Análise e Implementação de Árvores e Arborescências Geradoras Mínimas Aplicadas à Segmentação de Imagens

Alessandra Faria Rodrigues, Enzo Marques Pylo, Débora Luiza de Paula Silva, Gabriel Chaves Mendes, Guilherme Henrique da Silva Teodoro, Maria Eduarda P. Martins, Suzane Lemos de Lima [Pontifícia Universidade Católica de Minas Gerais]

Resumo

A construção de estruturas otimizadas em grafos é uma parte essencial da computação moderna e aparece em áreas como redes de comunicação, segmentação de imagens e análise de grandes volumes de dados. Entre essas estruturas, duas ganham destaque: a árvore geradora mínima (MST) e a arborescência geradora mínima (DMST ou AGS). Apesar de tratarem de problemas parecidos, cada uma apresenta características próprias e certos desafios específicos. Esses problemas vêm sendo estudados há muitos anos e deram origem a algoritmos que se tornaram clássicos, como os propostos por Edmonds, Tarjan e Gabow. Com o avanço das aplicações práticas, principalmente em visão computacional e processamento de imagens, a comparação entre essas técnicas voltou a ganhar importância. Métodos que antes eram considerados suficientes precisam ser reavaliados diante das exigências atuais, que pedem mais eficiência, precisão e robustez. O objetivo deste trabalho é analisar esses problemas de forma aplicada, implementando algoritmos tanto para árvores quanto para arborescências mínimas e avaliando o comportamento de cada método em tarefas de segmentação de imagens. Mais do que reproduzir soluções já conhecidas, busca-se observar na prática como essas abordagens funcionam, comparando vantagens, limitações e o grau de adequação a diferentes situações computacionais. O projeto envolve também uma integração entre teoria de grafos, implementação em C e C++, além de métodos experimentais de avaliação. A intenção é apresentar uma visão clara e fundamentada sobre como essas estruturas podem ser usadas em problemas reais, contribuindo para uma compreensão mais profunda e crítica dos algoritmos estudados.

Keywords: Grafos, Segmentação de Imagens, MST, Arborescência, Algoritmos.

1 Introdução

A busca por estruturas eficientes em grafos é um tema central na ciência da computação, principalmente quando se trata de árvores geradoras mínimas (MST) ou arborescências mínimas (DMST). Embora os dois problemas estejam relacionados, cada um possui particularidades que motivaram décadas de estudo e levaram ao desenvolvimento de algoritmos clássicos, como os de Edmonds, Tarjan e Gabow.

Com o avanço das tecnologias e o aumento constante no volume de dados, a comparação prática entre essas abordagens voltou a despertar interesse. A área de segmentação de imagens é um exemplo de aplicação que se beneficia bastante dessas estruturas, aproveitando suas propriedades de conectividade e representação mínima.

Neste trabalho, pretende-se implementar e analisar algoritmos para MST e arborescências mínimas, aplicando esses métodos ao problema de segmentação de imagens. A proposta é compreender tanto o funcionamento teórico quanto o desempenho prático, discutindo vantagens, limitações e possíveis cenários de uso.

2 Implementação

2.1 Estruturas de Dados

A representação dos grafos utiliza **listas de adjacência**, adequada para grafos esparsos. A estrutura *Union-Find* com *path compression* e *union by rank* é empregada tanto no pré-processamento (agrupamento de pixels) quanto nos algoritmos para detecção e contração de ciclos, garantindo complexidade amortizada $O(\alpha(n))$ por operação.

2.2 Estratégia de Superpixels

A modelagem direta de pixels como vértices ($V = N \times M$) resultaria em complexidade proibitiva. Adotamos uma estratégia de pré-processamento baseada em Superpixels, inspirada em abordagens como o SLIC [8]: pixels vizinhos com similaridade de cor (distância Euclidiana RGB < 15) são agrupados via Union-Find, formando vértices lógicos.

Esta técnica reduziu grafos de 10^5 pixels para 10^2–10^3 supernós (redução de 99%), viabilizando execução em tempo real. Um filtro de média (*Box Blur*) é aplicado previamente para reduzir ruído e evitar super-segmentação.

2.3 Algoritmo de Edmonds

O algoritmo clássico de Edmonds (Chu-Liu, 1965) foi implementado como referência, seguindo a formulação recursiva com seleção gulosa, detecção de ciclos e contração de super-nós. Sua complexidade $O(VE)$ o torna inadequado para grafos maiores, motivando a implementação dos métodos otimizados.

2.4 Algoritmo de Tarjan (1977)

O algoritmo de Tarjan melhora a complexidade através do uso de **filas de prioridade mescláveis**. A ideia central é manter, para cada componente do grafo, uma heap contendo todas as arestas de entrada ordenadas por peso.

