto os padrões a serem buscados foram editados manualmente. A figura 4.12 mostra dois exemplos de padrões procurados nas subdivisões geradas.

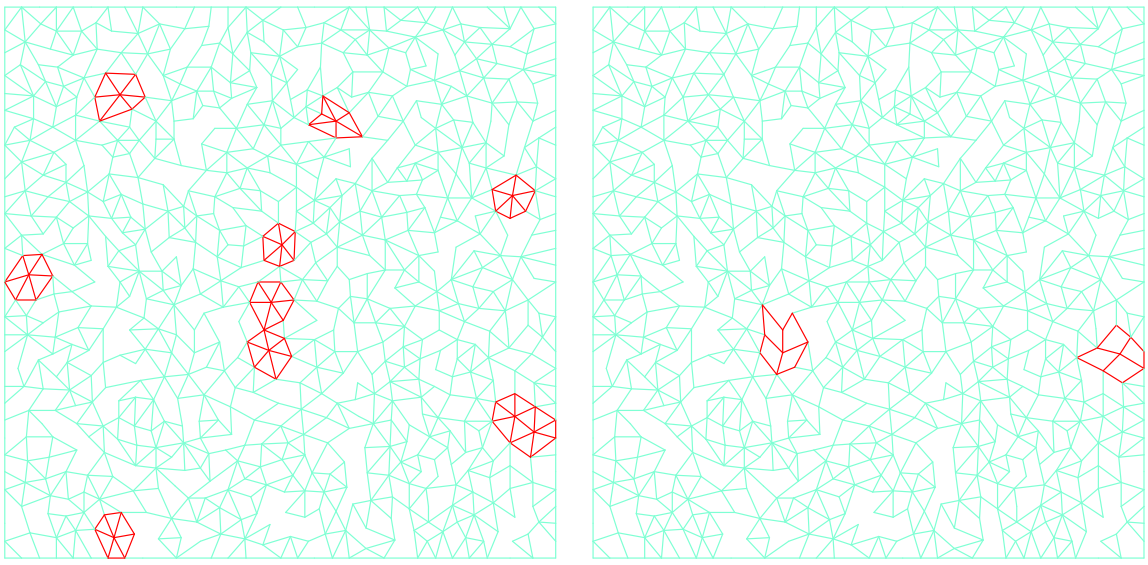


(a) (b)

Figura 4.12: Exemplos de padrões a serem buscados

4.7.2 Resultados das buscas

Para testes do algoritmo foi utilizado um computador Pentium IV 2.4GHz com 512Mb de RAM, sistema operacional Debian/Linux. A Figura 4.13 contém o resultado da busca dos padrões mostrados na Figura 4.12 em uma subdivisão gerada aleatoriamente. Na Figura 4.13(a) foram encontrados dez padrões, e dois deles compartilham cinco arestas, enquanto que na Figura 4.13(b) foram encontrados apenas duas ocorrências. Como a subdivisão foi gerada a partir de uma malha triangular, padrões que não seguem este formato são mais difíceis de serem encontrados.



(a) Resultado da busca do padrão da figura 4.12(a) (b) Resultado da busca do padrão da figura 4.12(b)

Figura 4.13: Resultado das buscas

O algoritmo foi testado com subdivisões de tamanhos variando entre 1.500 e 1.500.000 arestas, e os tempos obtidos com a busca do padrão da figura 4.12(a) podem ser visualizados na Figura 4.14. Como pode-se perceber, os resultados de tempo foram lineares conforme o tamanho da entrada, comprovando a análise da complexidade do algoritmo.

4.8 Considerações Finais

O algoritmo apresentado neste Capítulo realiza apenas o sub-isomorfismo topológico, e, desta forma, ele é exato, pois a topologia não tem como proporcionar uma solução aproximada. Em muitas aplicações de visão computacional, é necessário realizar o sub-isomorfismo geométrico e aproximado. Mas note que, pelo fato deste algoritmo ser linear, este pode ser modificado para suportar aproximações, mantendo a complexidade de tempo viável. Um exemplo de trabalho que apresenta operadores geométricos é o próprio trabalho de Lládos et al. [25]. A modificação deste algoritmo para suporte à geometria e a erros é um dos trabalhos a serem realizados no futuro.

