Projeto e Análise de Algoritmos - Implementação 4

Mateus Leal Sobreira[1],
Victor Cabral de Souza Oliveira[2]

[1] Departamento de Ciência da Computação – PUC-Minas
Belo Horizonte, MG – Brasil

{mateus.sobreira, victor.cabral}@sga.pucminas.br

Abstract. *The objective of this work is to test two distinct image segmentation algorithms based on graph theory. The algorithm proposed by [Felzenszwalb and Huttenlocher 2004] focuses on the joining of components using lower weight edges, by means of a modified version of Kruskal's algorithm. On the other hand, the algorithm by [Boykov and Funka-Lea 2006] employs image segmentation to distinguish between "background" and "object", using a variation of the Ford-Fulkerson graph flow maximization algorithm.*

Resumo. *O objetivo deste trabalho é testar dois algoritmos distintos de segmentação de imagens, baseados em teoria de grafos. O algoritmo proposto por [Felzenszwalb and Huttenlocher 2004] se concentra na junção de componentes utilizando arestas de menor peso, mediante uma versão modificada do algoritmo de Kruskal. Por outro lado, o algoritmo de [Boykov and Funka-Lea 2006] emprega segmentação de imagem para distinguir entre "background" e "object", utilizando uma variação do algoritmo de maximização de fluxo em grafo, Ford-Fulkerson.*

1. Introdução

A segmentação de imagens é uma tarefa essencial em diversos campos da visão computacional, com o objetivo de dividir uma imagem em regiões relevantes. Essa tarefa é crucial para o entendimento e análise de imagens, sendo amplamente aplicada em áreas como reconhecimento de padrões, segmentação médica, automação industrial, entre outras.

Uma abordagem comum para segmentação é representar a imagem como um grafo, onde os pixels são modelados como nós e as relações entre eles, como arestas. Nesse cenário, a segmentação pode ser formulada como um problema de corte mínimo, com o objetivo de encontrar a melhor divisão entre o objeto e o fundo da imagem, minimizando o custo do corte.

Outra técnica de segmentação utiliza a divisão por componentes conexos, baseada em uma árvore geradora mínima, que agrupa pixels com características semelhantes, geralmente diferenças de intensidade ou cor.

2. Segmentação Baseada em Árvore Geradora Mínima

2.1. Pré-processamento da Imagem

Antes de iniciar o processo de segmentação, a imagem passa por uma etapa de pre-processamento, na qual, é alterado o seu esquema de cores de colorido, utilizando a notação RGB (Red, Green, Blue), para a notação de tons de cinza, apresentando somente o valor da intensidade dos pixels.

2.2. Criação do Grafo

Após a etapa de pré-processamento, a imagem é transformada em um grafo não direcionado caracterizado por $G = (V, E)$, onde o conjunto V é representado pelo conjunto de todos os pixels da imagem e o conjunto E representa a relação de vizinhança entre dois pixels, tal que estes estejam em um raio de oito direções. Cada aresta do conjunto E é ponderada a partir do valor absoluto da diferença entre a intensidade do pixel μ e a intensidade do pixel υ.

2.3. Junção de Componentes

Para apresentar a etapa de junção de Componentes, é necessária a conceituação da diferença interna de um componente, função limiar baseada no tamanho do componente e a diferença mínima interna entre dois componentes.

A diferença interna de um componente é definida como o maior peso presente na árvore geradora mínima de um componente C. Podemos escrever esta definição de acordo com a Equação 1.

$$Int(C) = max \quad W_{\{\upsilon,\nu\}} \quad , \quad \{\upsilon, \nu\} \in MST(C) \tag{1}$$

Já a diferença mínima interna entre dois componentes é definida pela Equação 2. Onde é utilizada uma função limiar τ que controla o grau de diferença entre dois componentes, para estes tenham a evidência de uma separação entre eles, na qual valores maiores de k, causam uma maior preferência em componentes maiores. A Equação 3 demonstra a função de limiar.

