Análise Comparativa entre os Algoritmos de Kruskal e Prim em Grafos Esparsos e Densos

Lucas Rivetti, Lucas Campelo, Augusto Cezar, José Lopes, Bruno Henrique

16 de julho de 2025

Resumo

Este relatório apresenta uma análise teórica e prática dos algoritmos de Kruskal e Prim para construção de Árvores Geradoras Mínimas (AGM) em grafos esparsos e densos. São descritos: definição de AGM e propriedades, detalhes de implementação em C, experimentos de desempenho (tempo, operações, uso de memória) e análise comparativa com tabelas e gráficos.

Sumário

1 Introdução

Este trabalho, desenvolvido na Disciplina de Grafos da Universidade Federal de São João del-Rei sob orientação da Profª Jesuliana N. Ulysses, tem como objetivo comparar os algoritmos de Kruskal e Prim em termos de desempenho e aplicabilidade em grafos com diferentes características estruturais (esparsos e densos). Ambos os algoritmos foram implementados em C e avaliados em quatro tipos de grafos: esparsos aleatórios, densos aleatórios, malha 2D e grafos reais de pequena dimensão.

O relatório está organizado da seguinte forma: na Seção 2 apresenta-se a fundamentação teórica sobre AGMs e as propriedades do ciclo e do corte; na Seção 3 detalham-se os algoritmos de Kruskal e Prim; na Seção 4 descreve-se a metodologia de implementação e os conjuntos de testes; na Seção 5 são apresentados os resultados e suas análises, com tabelas e gráficos; na Seção 6 discute-se os achados; na Seção 7, a conclusão.

2 Fundamentação Teórica

2.1 Árvore Geradora Mínima (AGM)

Uma *árvore geradora* de um grafo não direcionado e conectado é um subgrafo que inclui todos os vértices, é acíclico e conexo. Quando atribuímos um peso a cada aresta, definimos o *peso da árvore* como a soma dos pesos de suas arestas. Uma *árvore geradora mínima* (AGM) é aquela cuja soma de pesos é mínima entre todas as árvores geradoras possíveis do grafo.

Dois teoremas garantem métodos gulosos para encontrar AGMs:

- **Propriedade do Ciclo**: em qualquer ciclo, a aresta de maior peso não pertence a nenhuma AGM.
- **Propriedade do Corte**: em qualquer partição dos vértices, a aresta de menor peso que cruza o corte pertence a toda AGM.

2.2 Paradigma Guloso

Algoritmos gulosos constroem soluções passo a passo, sempre escolhendo a opção de menor custo local. As propriedades do ciclo e do corte asseguram que decisões locais levam à solução ótima global para AGMs.

3 Algoritmos

3.1 Kruskal

Kruskal ordena todas as arestas por peso crescente e as adiciona à AGM se não formarem ciclo, detectado via *union-find*. A cada aresta:

1. Verifica-se se conecta componentes distintos (`find`).
2. Se sim, une componentes (`union`) e inclui a aresta.
3. Caso contrário, descarta a aresta.

Complexidade: $O(n \log n)$ para ordenação das n arestas, com `find`/`union` em $O(\alpha(n))$.

3.2 Prim

Prim cresce uma única árvore, iniciando em vértice arbitrário. A cada iteração:

- Extrai do *min-heap* o vértice de menor chave.
- Atualiza chaves dos vizinhos não visitados.

Complexidade: $O(n \log n)$ com heap; $O(n^2)$ em grafos densos usando matriz de adjacência.

4 Metodologia

Implementação em C (`src/kruskal.c`, `src/prim.c`), scripts Python para geração de grafos, Bash para automação de testes (`executar_testes.sh`).

Conjuntos de testes:

- Grafos esparsos: 1000 vértices, ≈1500 arestas;
- Grafos densos: 500 vértices, ≈100000 arestas;
- Malha 2D: 30×30;
- Grafos reais: 50 vértices, 75 arestas.

Métricas coletadas: tempo, número de operações (`find`/`union`, `extraiMin`/`diminuiChave`), uso de memória, correção.

5 Resultados e Análises

Neste ponto, apresentamos tabelas, gráficos e comparativos integrando dados tabulares e documentos ilustrativos em imagens.

