

Segmentação de Imagens baseados em Grafos: Clusterização e Corte

Gabriel C. Mourão¹, Luís A. L. Oliveira¹, Mateus F. Barbosa¹,
Victor F. Moraes¹, Vitória S. Araújo¹

¹Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Caixa Postal 30535-901 – Belo Horizonte – MG – Brazil

²Instituto de Ciências Exatas e Informática
ICEI

Abstract. This article discusses the operation of two different graph partitioning approaches, based on distinct application contexts in image segmentation. The objective is to understand the functioning of the employed algorithms and to evaluate the most suitable data structures to enhance the effectiveness and accuracy of the tested methods.

Resumo. Neste artigo, discute-se sobre o funcionamento de duas abordagens diferentes de particionamento de grafos, com base em contextos distintos de aplicação na segmentação de imagens. O objetivo é compreender o funcionamento dos algoritmos empregados, bem como avaliar as melhores estruturas de dados a serem utilizadas de modo a aumentar a eficácia e acurácia dos métodos testados.

1. Introdução

A segmentação de imagens é uma etapa fundamental no processamento digital de imagens, que busca dividir uma imagem em regiões ou objetos significativos. Essa técnica facilita a análise e a interpretação das imagens, destacando áreas de interesse com base em características como cor, textura ou forma. Os métodos de segmentação variam de abordagens baseadas em limiares simples a algoritmos mais avançados, como *Graph Cuts*, *Clustering*. Essas técnicas são amplamente utilizadas em áreas como visão computacional, diagnóstico médico e monitoramento ambiental, sendo essenciais para tarefas como detecção de objetos, reconhecimento de padrões e análise de cenas.

Neste contexto, o presente estudo visa apresentar duas abordagens de segmentação de imagens baseadas em grafos: *Pairwise Region Comparison Predicate* e *Combinatorial Graph Cuts*, que em princípio modelam imagens como grafos no qual os *pixels* são representados como nós conectados por arestas que indicam similaridade e continuidade entre outros *pixels*.

2. Conversão de Imagem em uma matriz numérica

O processo de conversão de imagens em matrizes numéricas é realizado por quatro algoritmos distintos, que garantem a interpretabilidade dos dados para os métodos de segmentação. Essas funções incluem a conversão de arquivos do formato PNG para PGM, a leitura e interpretação dos valores de intensidade de cor de cada pixel em arquivos PGM, a transcrição de vetores de inteiros em arquivos no formato PGM e, por fim, a conversão de arquivos PGM de volta para o formato PNG.

3. Pairwise Region Comparison Predicate

Proposto por [Felzenszwalb and Huttenlocher 2004], o algoritmo apresenta uma abordagem eficiente, com custo de $O(n \log(n))$, que busca diferenciar regiões de uma imagem. Este problema é recorrente em outras implementações que analisam apenas as diferenças locais entre os *pixels*.

Considere, por exemplo, uma imagem contendo uma parede branca e um gramado. Enquanto a parede branca apresenta uma aparência uniforme, o gramado possui alta variabilidade em suas cores. Um algoritmo que se baseia exclusivamente nas diferenças locais entre *pixels* pode apresentar dois problemas: incluir tanto o gramado quanto a parede em uma única região, ao considerar que, apenas variações muito altas entre *pixels* adjacentes são indicativos de distinção, ou dividir o gramado em várias regiões menores, ao adotar critérios baseados apenas em variações baixas entre *pixels*.

Apesar de os autores citarem outras abordagens, estas frequentemente apresentam implementações que não suficientemente capazes de analisar regiões de forma não local, ocasionando nos problemas citados anteriormente, ou implementações que são computacionalmente inviáveis para usos frequentes.

O algoritmo proposto procura resolver este problema utilizando um predicado de comparação de regiões para determinar se há evidências de uma fronteira entre dois componentes (ou regiões) em uma imagem. Este predicado é adaptativo e considera características locais dos dados, comparando a dissimilaridade entre componentes com a dissimilaridade dentro de cada componente.

3.1. Passo a passo do algoritmo

1. Inicialização: Cada pixel da imagem é tratado como um vértice pertencente a componentes únicos e são armazenados em um conjunto S . As arestas entre *pixels* adjacentes são ordenadas pelo peso (geralmente, diferenças de intensidade ou cor) em ordem crescente.
2. Repetir os itens 3 e 4 para todas as arestas.
3. Para cada aresta na ordem, verifica se os *pixels* conectados pertencem a componentes distintos.
4. Se $w(e)$ for menor quando comparado à $MInt(C_1, C_2)$, os componentes são unidos em S . $MInt$ consiste no cálculo da diferença mínima interna entre os componentes, calculada a partir da aresta de valor máximo da $MST(C_i)$ somada a uma função de limiar $\tau(C_i) = k/|C_i|$.
5. Finalização: Após processar todas as arestas, os componentes restantes são as regiões segmentadas armazenadas em S .