Fase de Contração: O algoritmo processa vértices iterativamente. Para cada vértice v, seleciona-se a aresta de entrada de menor peso da heap correspondente. Se esta aresta forma um ciclo (detectado via marcação de estados), os componentes do ciclo são contraídos em um super-nó. As he-

aps dos componentes são *fundidas* (operação *meld*), e os pesos são ajustados via **propagação preguiçosa** (*lazy propagation*): ao invés de atualizar cada aresta individualmente, armazena-se um valor de ajuste no nó raiz da heap, propagado apenas quando necessário.

Remoção de Auto-loops: Durante a seleção da aresta mínima, arestas que se tornaram auto-loops (origem e destino no mesmo componente após contrações) são descartadas da heap. A complexidade resultante é $O(E \log V)$, onde o fator logarítmico deriva das operações de heap.

2.5 Algoritmo de Gabow, Galil, Spencer e Tarjan (1986)

O algoritmo de Gabow et al. refina a abordagem de Tarjan através da técnica de **Path Growing** (Crescimento de Caminho) e uma estrutura hierárquica para reconstrução da solução.

Path Growing: Ao invés de processar vértices isoladamente, o algoritmo estende "caminhos" a partir de vértices não processados. Cada vértice assume um de três estados: *novo*, *ativo* (em processamento) ou *processado*. Quando um caminho encontra um vértice ativo, um ciclo é detectado e contraído.

Contração Hierárquica: Ciclos são registrados em uma pilha com informações sobre seus componentes e as arestas que os conectam. Cada super-nó mantém referência ao seu "pai" na hierarquia de contrações, permitindo rastrear a qual ciclo original cada vértice pertence.

Expansão: Após processar todo o grafo, a pilha de ciclos é desempilhada em ordem reversa. Para cada ciclo, determina-se qual sub-componente recebe a aresta externa (que entra no super-nó) e quais mantêm suas arestas internas originais. Esta fase reconstrói a arborescência no grafo original.

2.6 Decisão de Projeto: Skew Heaps

Para ambos os algoritmos (Tarjan e Gabow), optou-se por Skew Heaps em substituição aos Fibonacci Heaps sugeridos na literatura original. Esta decisão fundamenta-se nos princípios de Engenharia de Algoritmos (Böther et al., 2023): embora Fibonacci Heaps ofereçam complexidade teórica ótima $O(m + n \log n)$, suas constantes ocultas e overhead de memória frequentemente resultam em desempenho inferior para instâncias práticas.

As Skew Heaps são árvores binárias auto-ajustáveis propostas por Sleator e Tarjan [7], que suportam *merge*, *findMin* e *deleteMin* eficientemente. A operação de fusão, essencial para contração de ciclos, é implementada recursivamente. A implementação inclui *lazy propagation* para ajuste de pesos. Considerando que nossa estratégia de Superpixels já reduz o grafo para centenas de vértices, a complexidade $O(m \log n)$ das Skew Heaps é virtualmente indistinguível da ótima, garantindo código mais robusto.

2.7 Algoritmo de Kruskal (Baseline)

O algoritmo de Kruskal para MST não-direcionada foi implementado como *baseline* comparativo, utilizando ordenação de arestas e *Union-Find*. Os experimentos demonstraram que métodos direcionados (Tarjan, Gabow) preservam melhor gradientes de cor em transições suaves, validando a escolha da modelagem por arborescência.

3 Experimentos e Resultados

Os experimentos foram conduzidos em um processador Ryzen 7 5700X3D com 32 GB de memória RAM. O conjunto de dados principal consiste em imagens de resoluções variadas, pré-processadas conforme a estratégia de Superpixels descrita na Seção 1.2. Para a construção do grafo lógico, utilizou-se um limiar de similaridade de cor $\delta_{pixel} = 15$ (distância Euclidiana RGB) para o agrupamento de superpixels.

Na etapa de segmentação final, os limiares de corte normalizado foram ajustados conforme a resolução: utilizou-se $\lambda = 0.06$ para a imagem média e $\lambda = 0.006$ para a imagem grande, visando manter a consistência da granularidade nas diferentes escalas (Tabela 1).

Para a segmentação das imagens, foram empregados os algoritmos de Kruskal, Tarjan e Gabow, que apresentaram desempenho superior. O algoritmo de Kruskal, com complexidade $O(E\ log\ V)$, gerou a árvore geradora mínima não-direcionada, limitando-se a relações bidirecionais. Já os métodos de Tarjan e Gabow, otimizados com Skew Heaps, reduziram drasticamente o custo de união de conjuntos e contrações de ciclos, operando com complexidade amortizada competitiva de $O(E\ log\ V)$. O algoritmo de Edmonds foi testado em todas as instâncias para validar a correção da solução, servindo como baseline de desempenho para as implementações otimizadas (Tarjan e Gabow).