Grafo	Algoritmo	Tempo (s)	Operações	Memória (B)
Esparso	Kruskal	0.00065	finds: 7180; unions: 999	26 000
Esparso	Prim	0.00104	extraiMin: 1000; diminuiChave: 1380	64 000
Denso	Kruskal	0.02910	finds: 7430; unions: 499	1 204 000
Denso	Prim	0.04040	extraiMin: 500; diminuiChave: 2270	3 208 000
Grade 2D	Kruskal	0.00042	finds: 8524; unions: 899	28 080
Grade 2D	Prim	0.00059	extraiMin: 900; diminuiChave: 899	70 080
Real	Kruskal	0.00005	finds: 320; unions: 49	1 300
Real	Prim	0.00004	extraiMin: 50; diminuiChave: 67	3 200

Tabela 1: Resumo dos resultados médios de desempenho por tipo de grafo e algoritmo.

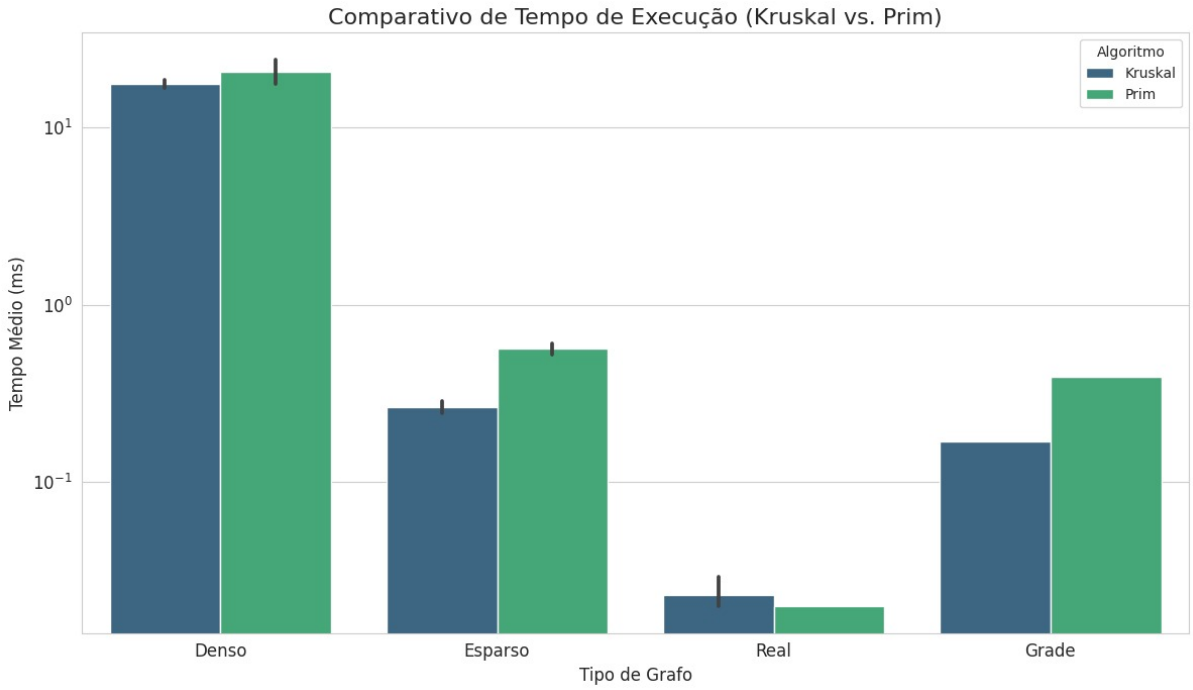

Figura 1: Comparativo de Tempo de Execução (Kruskal vs. Prim) nos diferentes tipos de grafos.