3.2. Estrutura da implementação do algoritmo

Para a implementação eficiente do algoritmo, foram usadas estruturas de dados de conjuntos disjuntos (*Disjoint Sets*), por união em *ranks*. Com isso, os passos de verificação da aresta máxima e união entre dois componentes podem ser feitas em complexidade $O(\alpha(m))$ em que α é uma função inversa de *ackermann*, ou seja, quase custo constante, para a complexidade total da análise em todas as arestas de $O(m\alpha(m))$.

Para a extração da lista de arestas, a imagem é analisada em apenas um canal de cores em que cada pixel é representado por apenas um valor. Cada pixel é percorrido e para cada aresta não existente entre seus vizinhos, esta é adicionada na lista com

custo $O(n)$, sempre menor que $O(m)$. O custo da ordenação desta lista é em média $O(m \log(m))$

Um método de *Gaussian Filter Smoothing* é aplicado nos vértices antes de gerar lista de arestas, para promover melhor união de *pixels* semelhantes e distinguir melhor as bordas entre grupos. O *kernel* aplicado possui tamanho 3x3, incluindo apenas os vizinhos de cada vértice, e de $\sigma = 0.8$.

A implementação usada neste artigo usa *grayscale* para simplificar a implementação ao transformar a imagem em preto e branco, em que apenas um canal precisa ser considerado.

Foi observado que, para imagens maiores, um k maior obtém agrupamentos melhores. O valor fornecido de k para MInt(C1, C2) variou entre 150, 300 e 600 para imagens com em média 120x120 *pixels*, 240x240 *pixels* e 480x480 *pixels*, respectivamente.

3.3. Resultados e discussões

Nessa seção, são apresentados alguns dos resultados obtidos ao aplicar o algoritmo de junção de conjuntos disjuntos para segmentação de imagens de teste. O algoritmo demonstrou sua capacidade de separar as diversas superfícies de imagens complexas de forma correta, gerando apenas alguns artefatos pequenos.



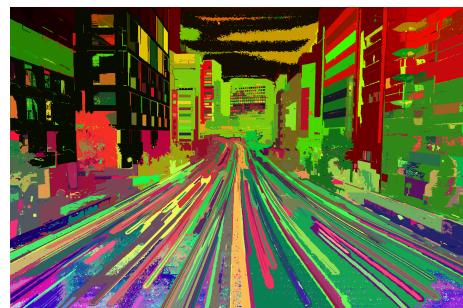
(a) Imagem original



(b) Resultado da segmentação



(a) Imagem original



(b) Resultado da segmentação

4. Combinatorial Graph Cuts

O algoritmo proposto por Boykov e Funka-Lea [Boykov and Funka-Lea 2006] utiliza métodos de corte de grafos, ou *graph cuts*, para realizar a segmentação de imagens. Essencialmente, a segmentação é um problema de divisão de uma imagem em diferentes regiões, com base na similaridade dos *pixels*. Nesse contexto, o grafo modela a imagem

onde cada pixel é representado por um nó, e as arestas conectam *pixels* vizinhos. Essas arestas possuem pesos que refletem a similaridade entre os *pixels* conectados, geralmente medidos pela diferença de intensidade ou outros critérios, como cor e textura. O objetivo é dividir a imagem de modo que os cortes minimizem a diferença entre as regiões segmentadas, enquanto maximizam a homogeneidade interna das regiões.

O processo de segmentação é então reduzido a um problema de corte mínimo em um grafo. Em termos simples, queremos cortar o grafo de forma que as arestas cortadas representem a separação das regiões, mas com um custo mínimo, o que significa que as regiões resultantes são as mais homogêneas possíveis. Isso é feito utilizando um algoritmo de fluxo máximo/corte mínimo.

Escolhemos o algoritmo *Ford-Fulkerson* para resolver esse problema de fluxo máximo, dado que ele é eficaz para determinar a quantidade máxima de fluxo que pode ser transportada de uma fonte (source) para um sumidouro (sink) em uma rede de fluxo. O algoritmo funciona em etapas iterativas, sempre buscando o caminho de maior fluxo possível e, em seguida, ajustando os fluxos nos caminhos até que nenhum caminho adicional possa ser encontrado. Esse processo permite encontrar a solução de corte mínimo, sendo a solução ideal para o problema de segmentação no contexto de Graph Cuts.

