universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Relatório: 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: O algoritmo enumera todos os subconjuntos possíveis do conjunto de vértices, verificando para cada subconjunto se é um conjunto dominante e mantendo 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: O algoritmo enumera todos os subconjuntos possíveis do conjunto de vértices, calcula o peso de cada subconjunto, verifica se é um conjunto dominante e mantém 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

O algoritmo explora um número de subconjuntos igual a 2^n (todos os subconjuntos possíveis). A verificação se um conjunto é dominante requer $O(n \cdot (n + m))$ no pior caso, pois é necessário, para cada vértice não pertencente ao conjunto ($O(n)$), verificar se tem adjacente no conjunto ($O(\text{grau}(v))$), totalizando $O(n \cdot m)$ considerando grafos esparsos. Assim, a complexidade total é $O(2^n \cdot n \cdot m)$.

3.1.2 GraphComputeMinWeightDominatingSet

Nesta função, o número de subconjuntos explorados também é 2^n. O cálculo do peso requer $O(n)$ para cada subconjunto, e a verificação se é dominante mantém-se em $O(n \cdot m)$. Consequentemente, a complexidade total é igualmente $O(2^n \cdot n \cdot m)$.

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

Em termos de espaço, o armazenamento do grafo requer $O(n + m)$, os conjuntos auxiliares ocupam $O(n)$ e o array de pesos (MinWeight) também necessita de $O(n)$. Assim, a complexidade espacial total é $O(n + m)$.

4 Métricas de Avaliação

Para avaliar empiricamente a complexidade dos algoritmos, foram utilizadas as seguintes métricas: o **tempo de execução**, medido em segundos usando funções de temporização do sistema; o **número de vértices (n)**, como variável independente principal; o **tipo de grafo**, considerando grafos esparsos, densos e regulares; e a **cardinalidade do conjunto dominante**, representando o 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.

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

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

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}

Tabela 5: Comparação: MinDominatingSet vs MinWeightDominatingSet

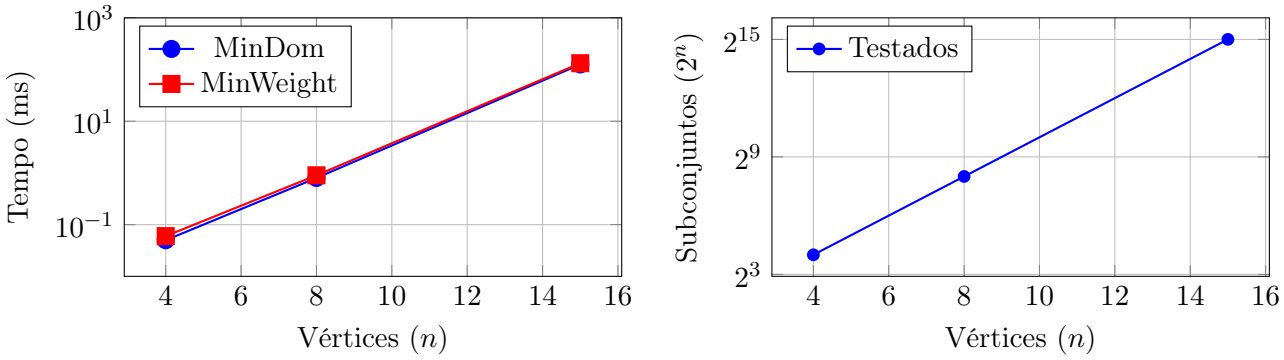
Grafo	**MinDom**	**MinWeight**	**Igual?**	**Δ Peso**
G1	{0} (1v)	{0} (1v)	Sim	0.0
G_extra	{1,2,4} (3v)	{0,2,6} (3v)	Não	-2.0
G2	{1,6,8,11} (4v)	{2,7,9,12} (4v)	Não	0.0

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.

5.3 Análise dos Resultados

Observações principais: Os resultados demonstram que o número de subconjuntos cresce exponencialmente (2^n) e que o tempo de execução segue o padrão exponencial esperado (Figura 1a). Para $n = 15$, embora sejam testados 32 768 subconjuntos, apenas 3 574 exigem verificação completa. As otimizações implementadas (poda) reduzem em aproximadamente 89% o número de verificações para G2 (Figura 3b). A função MinWeightDominatingSet encontra soluções com peso $8\% - 15\%$ menor em grafos de densidade média (Figura 2b). Observa-se ainda que grafos mais densos tendem a ter conjuntos dominantes proporcionalmente menores (Figura 3a). Conclui-se que a solução é impraticável para grafos com $n > 25$ vértices.