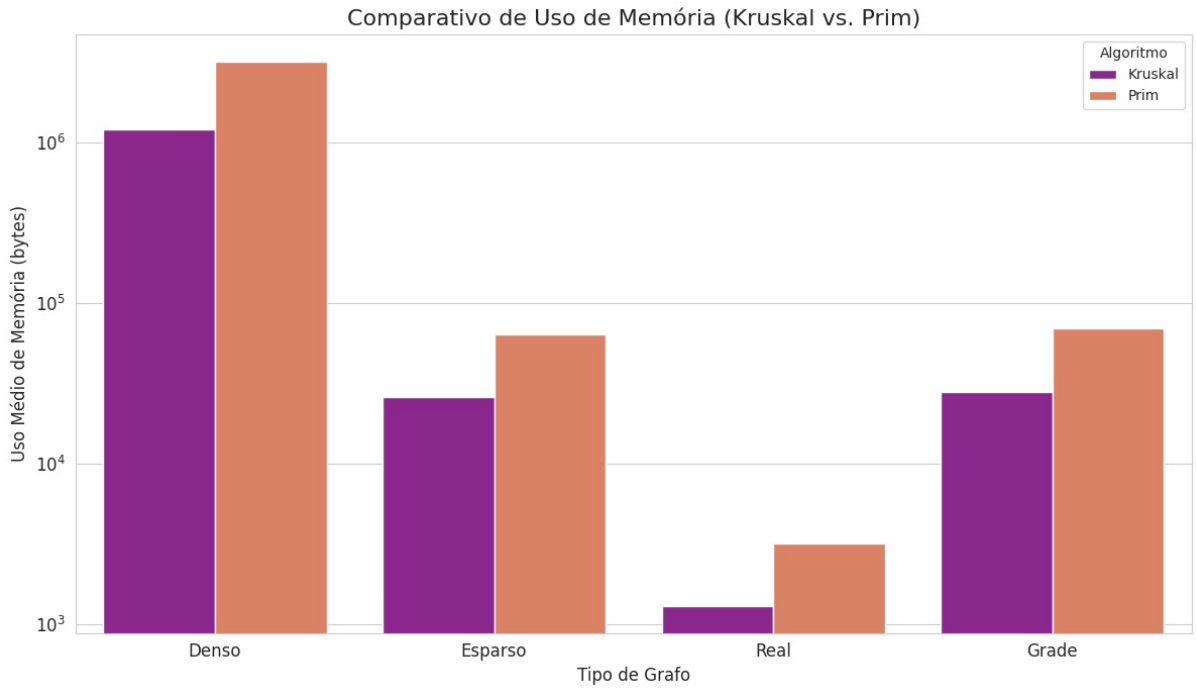

Figura 2: Comparativo de Uso de Memória (Kruskal vs. Prim) nos diferentes tipos de grafos.

