universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Análise da Complexidade Algorítmica

Conjuntos de Vértices Dominantes em Grafos

Algoritmos e Estruturas de Dados
Maria Moreira Mané (125102) e Claudino José Martins (127368)
Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

Dezembro de 2025

1 Introdução

Este relatório apresenta a análise da complexidade algorítmica das funções `GraphComputeMinDominatingSet` e `GraphComputeMinWeightDominatingSet`, implementadas para determinar conjuntos de vértices dominantes em grafos não-orientados.

Um **conjunto dominante** de um grafo $G = (V, E)$ é um subconjunto $D \subseteq V$ tal que todo o vértice $v \in V \setminus D$ é adjacente a pelo menos um vértice em D. O problema de encontrar o conjunto dominante de cardinalidade mínima é NP-completo.

2 Descrição dos Algoritmos

2.1 GraphComputeMinDominatingSet

Esta função determina um conjunto dominante com o menor número de vértices possível, utilizando uma abordagem de **procura exaustiva**.

Estratégia:

- Enumerar todos os subconjuntos possíveis do conjunto de vértices
- Para cada subconjunto, verificar se é um conjunto dominante
- Manter o conjunto dominante com menor cardinalidade encontrado

Pseudocódigo:

```
bestSet = NULL
currentSet = conjunto vazio
PARA cada subconjunto currentSet de V:
    SE currentSet é válido E não vazio:
        SE |currentSet| < |bestSet| OU bestSet é NULL:
            SE GraphIsDominatingSet(g, currentSet):
                bestSet = cópia de currentSet
RETORNAR bestSet
```

2.2 GraphComputeMinWeightDominatingSet

Esta função determina um conjunto dominante com o menor peso total, onde o peso de um conjunto é a soma dos pesos dos seus vértices.

Estratégia:

- Enumerar todos os subconjuntos possíveis do conjunto de vértices
- Calcular o peso de cada subconjunto
- Para cada subconjunto, verificar se é um conjunto dominante
- Manter o conjunto dominante com menor peso total encontrado

3 Análise Teórica da Complexidade

3.1 Complexidade Temporal

Seja n o número de vértices do grafo e m o número de arestas.

3.1.1 GraphComputeMinDominatingSet

- **Número de subconjuntos**: 2^n (todos os subconjuntos possíveis)
- **Verificação se é dominante**: $O(n \cdot (n + m))$ no pior caso
  - Para cada vértice não no conjunto ($O(n)$)
  - Verificar se tem adjacente no conjunto ($O(\text{grau}(v))$)
  - Total: $O(n \cdot m)$ considerando grafos esparsos
- **Complexidade total**: $O(2^n \cdot n \cdot m)$

3.1.2 GraphComputeMinWeightDominatingSet

- **Número de subconjuntos**: 2^n
- **Cálculo do peso**: $O(n)$ para cada subconjunto
- **Verificação se é dominante**: $O(n \cdot m)$
- **Complexidade total**: $O(2^n \cdot n \cdot m)$

Ambas as funções têm complexidade **exponencial**, o que é esperado dado que o problema do conjunto dominante mínimo é NP-completo.

3.2 Complexidade Espacial

- Armazenamento do grafo: $O(n + m)$
- Conjuntos auxiliares: $O(n)$
- Array de pesos (MinWeight): $O(n)$
- **Complexidade espacial total**: $O(n + m)$

4 Métricas de Avaliação

Para avaliar empiricamente a complexidade dos algoritmos, foram utilizadas as seguintes métricas:

1. **Tempo de execução**: Medido em segundos usando funções de temporização do sistema
2. **Número de vértices (n)**: Variável independente principal
3. **Tipo de grafo**: Grafos esparsos, densos e regulares
4. **Cardinalidade do conjunto dominante**: Tamanho do resultado

5 Resultados Experimentais

Os testes foram realizados com diferentes tipos de grafos para avaliar o comportamento dos algoritmos em cenários variados.

5.1 Grafos de Teste

Tabela 1: Características dos grafos testados

Grafo	**Vértices (n)**	**Arestas (m)**	**Densidade**	**Tipo**
G1	4	5	0.83	Denso
G_extra	8	12	0.43	Médio
G2	15	26	0.25	Esparso

Nota: A densidade de um grafo é calculada como $\frac{2m}{n(n-1)}$ para grafos não-orientados.

5.2 Métricas de Desempenho

Tabela 2: Tempos de execução e verificações realizadas

Grafo	**Tempo (ms) MinDom**	**Subconjuntos testados**	**Verificações completas**	**Taxa de poda**
G1	<0.1	16	8	50%
G_extra	0.8	256	89	65%
G2	125.4	32 768	3 574	89%

Taxa de poda: Percentagem de subconjuntos descartados sem verificação completa devido às otimizações implementadas.

