Aplicação de Métodos de Segmentação de Imagens

Ana Fernanda Cancado[1] **, Arthur Matos**[1]
Gabriel Praes[1]**, Guilherme Otávio**[1]**, Júlia Pinheiro**[1]

[1] [1]Instituto de Ciências Exatas e Informática – PUC Minas
Curso de Ciência da Computação
Belo Horizonte, MG, Brasil

{afscancado, amatos, gabriel.nunes.1205431, gooliveira, jproque}@sga.pucminas.br

Abstract. *This paper presents two image segmentation methods based on graph partitioning. The first method uses a segmentation algorithm that employs a predicate to evaluate the evidence of boundaries between regions, ensuring efficiency and detail preservation in low-variability regions. The second method uses the graph-cuts technique, which performs segmentation by separating objects from the background.*

Resumo. *Este artigo apresenta dois métodos de segmentação de imagens baseados em particionamento de grafos. O primeiro método utiliza um algoritmo de segmentação que usa um predicado para avaliar a evidência de limites entre regiões, garantindo eficiência e preservação de detalhes em regiões de baixa variabilidade. O segundo método utiliza a técnica de graph-cuts, que realiza a segmentação ao separar objetos do fundo.*

1. Introdução

Nos últimos anos, diversas técnicas de segmentação de imagens foram desenvolvidas, cada uma com abordagens e algoritmos diferentes. Este artigo explora dois métodos eficientes para esse problema.

O primeiro método combina características de técnicas clássicas de agrupamento com algumas adaptações que aprimoram a qualidade da segmentação. A imagem é representada como um grafo não direcionado, onde cada pixel corresponde a um nó. As conexões entre os nós são estabelecidas entre pixels vizinhos, e cada aresta recebe um peso que mede a dissimilaridade entre os pixels conectados. Diferentemente dos métodos clássicos, este método ajusta o critério de segmentação para obter resultados melhores.

O segundo método é baseado em cortes de grafos (graph cuts). Seu objetivo é dividir a imagem em dois segmentos principais: ”objeto”e ”fundo”. O grafo é construído de forma semelhante, com cada pixel sendo representado por um nó e arestas conectando os nós para indicar a similaridade entre pixels. Existem dois nós extras: source (s), que representa o objeto, e sink (t), que representa o fundo. A segmentação é realizada minimizando uma função de energia que combina informações de bordas e regiões. Assim, é encontrado o menor corte no grafo, ou seja, o conjunto de arestas de menor custo que separa os nós em dois grupos.

2. Trabalhos Relacionados

O artigo *Efficient Graph-Based Image Segmentation*, de Pedro F. Felzenszwalb e Daniel P. Huttenlocher, apresenta um algoritmo que utiliza abordagens baseadas em grafos para segmentar imagens de forma rápida e precisa. O método identifica fronteiras entre regiões perceptualmente distintas, captura propriedades globais da imagem, opera em tempo quase linear e tem grande relevância para aplicações em visão computacional.

Por outro lado, o segundo artigo explora o uso de cortes em grafos para a segmentação de imagens, formulando o problema como uma rotulagem binária que combina propriedades regionais e de contorno. Ele apresenta a construção de grafos para separar objetos do fundo e discute a generalização para grafos direcionados, além de oferecer uma reotimização eficiente para edições em 3D.

3. Segmentação por agrupamento de componentes

Em um grafo, uma segmentação divide o conjunto de vértices em componentes, e cada componente é formado por vértices conectados por um subconjunto das arestas. A qualidade de uma segmentação pode ser medida pela semelhança dos elementos de um componente e pela diferença entre os elementos de componentes diferentes. Ou seja, as conexões dentro de um componente devem ter pesos menores do que entre vértices de componentes diferentes.

Para determinar se dois componentes devem estar separados é utilizado um predicado D, que avalia se existe evidências para um limite entre duas regiões. O predicado compara a dissimilaridade entre os dois componentes (nas bordas que os conectam) e a dissimilaridade interna dentro de cada componente.