$$MInt(C_1, C_2) = min(Int(C_1) + \tau(C_1)\ ,\ Int(C_2) + \tau(C_2)) \tag{2}$$

$$\tau(C) = \frac{k}{|C|} \quad , \quad k \in \mathbb{R}^+ \tag{3}$$

O algoritmo de segmentação de imagem é baseado no trabalho escrito por [Kruskal 1956], que apresenta um método para extrair uma Árvore Geradora Mínima em um grafo. Os passos deste método são:

1. Extrair o Grafo nulo, tal que $G_n = (V', \emptyset) \quad , \quad V' = V_{original}$
2. Ordenar em ordem não-decrescente as arestas $\{\upsilon, \nu\}$ de acordo com o seu peso $W_{\{\upsilon,\nu\}}$
3. Inserir uma aresta entre dois componentes, tal que o número de componentes totais do Grafo diminua.
4. repetir o passo anterior até que o número de componentes seja igual a um.
5. O retorno do algoritmo de [Kruskal 1956] é uma AGM (Árvore Geradora Mínima) do Grafo Original.

A principal diferença do algoritmo descrito por [Felzenszwalb and Huttenlocher 2004] é a maneira de se inserir uma aresta ligando a dois componentes, já que além de buscar a aresta mínima que diminui a quantidade de

componentes gerais, esta tem que ter peso menor ou igual a diferença mínima interna entre seus dois componentes. A equação 4 representa esta relação.

$$\{\upsilon, \nu\} \in E \; sse \; W_{\{\upsilon,\nu\}} \leq MInt(C_1, C_2) \quad , \quad \upsilon \in C_1 \; e \; \nu \in C_2 \tag{4}$$

2.4. Análise de Complexidade

Neste algoritmo de segmentação de imagens, os trechos que impactam significativamente a execução são principalmente a leitura e processamento da imagem e a criação de componentes. O processo de conversão da imagem para um modelo de grafo também desempenha um papel importante no tempo de execução.

2.4.1. Leitura, Processamento e Criação das Arestas

A função responsável pela leitura e processamento da imagem converte os pixels da imagem em uma matriz de componentes e cria uma lista de arestas representando as diferenças de intensidade entre pixels adjacentes.

- A leitura da imagem e a conversão para uma matriz de componentes envolve a iteração sobre cada pixel da imagem, o que resulta em uma complexidade de $O(H \cdot W)$, onde H e W são a altura e a largura da imagem.

2.4.2. Mesclagem de Componentes

A função de mesclagem de componentes combina dois componentes com base na diferença de intensidade das arestas. A cada iteração, dois componentes são unidos, e os pixels de um são transferidos para o outro.

- A mesclagem de componentes pode envolver a iteração sobre todos os pixels de um componente, o que em termos de complexidade pode ser $O(H \cdot W)$ no pior caso.
- O número total de mesclagens de componentes depende das arestas, e pode ser da ordem de $O(H \cdot W)$.
- A complexidade total da mesclagem é $O(H \cdot W)$.

2.4.3. Complexidade Total

A complexidade total do algoritmo é a soma da complexidade das operações de leitura e processamento de imagem, criação de componentes e arestas, e mesclagem de componentes:

$$O(H \cdot W) + O(H \cdot W) = O(H \cdot W)$$

3. Segmentação Baseada em Fluxo Máximo

3.1. Pré-processamento da Imagem

Para o funcionamento do algoritmo de segmentação de imagens, é necessária a leitura das imagens e sua adaptação para a representação em grafo. Utilizou-se a biblioteca OpenCV (Open Source Computer Vision) em C++, que permite criar uma interface para a marcação das *seeds* pertencentes ao *background* ou ao *object*, além de possibilitar a leitura dos pixels da imagem.