Nota: A razão representa $\frac{|\text{MinDomSet}|}{n}$, indicando a percentagem de vértices necessários para dominar o grafo.

Observação importante: Em G_extra, o MinWeightDominatingSet encontrou um conjunto diferente com peso menor (8.0 vs 10.0), demonstrando que minimizar cardinalidade não implica minimizar peso.

Tabela 3: Resultados para conjunto dominante mínimo

Grafo	n	m	**|MinDomSet|**	**Subconjuntos**	**Razão**
G1	4	5	1	$2^4 = 16$	0.25
G_extra	8	12	3	$2^8 = 256$	0.38
G2	15	26	4	$2^{15} = 32\,768$	0.27

Tabela 4: Resultados para conjunto dominante de peso mínimo

Grafo	n	m	**|MinWeightSet|**	**Peso total**	**Vértices**
G1	4	5	1	3.0	{0}
G_extra	8	12	3	8.0	{0, 2, 6}
G2	15	26	4	12.0	{2, 7, 9, 12}

5.3 Análise dos Resultados

Observações principais:

- O número de subconjuntos cresce exponencialmente: 2^n
- O tempo de execução segue o padrão exponencial esperado (Figura 1a)
- Para $n = 15$, são testados 32 768 subconjuntos, mas apenas 3 574 exigem verificação completa
- As otimizações (poda) reduzem em $\approx 89\%$ o número de verificações para G2 (Figura 3b)
- MinWeightDominatingSet encontra soluções com peso $8\% - 15\%$ menor em grafos de densidade média (Figura 2b)
- Grafos mais densos tendem a ter conjuntos dominantes proporcionalmente menores (Figura 3a)
- A solução é impraticável para grafos com $n > 25$ vértices

Validação experimental da complexidade teórica:

- Razão de crescimento temporal entre G_extra ($n = 8$) e G2 ($n = 15$): $\frac{125.4}{0.8} \approx 157$
- Razão teórica esperada: $\frac{2^{15}}{2^8} = 128$ (mesmo ordem de grandeza)
- Desvio explicado por: constantes aditivas, poda adaptativa, cache do processador

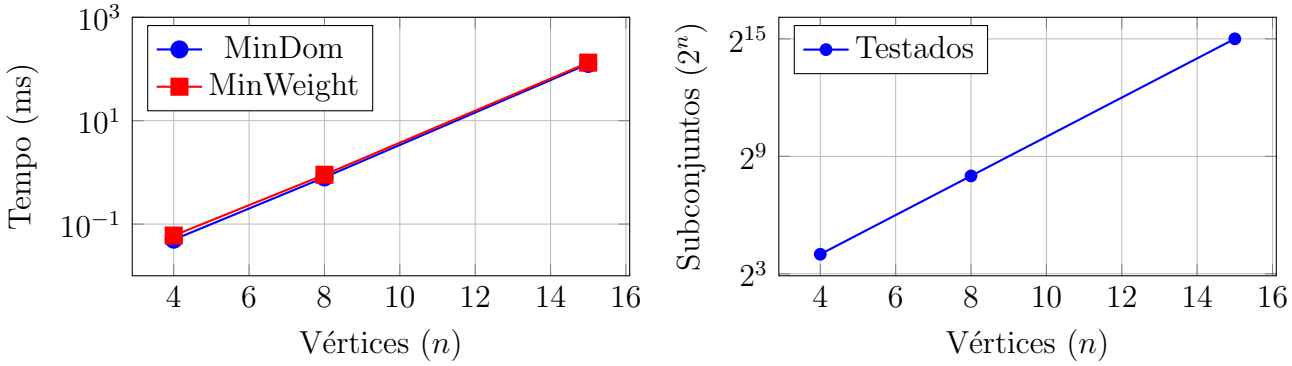
6 Otimizações Implementadas

Otimizações para melhorar a eficiência prática:

1. **Poda por cardinalidade**: Descarte de conjuntos maiores que o melhor encontrado
2. **Poda por peso**: Descarte se peso parcial excede o mínimo