A diferença interna de um componente é dada como o maior peso da árvore geradora mínima do componente:

$$\text{Int}(C) = \max_{e \in \text{MST}(C,E)} w(e)$$

Int(C) representa o ponto de ”ruptura”do componente C: ele só permanece conectado se considerarmos arestas com pesos maiores ou iguais a Int(C).

A diferença entre dois componentes C1 e C2 é a menor aresta que conecta um vértice de C1 a um vértice de C2:

$$\text{Dif}(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j))$$

O predicado D avalia se a diferença entre os dois componentes (Dif(C1,C2)) é maior do que a menor diferença interna entre os dois componentes:

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{se Dif}(C_1, C_2) > \text{MInt}(C_1, C_2) \\ \text{false} & \text{caso contrário} \end{cases}$$

MInt(C1,C2) é o mínimo da soma da diferença interna (Int) e de uma função de limiar τ:

$$\text{MInt}(C_1, C_2) = \min\left(\text{Int}(C_1) + \tau(C_1), \text{Int}(C_2) + \tau(C_2)\right)$$

$$\tau(C) = \frac{|C|}{k}$$

Quando o tamanho de um componentes é pequeno, Int(C) pode ser 0, então a função limiar corrige isso. A constante K funciona como um parâmetro para lidar com componentes pequenos.

3.1. Algoritmo

O algoritmo descrito por [Felzenszwalb and Huttenlocher 2005] recebe como entrada um grafo G=(V,E) e produz uma segmentação S, que divide os vértices em componentes.

1. Ordenação das Arestas: Ordena todas as arestas E em ordem não descrescente de peso, criando uma sequência $\pi = (o_1, o_2, \ldots, o_m)$.

2. Inicialização: Cada vértice começa como seu próprio componente. Inicialmente há n componentes separados.

3. Processamento das Arestas: Para cada aresta o_q (na ordem π), que conecta dois vértices v_i e v_j:

- Verifique em quais componentes v_i e v_j estão atualmente.
- Se os dois vértices estão em componentes diferentes:
  - Compare o peso da aresta $w(o_q)$ com o valor MInt.
  - Se $w(o_q)$ for menor ou igual a MInt, una os dois componentes.
  - Caso contrário, não faça nada.

4. Resultado Final: Após processar todas as arestas, a segmentação final S é formada pelos componentes resultantes.

3.2. Implementação do Algoritmo em C++

O algoritmo de segmentação foi implementado dessa maneira:

3.2.1. 1. Inicialização do Grafo

- A imagem PPM é convertida em um grafo, onde os vértices representam os pixels e as arestas conectam pixels vizinhos com pesos baseados na diferença de cor entre eles.
- A imagem é lida pelo método `lerPPM` e convertida para grafo pelo método `ppmParaGrafo`.
- A função `lerPPM` lê um arquivo de imagem no formato PPM, onde os pixels são armazenados em um vetor de tuplas (R, G, B).
- Após a leitura, a imagem é convertida em um grafo em que cada pixel é mapeado para um vértice e conexões entre pixels adjacentes são representadas como arestas com pesos baseados na diferença de cor.

3.2.2. 2. Ordenação das Arestas

- Todas as arestas do grafo são armazenadas em um vetor.
- As arestas são ordenadas em ordem não decrescente com base nos pesos (diferença entre pixels), utilizando a estrutura `Edge`.

3.2.3. 3. Segmentação com Union-Find

A implementação utiliza a estrutura de dados **Union-Find** para gerenciar e unir conjuntos disjuntos.

- O **Union-Find** gerencia os componentes conectados:
  - Cada pixel começa como um componente separado.
  - A função `unionSets` une dois componentes se:
    - Eles estão desconectados.
    - O peso da aresta entre eles é menor que o limite definido.
- Para cada aresta ordenada, os dois pixels conectados por ela são processados:
  - Se estiverem em componentes diferentes e o peso for menor que o limite, os componentes são fundidos.
  - Caso contrário, nada é feito.