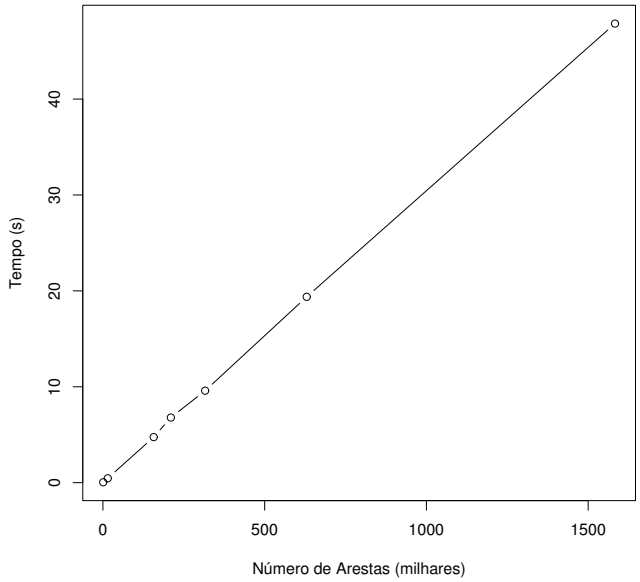


Figura 4.14: Tempos obtidos com a busca de padrões

Capítulo 5

Conclusões

A comparação de grafos é muito utilizada em visão computacional tanto em áreas como robótica como em CAD. Este trabalho tem como foco central as complexidades de tempo e espaço para resolver o problema do sub-isomorfismo em subdivisões planares. Aqui, o sub-isomorfismo é realizado utilizando apenas a topologia, sem qualquer comparação geométrica.

O uso do RAG, proposto por Lládos et al., nem sempre reduz significativamente o tamanho da subdivisão e, portanto, esta representação nem sempre tem efeito significativo na redução do tempo do algoritmo associado. Então, para realizar o sub-isomorfismo, foi proposta uma representação híbrida entre o dual e o RAG, de forma a possibilitar o seu armazenamento na mesma estrutura de dados da subdivisão.

O algoritmo apresentado realiza sub-isomorfismo de subdivisões planares baseado na comparação de arestas, e os vértices e faces são utilizados apenas para a verificação, ao contrário dos algoritmos mais conhecidos, que se baseiam nos vértices. Existem $2|A|$ possibilidades de equivalência de uma aresta do padrão na subdivisão de busca, e como a existência de faces mantém a estrutura de dados unida, o algoritmo tem apenas que verificar se as outras arestas se encaixam na subdivisão de busca. Desta forma, a verificação de um sub-isomorfismo dadas as duas arestas iniciais a serem equivalentes consome tempo igual ao tamanho do padrão. Como o padrão é bem menor que a subdivisão de busca, é necessário um tempo linear para a solução do problema.

O algoritmo adapta-se perfeitamente à estrutura de dados DCEL, utilizando todos os seus ponteiros e não necessitando dos ponteiros ausentes. O algoritmo foi implementado, e os testes comprovaram que a sua complexidade de tempo é linear.

Um estudo mais aprofundado no trabalho de Lládos et al. pode comprovar que o algoritmo por eles proposto possui complexidade de tempo polinomial, como mostrado nos seus testes, mas não por causa da redução do número de arestas pela geração do RAG, como justificado por ele, e sim pelo fato da estrutura de dados estar unida por causa das faces. Dois melhoramentos podem ser realizados no seu

algoritmo. O primeiro é o uso da representação híbrida aqui apresentada, e a segunda é a realização de uma busca por arestas no padrão, antes de iniciar o algoritmo, para determinar a ordem das arestas para realizar o crescimento de regiões.

Este trabalho também realizou uma comparação das estruturas de dados mais utilizadas para a representação de subdivisões planares. Foi mostrado que a DCEL é um contra exemplo de que são necessários $4|A|$ para armazenar os vizinhos de uma aresta, como proposto por Woo, pois não existe qualquer necessidade de se acessar todos os vizinhos de um elemento ao mesmo tempo. Os estudos que tem como foco principal o armazenamento das adjacências são muito teóricos, pois o computador é capaz de manipular apenas um valor por vez.

Também foi mostrado que a consulta de três operações básicas para realizar uma relação indireta implica em um gasto temporal significativo, como na estrutura de dados Δ, proposta por Ala. Uma exceção para este caso é quando o número de elementos adjacentes está limitado por uma constante.

Como trabalhos futuros pode-se citar:

- Adicionar verificação geométrica ao algoritmo proposto, de forma que sejam permitidas aproximações na busca dos padrões, mas mantendo a complexidade do algoritmo viável, se não for possível provar que o algoritmo proposto por Lládos et al. é polinomial;
- Utilizar o algoritmo em casos reais de aplicações de visão computacional, ao invés de utilizá-lo em subdivisões geradas aleatoriamente.

Referências Bibliográficas

[1] S. R. Ala. Design methodology of boundary data structures. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 13–23, Austin, Texas, EUA, 1991.

[2] S. R. Ala. Performance anomalies in boundary data structures. *IEEE Computer Graphics and Applications*, 12(2):49–58, March 1992.

[3] H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 15(5):522–525, 1993.

[4] B. G. Baumgart. Winged-edge polyhedron representation. Technical report, Stanford University, Computer Science Departament, report CS-320, October 1972.

[5] B. G. Baumgart. A polyhedron representation for computer vision. *AFIPS National Computer Conference*, pages 589–596, 1975.

[6] H. Bunke and B. T. Messmer. Recent advances in graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(1):169–203, 1997.

[7] W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 17(8):749–764, 1995.

[8] A. D. J. Cross, R. C. Wilson, and E.R. Hancock. A inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–970, 1997.

[9] C. M. Eastman. Introduction to computer aided design, 1979. Course Notes.

[10] A. M. Finch, R. C. Wilson, and E. R. Hancock. Matching Delaunay graphs. *Pattern Recognition*, 30(1):123–140, 1997.