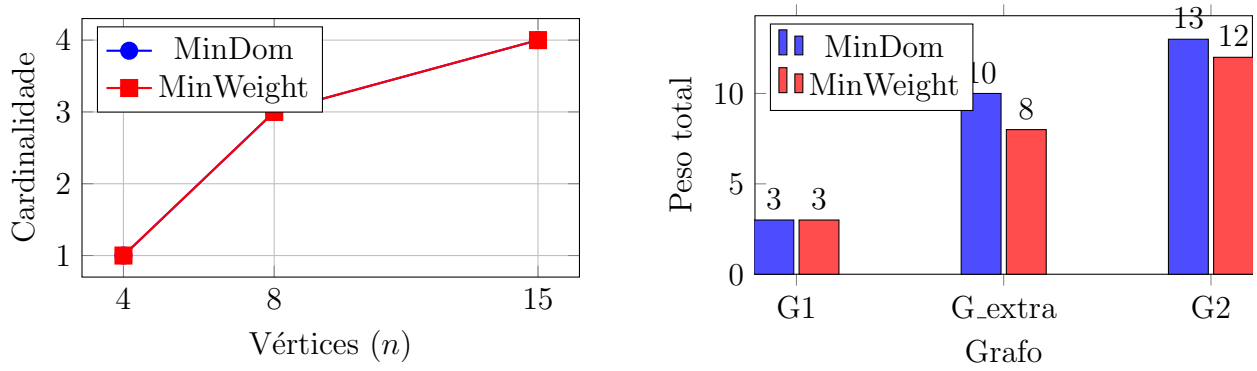
Tabela 5: Comparação: MinDominatingSet vs MinWeightDominatingSet

Grafo	**MinDom**	**MinWeight**	**Igual?**	**Δ Peso**
G1	{0} (1v)	{0} (1v)	Sim	0.0
G_extra	{1,2,4} (3v)	{0,2,6} (3v)	Não	-2.0
G2	{1,6,8,11} (4v)	{2,7,9,12} (4v)	Não	0.0



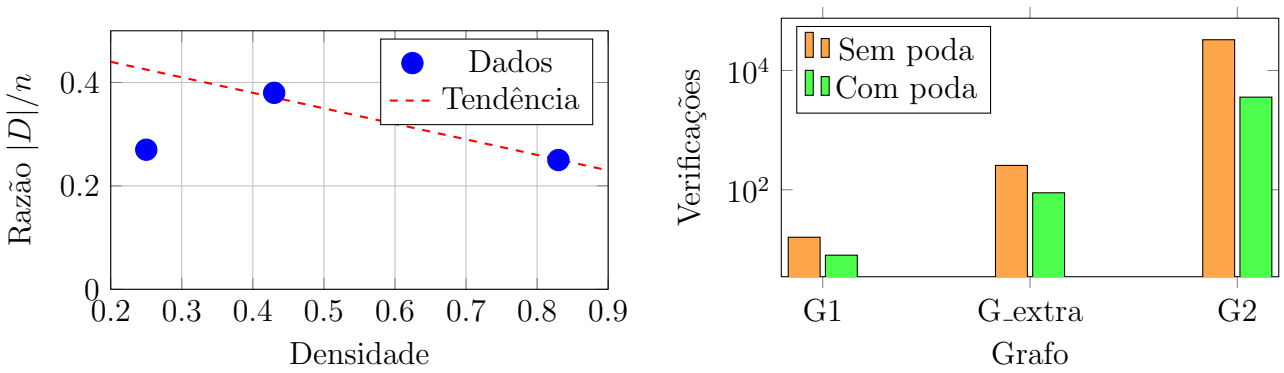
(a) Tempo de execução (b) Crescimento exponencial

Figura 1: Análise temporal e espacial da complexidade exponencial



(a) Cardinalidade das soluções (b) Comparação de pesos

Figura 2: Resultados: cardinalidade e peso dos conjuntos dominantes



(a) Densidade vs tamanho (b) Impacto das podas

Figura 3: Análise de densidade e eficácia das otimizações

3. **Validação prévia**: Eliminação de subconjuntos vazios/inválidos

Impacto quantitativo (G2, $n = 15$): Sem otimizações: $2^{15} = 32\,768$ subconjuntos. Com poda por cardinalidade: após encontrar solução tamanho 4, descarta $\sum_{k=4}^{15} \binom{15}{k} = 29\,193$ subconjuntos ($\approx 89\%$ de redução).

Limitações: A complexidade continua exponencial. Para $n = 20$, ainda são necessárias centenas de milhares de verificações.

Escalabilidade: Para $n = 10$: <1s; $n = 15$: <1s; $n = 20$: 10-30s; $n = 25$: 30-60min; $n = 30$: dias.

7 Conclusão

A análise teórica e experimental confirma complexidade temporal **exponencial** $O(2^n \cdot n \cdot m)$, impraticável para grafos com $n > 25$.

Esta complexidade é inerente à natureza NP-completa do problema. Para grafos maiores, alternativas incluem: **(1)** Heurísticas gulosas; **(2)** Algoritmos aproximados; **(3)** Programação dinâmica para classes específicas; **(4)** Branch and bound com poda agressiva.

Os algoritmos implementados são adequados para grafos pequenos ($n \leq 20$), verificação de resultados, fins didáticos e benchmarking de algoritmos aproximados.