3.2.4. 4. Construção dos Componentes

- Após processar todas as arestas, os pixels que pertencem ao mesmo componente são agrupados.
- Cada componente é um conjunto de vértices conectados que satisfazem o critério de similaridade.

A classe `ImageSegmentation` realiza a segmentação da imagem baseada em agrupamento de pixels. A ideia é usar os pesos das arestas (diferença de cor entre pixels) para dividir os pixels em regiões.

3.2.5. 5. Saída

- **Imagens Segmentadas**: Cada segmentação é salva como uma nova imagem no formato PPM, com cores aleatórias atribuídas a cada componente.
- **Informações da Segmentação**: O programa imprime:
  - O número de componentes gerados.
  - O tamanho de cada componente.

3.3. Resultados Obtidos

A seguir, apresentam-se os resultados obtidos com o código descrito anteriormente. Em um dos testes, a imagem utilizada como entrada é mostrada na Figura 1.

Figura 1. Imagem selecionada como entrada.

Como resultado, obteve-se essas três segmentações mostrada na Figura 2, em cada uma, com um valor de K diferente.

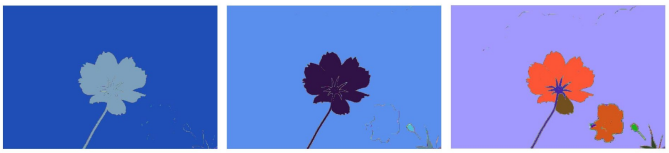

Figura 2. Resultado da segmentação.

4. Segmentação via Cortes em Grafos (Graph Cuts)

Outro método de segmentação explorado foi o de cortes em grafos, uma técnica eficiente para separar ”objeto”e ”fundo”de uma imagem. Para isso é preciso representar a imagem em um grafo:

- Cada pixel da imagem é representado como um nó em um grafo.
- Existem dois nós adicionais:
  - **Source (s)**: Representa o objeto.
  - **Sink (t)**: Representa o fundo.
- Os pixels vizinhos na imagem são conectados por arestas n-links, que representam divisória entre regiões. O peso da aresta é maior se os pixels conectados forem mais parecidos.
- Cada pixel é conectado a *s* ou a *t* por arestas t-links. Um peso menor indica que o pixel tem alta probabilidade de pertencer ao objeto ou ao fundo.

Corte mínimo

O objetivo é encontrar um corte mínimo no grafo, ou seja, remover um conjunto de arestas que separa os nós *fonte* e *sumidouro* com o menor custo total. O corte divide os nós em dois grupos: pixels atribuídos ao objeto e pixels atribuídos ao fundo.

Função de energia

O algoritmo minimiza uma função de energia composta por:

- **Termo de Região (R)**: Avalia a compatibilidade de cada pixel com o objeto ou fundo.
- **Termo de Borda (B)**: Penaliza cortes entre pixels muito similares para evitar bordas irregulares. (Se dois pixels vizinhos têm intensidades semelhantes, eles devem ser atribuídos à mesma classe)

  A função de energia pode ser representada como:

$$E(S) = R(S) + \lambda B(S),$$

Ao minimizar a função de energia, é possível encontrar uma segmentação que combine propriedades regionais e de borda.

Max-flow

Max-Flow (Fluxo Máximo) é usado para calcular o fluxo da fonte para o sumidouro no grafo, o que determina o corte mínimo da imagem entre ”objeto”e ”fundo”. Ao adicionar sementes, o algoritmo pode fornecer uma melhor segmentação, já que as sementes de ”objeto”e ”fundo”afetam as arestas t-links. Por exemplo, se um pixel foi identificado como uma semente de ”objeto”, o peso das arestas são determinados de uma forma que é altamente provável que esse pixel pertença ao objeto e improvável que pertença ao fundo.

4.1. Algoritmo

O algoritmo descrito por [Boykov and Funka-Lea 2006] segue o seguinte fluxo:

Entrada

- Pixels da imagem.
- Sementes de objeto e fundo.
- Parâmetro λ (peso entre termos regionais e de fronteira).