Entretanto, o algoritmo Ford-Fulkerson tem limitações em termos de desempenho. Para imagens de grandes dimensões, como aquelas com mais de 150x150 *pixels*, o custo computacional pode se tornar muito alto. O número de possíveis caminhos de fluxo e a quantidade de operações necessárias para calcular o corte mínimo crescem significativamente com o aumento da resolução da imagem. Isso pode resultar em tempos de execução elevados, tornando o algoritmo menos prático para imagens de grandes dimensões ou em sistemas com recursos limitados.

4.1. O Grafo

A representação é dada por:

- **Vértice:** Cada vértice representa a intensidade da cor de um pixel, em uma escala de cinza e, além disso, existem dois nós especiais denominados como *Source* (representa o objeto de segmentação) e *Sink* (representa o fundo da segmentação).
- **Arestas:** As arestas do grafo seguem duas abordagens:
 1. **T-Links:** Conectam os *pixels* ao nó de origem(*source*), representando o objeto ou o nó de destino(*sink*), representando o fundo, com pesos baseados em probabilidades de pertencimentos as classes.
 2. **N-Links:** Conectam *pixels* vizinhos com pesos baseados na similaridade de intensidades, priorizando a continuidade espacial.

4.1.1. Estrutura do Grafo

- **int vértices:** Número de vértices do grafo.
- **int source:** Vértice de origem no grafo (ponto inicial da rede de fluxo).
- **int sink:** Vértice de destino no grafo (ponto final da rede de fluxo).
- **unordered_map<int, unordered_map <int, int> capacity:** Estrutura de dados que armazena as capacidades das arestas no grafo, onde a chave externa representa um vértice e a chave interna representa outro vértice, com o valor sendo a capacidade da aresta entre eles.

- **unordered_map(int, unordered_map<int, int>)** **flow**: Estrutura de dados que armazena o fluxo das arestas no grafo, onde a chave externa representa um vértice e a chave interna representa outro vértice, com o valor sendo o fluxo atual na aresta entre eles.

4.2. Ford-Fulkerson e o Problema do Corte Mínimo

Para segmentar uma imagem em duas regiões distintas, utilizamos o algoritmo padrão de corte mínimo [Ford and Fulkerson 1962], que permite separar os *pixels* pertencentes ao “objeto” daqueles pertencentes ao “fundo”. Nesta abordagem, o grafo construído para a imagem é particionado em dois subconjuntos: S (source), representando o objeto, e T (sink), representando o fundo. O corte mínimo, por sua vez, identifica a borda que desconecta o grafo, separando os vértices alcançáveis a partir de S daqueles que não podem ser alcançados a partir de S, ou seja, que estão conectados a T.

4.2.1. Estrutura do algoritmo

- **bool findAugmentingPath(vector<int>&parent)**: Este método realiza uma Busca em Largura (BFS) no Grafo Residual, verificando a existência de um caminho de S (source) até T (sink).
- **int pushFlow(vector<int>&parent)**: Atualiza os fluxos no Grafo Residual com base no caminho encontrado, ajustando as capacidades das arestas.
- **int fordFulkerson(vector<int>&setS, vector<int>&setT)**: Implementa o algoritmo principal de Ford-Fulkerson para calcular o fluxo máximo entre dois conjuntos de vértices (S e T).

4.3. Passo a Passo do Algoritmo

1. Inicialização dos Histogramas: A função *computeHistograms* é chamada para calcular os histogramas de intensidade do objeto e do fundo, dividindo a imagem com base no valor médio das intensidades.
2. Criação do Grafo: O grafo é inicializado com o número de vértices (*pixels* da imagem) e dois vértices especiais: o vértice de origem (*source*) e o vértice de destino (*sink*).
3. Cálculo dos T-links:
 - O método *computeTlinks* calcula os t-links entre os *pixels* e os vértices de origem (objeto) e destino (fundo).
 - O peso das arestas entre os *pixels* e os vértices é baseado nas probabilidades calculadas a partir dos histogramas de intensidade do objeto e do fundo, utilizando a equação do logaritmo negativo:

$$Peso = -\ln \Pr(I_p | "obj") \quad e \quad Peso = -\ln \Pr(I_p | "bkg")$$

Onde I_i e I_j são as intensidades dos *pixels* i e j .

4. Cálculo dos N-links:
 - O método *computeNlinks* calcula os n-links entre *pixels* vizinhos (direita e abaixo), modelando a suavização das transições de intensidade entre os *pixels* vizinhos.