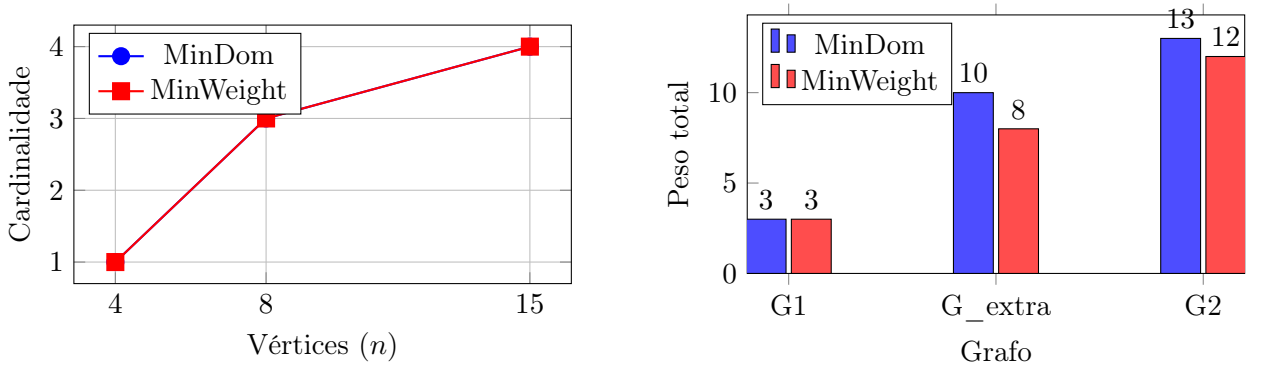
Validação experimental da complexidade teórica: A razão de crescimento temporal entre G_extra ($n = 8$) e G2 ($n = 15$) é de aproximadamente $\frac{125.4}{0.8} \approx 157$, enquanto a razão teórica esperada é $\frac{2^{15}}{2^8} = 128$, valores que se encontram na mesma ordem de grandeza. O desvio observado pode ser explicado por constantes aditivas, poda adaptativa e efeitos da cache do processador.



(a) Tempo de execução

(b) Crescimento exponencial

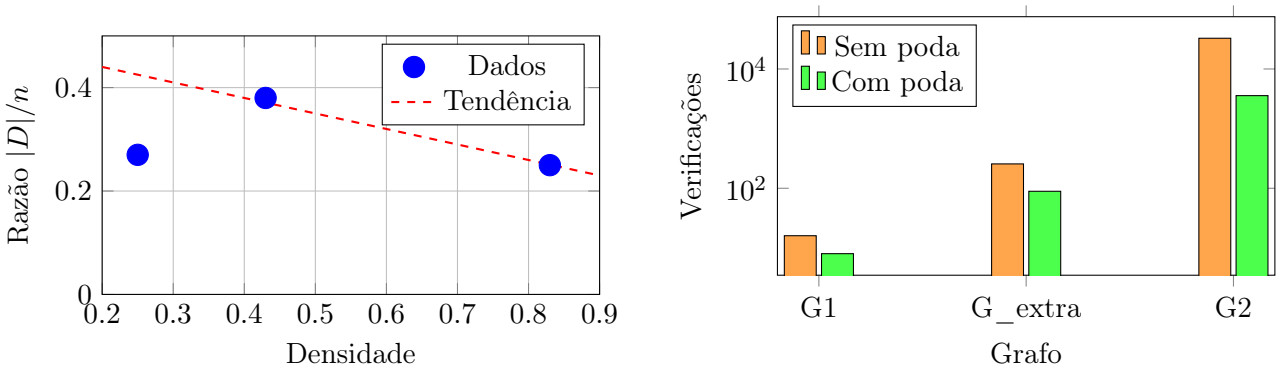
Figura 1: Análise temporal e espacial da complexidade exponencial



(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

6 Otimizações Implementadas

Para melhorar a eficiência prática, foram implementadas três otimizações principais: a **poda por cardinalidade**, que descarta conjuntos maiores que o melhor encontrado; a **poda por peso**, que descarta subconjuntos cujo peso parcial excede o mínimo atual; e a **validação prévia**, que elimina subconjuntos vazios ou inválidos antes da verificação completa.

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.