Etapas do Algoritmo

1. **Penalidades Regionais:** Para cada pixel p, calcular:

$$R_p(\text{obj}) = -\ln\big(Pr(I_p \mid \text{obj})\big), \quad R_p(\text{bkg}) = -\ln\big(Pr(I_p \mid \text{bkg})\big)$$

2. **Penalidades de Fronteira:** Para cada par de pixels vizinhos p, q, calcular:

$$B_{p,q} = \frac{\exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right)}{\text{dist}(p,q)}$$

3. **Construção do Grafo:** Criar nós para os pixels, conectar com:
   - *n-links:* entre pixels vizinhos usando $B_{p,q}$.
   - *t-links:* para a fonte (objeto) e o sumidouro (fundo) usando R_p.
4. **Corte Mínimo:** Aplicar um algoritmo de fluxo máximo/corte mínimo para separar objeto (conectado à fonte) do fundo (conectado ao sumidouro).

Saída

Segmentação da imagem: pixels conectados à fonte são classificados como objeto, e ao sumidouro, como fundo.

4.2. Implementação do algoritmo em C++

O algoritmo implementado segue as etapas descritas abaixo para dividir a imagem em duas partes.

4.2.1. 1. Leitura da Imagem (Classe `ImageReader`)

A função `readPPM` é responsável por ler uma imagem no formato PPM, armazenando dados de cada pixel em formato RGB. A imagem é lida e convertida em uma lista de objetos `Pixel`, onde cada um representa a cor de um pixel específico. A função retorna os pixels da imagem e suas dimensões.

4.2.2. 2. Construção do Grafo e Fluxo Máximo

A classe `ImageSegmentation` configura o grafo para o algoritmo de fluxo máximo:

- A fonte e o sumidouro são definidos, representando os vértices extras na matriz de pixels. Os pixels são conectados à fonte ou ao sumidouro com base na intensidade da cor.

- Os pixels vizinhos são conectados entre si, com um peso baseado na diferença de cor entre eles.
- O grafo é armazenado como uma matriz de adjacência (`graph`) e uma matriz residual (`residualGraph`), que armazena as capacidades restantes das arestas.
- O algoritmo de Ford-Fulkerson é utilizado para calcular o fluxo máximo entre a fonte e o sumidouro, segmentando a imagem com base nesse fluxo.

4.2.3. 3. Segmentação e Resultado

O método `minCutSegmentation` utiliza o corte mínimo (*min-cut*) para dividir os pixels da imagem em dois grupos: fundo e objeto. O corte mínimo é obtido após a execução do algoritmo de fluxo máximo. O resultado é uma lista de segmentos, onde cada segmento contém os índices dos pixels classificados como parte de uma das duas regiões.

4.2.4. 4. Escrita da Imagem Segmentada (Classe `ImageWriter`)

A função `saveSegmentationImage` salva a imagem segmentada em um novo arquivo no formato PPM. Os pixels são coloridos de maneira aleatória para representar os dois segmentos.

4.3. Resultados Obtidos

A seguir, é apresentado os resultados de um dos testes realizados com o código descrito anteriormente. A imagem utilizada como entrada é mostrada na Figura 3, e a divisão entre as duas regiões é evidenciado na Figura 4.

Figura 3. Imagem selecionada como entrada.

Figura 4. Resultado da segmentação da imagem.

5. Conclusão

Os métodos baseados em grafos demonstraram ser abordagens eficazes para a segmentação de imagens, oferecendo flexibilidade e precisão ao explorar as relações locais e globais entre pixels. Este trabalho apresentou duas técnicas distintas: segmentação

por agrupamento de componentes, que equilibra eficiência e preservação de detalhes, e cortes em grafos, que utiliza o fluxo máximo para separar regiões de interesse. Os resultados obtidos evidenciam a robustez e a adaptabilidade desses métodos em diferentes cenários.

Referências

Boykov, Y. and Funka-Lea, G. (2006). Graph cuts and efficient n-d image segmentation. *International Journal of Computer Vision*, 70(2):109–131.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Efficient graph-based image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2).