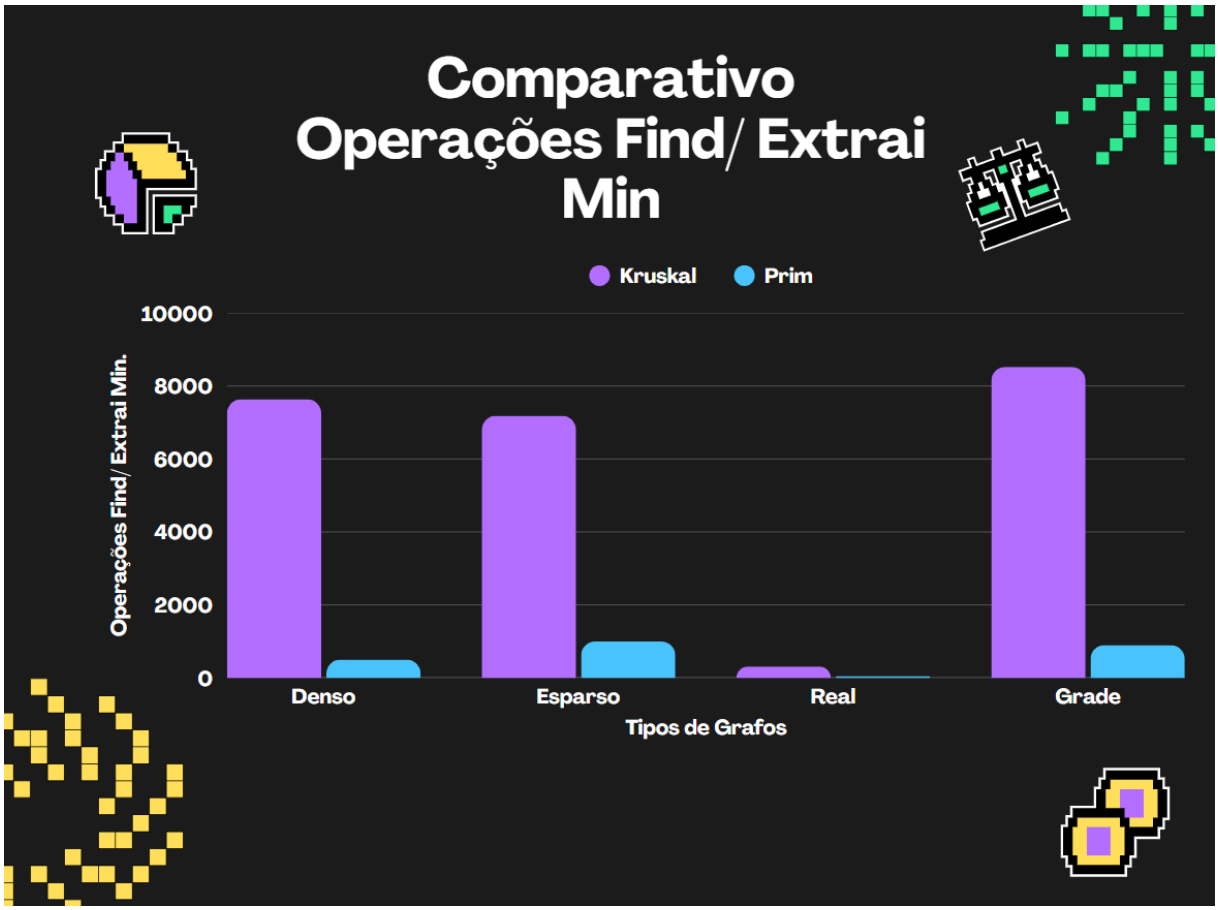

Figura 3: Contagem de operações `find` / `extraiMin` por iteração.