Tabela 1. Comparação de Tempo (ms) por Resolução

Algoritmo	**Complexidade**†	**Img. Média** (1.169 nós)	**Img. Grande** (7.546 nós)
Edmonds	$\overline{O(VE)}$	2.117 ms	260.248 ms
Kruskal	$O(E \log V)$	< 1 ms	2 ms
Tarjan	$O(E \log V)$	< 1 ms	4 ms
Gabow	$O(m \log n)$	< 1 ms	4 ms

3.1 Discussão dos Resultados

A Tabela 1 apresenta um contraste de desempenho expressivo. Na imagem média, o algoritmo de Edmonds exigiu cerca de 2 segundos, enquanto Tarjan, Gabow e Kruskal executaram instantaneamente ($<$ 1 ms). Na imagem grande (7.546 nós), a diferença de complexidade tornou-se crítica: Edmonds levou aproximadamente 260 segundos (4,3 min), enquanto as versões otimizadas processaram a mesma estrutura em apenas 4 ms.

Isso representa um *speedup* prático superior a 65.000x para o algoritmo de Gabow, validando empiricamente a necessidade de estruturas como Skew Heaps para aplicações de tempo real.

Qualitativamente, observou-se uma divergência na granularidade apenas na instância média, onde Gabow identificou mais regiões (469) que os demais (422/436). Contudo, na imagem grande, todos os métodos (incluindo Edmonds e

Kruskal) convergiram para exatas 7.512 regiões. Essa convergência na alta resolução reforça a correção das implementações otimizadas, demonstrando que a agilidade do método de Gabow não compromete a precisão da arborescência calculada.

A coincidência numérica entre Kruskal (não-direcionado) e os métodos direcionados nesta instância sugere que, para esta topologia de superpixels em grade, a direcionalidade não foi o fator limitante primário para as fronteiras, embora as otimizações de Gabow tenham oferecido o melhor balanço entre sensibilidade e desempenho extremo.

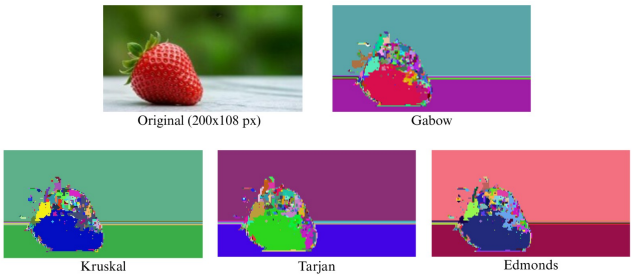

Figura 1. Comparação Visual. Resultados de segmentação obtidos pelos métodos de Gabow (469 regiões), Kruskal (422), Tarjan (436) e Edmonds (422) comparados à original.

4 Conclusão

Este trabalho demonstrou que, embora teoricamente complexa, a aplicação de arborescências geradoras mínimas em segmentação de imagens torna-se viável mediante o uso de Superpixels e estruturas de Heap eficientes. O algoritmo de Gabow (GGST) destacou-se como a solução definitiva, reduzindo o tempo de processamento de minutos (no clássico Edmonds) para milissegundos, sem perda de qualidade na segmentação.

Declarações

Authors' Contributions

A execução do projeto foi estruturada da seguinte forma: Débora e Alessandra ficaram responsáveis pela implementação dos algoritmos; Gabriel realizou a análise, documentação e organização do código; Enzo, Guilherme e Suzane desenvolveram o relatório e executaram os experimentos; Maria Eduarda organizou a versão final do relatório. O integrante Douglas Nicolas Silva Gomes contribuiu na primeira parte do trabalho, mas desligou-se do grupo antes da segunda etapa.

Competing interests

Os autores declaram que não possuem interesses conflituosos.

Availability of data and materials

Os códigos fonte e datasets gerados durante este estudo estão disponíveis mediante solicitação.

Referências

[1] Böther, M., et al. *Efficiently Computing Directed Minimum Spanning Trees*. In: 2023 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX), pp. 86-95, 2023.

[2] Edmonds, J. *Optimum Branchings*. Journal of Research of the National Bureau of Standards Section B, vol. 71B, no. 4, pp. 233-240, 1967.

[3] Tarjan, R. E. *Finding Optimum Branchings*. Networks, vol. 7, pp. 25-35, 1977.

[4] Gabow, H. N.; Galil, Z.; Spencer, T.; Tarjan, R. E. *Efficient algorithms for finding minimum spanning trees in undirected and directed graphs*. Combinatorica, vol. 6, no. 2, pp. 109-122, 1986.

[5] Kruskal, J. B. *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proceedings of the American Mathematical Society, vol. 7, no. 1, pp. 48-50, 1956.

[6] Cousty, J.; Najman, L.; Kenmochi, Y.; Guimarães, S. J. F. *Hierarchical Segmentations with Graphs: Quasi-flat Zones, Minimum Spanning Trees, and Saliency Maps*. J. Math. Imaging Vis., vol. 60, no. 4, pp. 479-502, 2018.

[7] Sleator, D. D.; Tarjan, R. E. *Self-adjusting heaps*. SIAM Journal on Computing, vol. 15, no. 1, pp. 52-69, 1986.

[8] Achanta, R.; Shaji, A.; Smith, K.; Lucchi, A.; Fua, P.; Süsstrunk, S. *SLIC Superpixels Compared to State-of-the-Art Superpixel Methods*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274-2282, 2012.