Após a leitura, cada pixel da imagem é representado como um nó em uma matriz, contendo ponteiros para os pixels adjacentes e um peso, calculado conforme a Equação 5. Esse peso é posteriormente utilizado pelo algoritmo de fluxo para determinar as conexões no grafo.

$$W_{(u,v)} = \begin{cases} 1 & if I_u <= I_v \\ exp(-\frac{(I_u - I_v)^2}{2\sigma^2}) & if I_u > I_v \end{cases} \tag{5}$$

3.2. Implementação

Esse tipo de segmentação utiliza um algoritmo capaz de calcular o fluxo máximo em um grafo direcionado de um vértice inicial S para um vértice terminal T. No artigo [Boykov and Funka-Lea 2006] são mencionados 3 tipos de algoritmos de fluxo máximo: *Augmenting Path* (Ford-Fulkerson), *Push-Relabel* e Boycov-Kolmogorov. No código implementado foi utilizado o algoritmo de [Ford and Fulkerson 1956].

O algoritmo de utilizado é um método clássico utilizado para calcular o fluxo máximo em uma rede de fluxo. A ideia básica é encontrar caminhos aumentantes no grafo, ou seja, caminhos de um nó fonte S até um nó terminal T, ao longo dos quais ainda é possível enviar fluxo, isto é, há capacidade residual nas arestas.

O algoritmo segue os passos abaixo:

1. Inicialize o fluxo de todas as arestas como zero.
2. Enquanto houver um caminho aumentante P de S para T no grafo residual, faça:
    - Encontre a capacidade mínima (ou gargalo) do caminho P.
    - Aumente o fluxo nas arestas de P em uma quantidade igual ao gargalo.
    - Diminua a capacidade residual nas arestas de P de acordo com o fluxo enviado e aumente o fluxo reverso nas arestas.
3. Quando não houver mais caminhos aumentantes, o fluxo máximo será atingido.

A partir do fluxo máximo calculado, o corte mínimo pode ser determinado, que é o conjunto de arestas cujas capacidades residuais são positivas e que conectam o subgrafo alcançável de S ao subgrafo de T.

Após a divisão da imagem em dois subgrafos S e T os pixels do objeto são realçados com um aumento no nível de vermelho, enquanto os pixels do fundo são ajustados para tons de azul, assim possibilitando criar uma imagem que diferencie os dois conjuntos, assim como proposto no artigo.

3.3. Análise de complexidade

Para este algoritmo de segmentação, os trechos que impactam a execução são, principalmente, a Busca em Largura (BFS), utilizada para encontrar caminhos aumentantes no grafo residual; a execução do Ford-Fulkerson, que calcula o fluxo máximo e determina o corte mínimo; e a iteração sobre imagens, que envolve a conversão dos pixels para um modelo de grafo e o cálculo das capacidades das arestas com base nas intensidades dos pixels.

3.3.1. Busca em Largura (BFS)

A função BFS busca um caminho de aumento no grafo residual, explorando vértices e arestas. Ela utiliza uma fila para verificar os vértices acessíveis a partir do vértice fonte.

- A BFS percorre cada vértice e cada aresta no grafo uma vez.
- A complexidade dessa operação é $O(V + E)$, onde V é o número de vértices e E é o número de arestas.

3.3.2. Atualização de Fluxo

A atualização de fluxo ajusta as capacidades das arestas com base no caminho de aumento encontrado pela BFS. Essa operação percorre os vértices no caminho e ajusta o fluxo na direção direta e reversa.

- Para cada caminho de aumento, a complexidade é $O(V)$, pois o algoritmo percorre cada vértice do caminho.
- Então, a complexidade total dessa operação é de $O(P \cdot V)$, onde P é o número de caminhos de aumento encontrados.

3.3.3. Ford-Fulkerson (Loop Principal)

O algoritmo de [Ford and Fulkerson 1956] utiliza a BFS para encontrar repetidamente caminhos de aumento e atualizar os fluxos. A cada iteração, uma BFS é executada, e o fluxo é ajustado ao longo do caminho.

- O número de iterações depende do número de caminhos de aumento encontrados, denotado por P.
- Em cada iteração, são realizadas uma BFS ($O(V + E)$) e uma atualização de fluxo ($O(V)$).
- A complexidade de cada iteração é $O(V + E)$.
- A complexidade total do algoritmo de Ford-Fulkerson é $O(P \cdot (V + E))$, onde P é o número de caminhos de aumento encontrados.