- O peso das arestas é calculado usando uma função gaussiana com base nas diferenças de intensidade entre os *pixels* vizinhos, controlada pelos parâmetros λ e σ :

$$Peso_{ij} = \lambda \cdot \exp \left(-\frac{(I_i - I_j)^2}{2\sigma^2} \right)$$

5. Busca por Caminho de Aumento (Ford-Fulkerson):

- O método *fordFulkerson* é utilizado para encontrar o fluxo máximo no grafo. O algoritmo busca caminhos de aumento a partir da origem até o destino, atualizando os fluxos nas arestas do grafo.
- Durante a execução, o fluxo é atualizado até que não haja mais caminhos de aumento, maximizando o fluxo entre o vértice de origem e o vértice de destino.

6. Criação da Máscara de Segmentação: Criação de uma máscara binária onde os *pixels* pertencentes ao conjunto S (com fluxo para a origem) são marcados como parte do objeto, e os pixels pertencentes ao conjunto T (com fluxo para o fundo) são marcados como parte do fundo.

7. Saída de Resultados: A máscara de segmentação é impressa no console e salva como um arquivo PGM (*segmented_output.pgm*).

4.4. Resultados e Discussões

Nesta seção, são apresentados os resultados obtidos ao aplicar o algoritmo de corte mínimo para segmentação das imagens de teste, o desempenho geral do algoritmo foi satisfatório, demonstrando boa capacidade de segmentação em cenários simples.



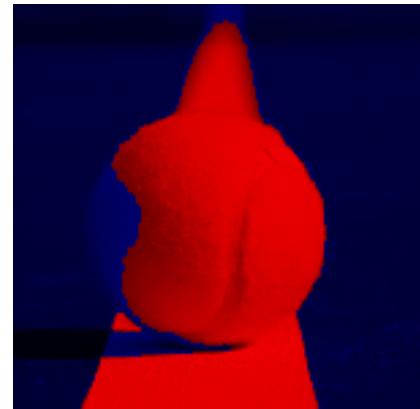
(a) Imagem original



(b) Resultado da segmentação



(a) Imagem original



(b) Resultado da segmentação

5. Divisão de Tarefas

O trabalho foi separado em grupos de forma a não sobrecarregar nenhum dos membros da equipe. A divisão de tarefas, portanto, foi feita da seguinte maneira:

Integrante	Algoritmo Atribuído	Tarefas Realizadas
Gabriel	P.R Comparison Predicate	Gaussian Smoothing, Extração das arestas
Luís	P.R Comparison Predicate	Estrutura de Conjuntos Disjuntos
Mateus	Combinatorial Graph Cuts	Tratamento de Imagem
Victor	Combinatorial Graph Cuts	Ford-Fulkerson
Vitória	Combinatorial Graph Cuts	Estrutura de Grafos

6. Conclusão

Neste artigo, exploramos duas abordagens distintas de segmentação de imagens baseadas em grafos: o algoritmo que usa do *Pairwise Region Comparison Predicate* e o método de *Combinatorial Graph Cuts*. Ambos os métodos demonstraram eficácia em segmentar imagens complexas, embora apresentem diferentes vantagens e desafios em termos de implementação e aplicação.

O algoritmo baseado em *Pairwise Region Comparison Predicate* destacou-se por sua simplicidade e eficiência computacional, sendo especialmente útil em aplicações que requerem segmentações rápidas com um custo computacional reduzido. A utilização de conjuntos disjuntos e a adaptação do predicado de comparação garantem uma segmentação robusta em diversas condições, desde que parâmetros adequados sejam definidos para imagens de diferentes dimensões.

Por outro lado, o método de *Combinatorial Graph Cuts* ofereceu maior precisão, permitindo a incorporação de informações probabilísticas sobre os *pixels* e suas relações com as classes de objeto e fundo. Apesar de sua alta complexidade computacional, que pode limitar seu uso em imagens de alta resolução, o método mostrou-se uma opção valiosa para aplicações que exigem segmentações altamente refinadas.

Ambos os métodos apresentaram resultados promissores e podem ser adaptados a diferentes cenários de segmentação de imagens.

7. References

References

- Boykov, Y. and Funka-Lea, G. (2006). Graph cuts and efficient nd image segmentation. *International journal of computer vision*, 70(2):109–131.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmen-
tation. *International journal of computer vision*, 59:167–181.
- Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press.
- Mourão, G. C., Oliveira, L. A. L., Barbosa, M. F., Moraes, V. F., and Araújo, V. S. (2024). Implementações de grafos. Acessado em: 15 dez. 2024.