[11] G. P. Ford and J. Zhang. A structural graph matching approach to image understanding. *Proceddings of SPIE Conference on Intelligent Robots and Computer Vision X: Algorithms and Techniques*, 1607:559–569, 1991.

[12] S. Gold and A. Rangarajan. A graduated assignment for graph matching. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[13] A. L. P. Guedes. Representação de variedades combinatórias de dimensão n. Master's thesis, COPPE/Engenharia de Sistemas e Computação – UFRJ, Rio de Janeiro – RJ, January 1995.

[14] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[15] T. C. Henderson. *Discrete Relaxation Techniques*. Oxford University Press, 1990.

[16] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184, Seattle, Washington, EUA, 1974. ACM Press.

[17] S. Jiaguang and C. Yujian. Data structures for geometric modeling. *Chinese Journal of Computing*, 12(3):181–193, 1989.

[18] X. Jiang, A. Munger, and H. Bunke. Synthesis of representative graphical symbols by computing generalized median graphs. In A. K. Chhabra and D. Dori, editors, *Graphics Recognition: Recent Advances*, pages 183–192. 2000.

[19] Y. E. Kalay. The hybrid edge: a topological data structure for vertically integrated geometric modeling. *Computer-aided Design*, 21(3):130–140, 1989.

[20] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[21] D. G. Kirkpatrick. Establishing order in planar subdivisions. In *Proceedings of the Third Annual Symposium on Computational Geometry*, pages 316–321, Waterloo, Ontário, Canadá, June 1987. ACM Press.

[22] D. E. Knuth. *Art of computer science, vol 3: sorting and searching*. Addison-Wesley, 1973.

[23] P. Kuner and B. Ueberreiter. Pattern recognition by graph matching: Combinatorial versus continuous optimization. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(3):527–542, September 1988.

[24] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on Computing*, 6(1):594–606, 1977.

[25] J. Lladós, E. Martí, and J. J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, October 2001.

[26] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, 1988.

[27] K. Mehlhorn. *Graph Algorithms and NP-Completeness*. Springer-Verlag, 1984.

[28] S. Mitchell, T. Beyer, and W. Jones. Linear algorithms for isomorphism of maximal outerplanar graphs. *J. ACM*, 26(4):603–610, 1979.

[29] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[30] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, October 1978.

[31] X. Ni. *Free-form Solid Modeling using a Hybrid CSG/Brep Environment*. PhD thesis, University of Leeds, Leeds – UK, December 1990.

[32] X. Ni and M. S. Bloor. Performance evaluation of boundary data structures. *IEEE Computer Graphics and Applications*, 1994.

[33] T. Pavlidis. *Graphics and Image Processing*. Computer Science Press, 1982.

[34] M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 21(11):1105–1200, November 1999.

[35] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[36] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. *First Workshop on Applied Computational Geometry*, pages 124–133, May 1996.

[37] H. Sossa and R. Horaud. Model indexing: The graph-hashing approach. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 811–814, June 1992.

[38] P. N. Suganthan, E. K. Teoh, and D. P. Mital. Pattern recognition by homomorphic graph matching using Hopfield neural networks. *Image and Vision Computing*, 13(1):45–60, February 1995.

[39] P. N. Suganthan, E. K. Teoh, and D. P. Mital. Pattern recognition using the Potts MFT neural networks. *Pattern Recognition*, 28(7):997–1009, 1995.

[40] K. Tombre, C. Ah-Soon, P. Dosch, A. Habed, and G. Masini. Stable, robust and off-the-shelf methods for graphics recognition. In *Proceedings of the 14th International Conference on Pattern Recognition*, pages 406–408, Brisbane, Austrália, 1998.

[41] K. Tombre, C. Ah-Soon, P. Dosch, G. Masini, and S. Tabbone. Stable and robust vectorization: How to make the right choices. *Lecture Notes in Computer Science*, pages 3–18, 2000.

[42] J. R. Ullman. An algorithm for subgraph isomorphism. *Journal Assoc. for Computing Machinery*, 23(1):31–42, 1976.

[43] S. Umeyama. An eingendecomposition approach to weighted graph matching problems. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 10(5):695–703, September 1988.

[44] K. Weiler. Edge-based data structures for solid modeling on curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985.

[45] P. R. Wilson. Data transfer and solid modeling. In M. J. Mozny, H. W. McLaughlin, and J. L. Encarnacao, editors, *Geometric Modeling for CAD Applications*, pages 130–140. Elsevier Science, 1988.

[46] R. C. Wilson and E. R. Hancock. A Bayesian compatibility model for graph matching. *Pattern Recognition Letters*, 17:263–276, 1996.

[47] T. C. Woo. A combinatorial analysis of boundary data structure schemata. *IEEE Computer Graphics and Applications*, 5(3):19–27, 1985.

[48] T. C. Woo and J. D. Wolter. A constant expected time, linear storage data structure for representing three-dimensional objects. *IEEE Transations on Systems, Man and Cybernetics*, 14(3):510–515, 1984.