3.4. Iteração nos pixels

A iteração sobre os pixels de uma imagem percorre todos os pixels e altera suas cores conforme os vértices visitados, no caso da criação do resultado.

- O número de operações é proporcional ao número de pixels da imagem, resultando em uma complexidade de $O(H \cdot W)$, onde H e W são a altura e a largura da imagem.

3.4.1. Complexidade Total

A complexidade total do algoritmo é a soma da complexidade das operações de BFS, atualização de fluxo, e o loop principal apresentado em [Ford and Fulkerson 1956]

$$O(P \cdot (V + E)) + O(H \cdot W),$$

onde P é o número de caminhos de aumento encontrados, V é o número de vértices, E é o número de arestas, e $H \times W$ é o número de pixels da imagem.

A complexidade desse algoritmo está mais atrelada a complexidade do loop principal do algoritmo [Ford and Fulkerson 1956] que, no pior caso pode ter um comportamento exponencial em relação ao fluxo máximo encontrado, assim pertencendo à classe de complexidade NP-Hard, uma classe onde os problemas não têm soluções eficientes conhecidas.

4. Resultados

4.1. Ambiente de teste

Os algoritmos foram escritos na linguagem C++ e executados em uma máquina com o processador Ryzen 5 4600H 3,00 Ghz, 24 Gigabytes de memória RAM, placa de vídeo Nvidea GeForce GTX 1650 4 GB, SSD SATA de 512 Gigabytes, Sistema Operacional Ubuntu 24.04.1 LTS e compilador G++ 13.3.0.

4.2. Resultados encontrados

Table 1. Tempo Médio de Execução

Exemplos	Algoritmo de Felzenszwalb	Algoritmo de Boykov
imagem1	2 ms	*11 ms
imagem2	2 ms	*13 ms
imagem3	57 ms	*247 ms
imagem4	56 ms	*182 ms

Os resultados do algoritmo de [Boykov and Funka-Lea 2006] demonstram a implementação mais atualizada que, de acordo com as imagens resultantes, não obteve a resposta esperada. O algoritmo de [Felzenszwalb and Huttenlocher 2004] utiliza como valor de $k = 420$ para o cálculo do $\tau(C)$.

4.3. Imagens de teste

Figure 1. Imagens de teste e resultados do algoritmo. a) Original. b) Algoritmo de [Felzenszwalb and Huttenlocher 2004]. c) Algoritmo de [Boykov and Funka-Lea 2006]

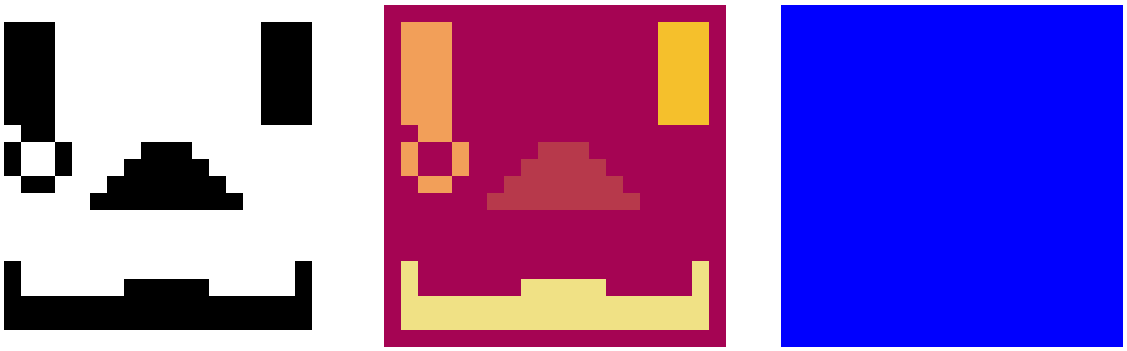

Figure 2. Imagens de teste e resultados do algoritmo. a) Original. b) Algoritmo de [Felzenszwalb and Huttenlocher 2004]. c) Algoritmo de [Boykov and Funka-Lea 2006]