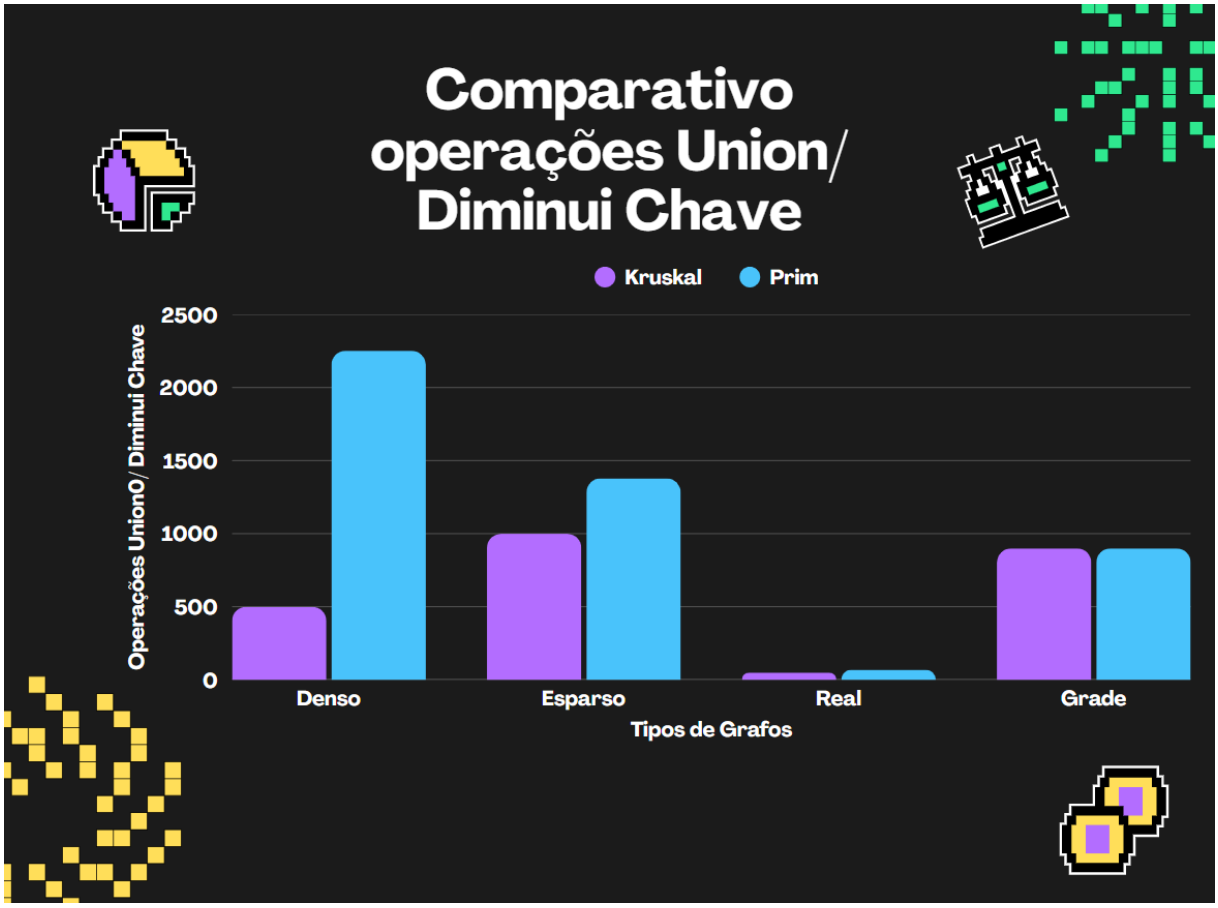

Figura 4: Contagem de operações `union` / `diminuiChave` por iteração.

6 Discussão

Os resultados mostram, de forma geral, que não existe um "melhor algoritmo" absoluto, mas sim aquele mais adequado ao tipo de grafo e aos recursos disponíveis:

- **Grafos esparsos:** Kruskal foi visivelmente mais rápido nos casos em que o número de arestas era pequeno em relação aos vértices. Como só precisa ordenar e unir componentes poucas vezes, ele finaliza quase instantaneamente, enquanto Prim gasta mais tempo gerenciando a fila de prioridades.
- **Grafos densos:** Quando há muitas arestas, Prim mostrou-se competitivo, pois amplia a árvore passo a passo sem precisar ordenar todas as conexões. Nesse cenário, o custo extra de ordenar centenas de milhares de arestas (como faz Kruskal) pesa mais do que a manutenção do heap de Prim.
- **Malha 2D:** Em grafos de tamanho moderado, como a grade 30×30, ambos ficaram quase empatados. Isso indica que, para redes de complexidade intermediária, qualquer dos dois atende bem.

- **Grafo real de pequena dimensão:** Com apenas dezenas de vértices e poucas arestas, a diferença entre Kruskal e Prim se torna irrelevante: ambos concluem em frações de milissegundo.
- **Uso de memória:** Prim usa um pouco mais de memória para manter a fila de prioridades e vetores auxiliares, enquanto Kruskal exige apenas espaço para a lista de arestas e a estrutura de união. Em situações de hardware limitado, isso pode influenciar a escolha.
- **Recomendações práticas:**
  - Para *redes esparsas* (poucas conexões), prefira Kruskal.
  - Para *redes densas* (muitas conexões), use Prim.

Em suma, a decisão entre Kruskal ou Prim deve considerar o tipo de grafo em que se trabalha e as limitações do ambiente, em vez de se basear apenas no desempenho médio teórico.

7 Conclusão

Este trabalho mostrou que não há uma solução única para todas as situações: a escolha entre Kruskal e Prim depende principalmente da estrutura do grafo e dos recursos disponíveis. Em grafos com poucas conexões (esparsos), Kruskal costuma ser mais veloz e simples de implementar, enquanto em grafos muito conectados (densos), Prim se beneficia de não precisar ordenar todas as arestas e, assim, escala melhor. Para redes de tamanho intermediário ou reais de pequena dimensão, ambos os métodos apresentam desempenho semelhante.

Além do aspecto de tempo de execução, deve-se levar em conta o uso de memória e o formato dos dados de entrada. Em sistemas com restrição de espaço, a implementação de Kruskal e sua leve estrutura de união-encontro podem ser vantajosas; já em ambientes onde a geração de grafos densos é frequente, Prim com heap ou com abordagem $O(n^2)$ pode ser mais apropriado.

Em suma, Kruskal e Prim continuam sendo ferramentas robustas e complementares para resolver o problema da árvore geradora mínima; a seleção entre eles deve refletir as características do problema prático em questão.

Referências

[1] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. 3. ed. Elsevier, 2012.

[2] BHARGAVA, A. Y.; FAZAR, E. *Entendendo Algoritmos*. Novatec, 2018.

[3] JESULIANA, N. Ulysses. *Slides de Grafos*. UFSJ, 2025.