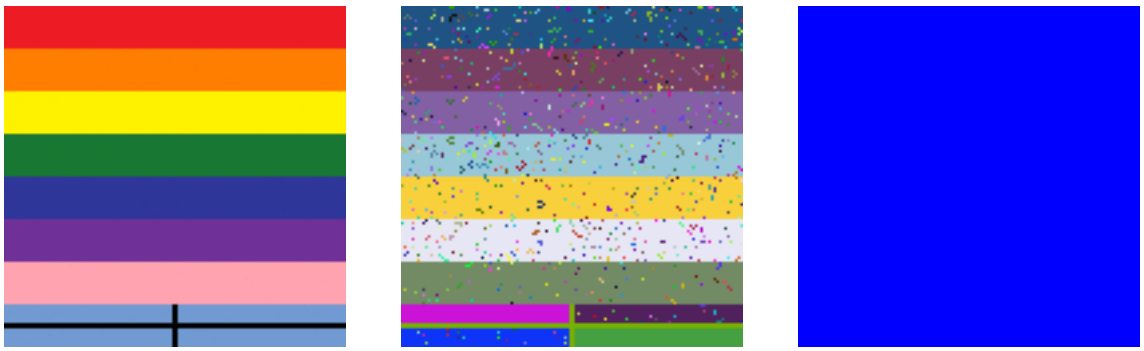

Figure 3. Imagens de teste e resultados do algoritmo. a) Original. b) Algoritmo de [Felzenszwalb and Huttenlocher 2004]. c) Algoritmo de [Boykov and Funka-Lea 2006]

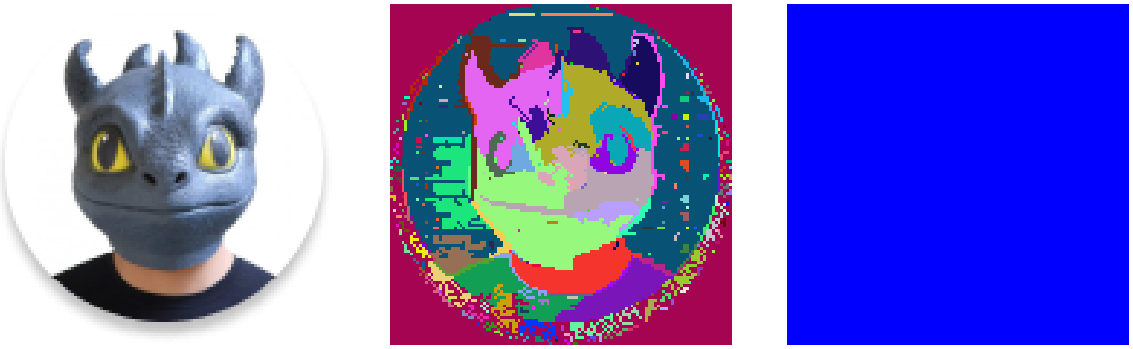

Figure 4. Imagens de teste e resultados do algoritmo. a) Original. b) Algoritmo de [Felzenszwalb and Huttenlocher 2004]. c) Algoritmo de [Boykov and Funka-Lea 2006]

5. Conclusão

O algoritmo de segmentação de imagens de Felzenszwalb e Huttenlocher [Felzenszwalb and Huttenlocher 2004] apresentou resultados superiores ao esperado, graças a otimizações realizadas que reduziram o custo computacional e o número de iterações necessárias. A abordagem mostrou-se eficiente e consistente com os resultados relatados pelos autores originais.

Por outro lado, o algoritmo de cortes mínimos em grafos de Boykov e Jolly [Boykov and Funka-Lea 2006] não atendeu às expectativas. Apesar de vários testes com diferentes grafos e configurações, os resultados obtidos ficaram aquém dos esperados quando aplicado ao contexto de segmentação de imagem, possivelmente devido a limitações na implementação ou nos parâmetros utilizados. Esses resultados destacam a importância de ajustes e estudos mais aprofundados para adaptar melhor cada técnica a diferentes cenários.

References

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

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181.

Ford, L. R. and Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.