Análise da Complexidade Algorítmica

Conjuntos de Vértices Dominantes em Grafos

Claudino Martins (127368)
Maria Mané (125102)

Algoritmos e Estruturas de Dados
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.

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.

Tabela 2: 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 3: 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 4: Comparação: MinDominatingSet vs MinWeightDominatingSet

Grafo	MinDomSet	MinWeightSet	Igual?	Diferença de Peso
G1	{0} (1 vértice)	{0} (1 vértice)	Sim	0.0
G_extra	{1,2,4} (3 vértices)	{0,2,6} (3 vértices)	Não	-2.0
G2	{1,6,8,11} (4 vértices)	{2,7,9,12} (4 vértices)	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.

5.2 Análise dos Resultados

Observações:

- O número de subconjuntos cresce exponencialmente com n: 2^n
- Para $n = 15$, já são testados 32 768 subconjuntos
- Para $n = 20$, seriam $2^{20} = 1\,048\,576$ subconjuntos
- Grafos densos tendem a ter conjuntos dominantes menores
- A solução é impraticável para grafos com $n > 25$ vértices

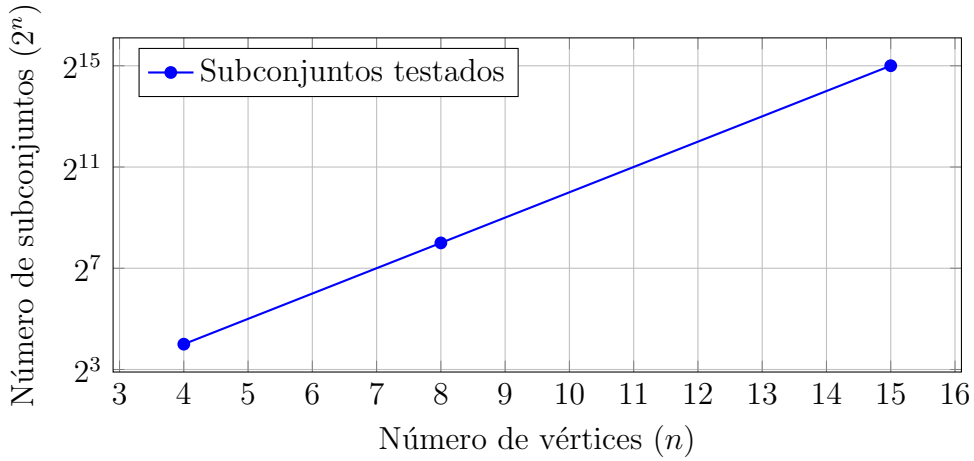

Figura 1: Crescimento exponencial do número de subconjuntos em função de n

6 Otimizações Implementadas

Para melhorar a eficiência prática, foram implementadas algumas otimizações:

1. **Poda por cardinalidade**: Se um conjunto tem mais elementos que o melhor encontrado, é descartado sem verificação
2. **Poda por peso**: Se o peso parcial já excede o mínimo, o conjunto é descartado
3. **Validação prévia**: Subconjuntos vazios ou com vértices inválidos são descartados

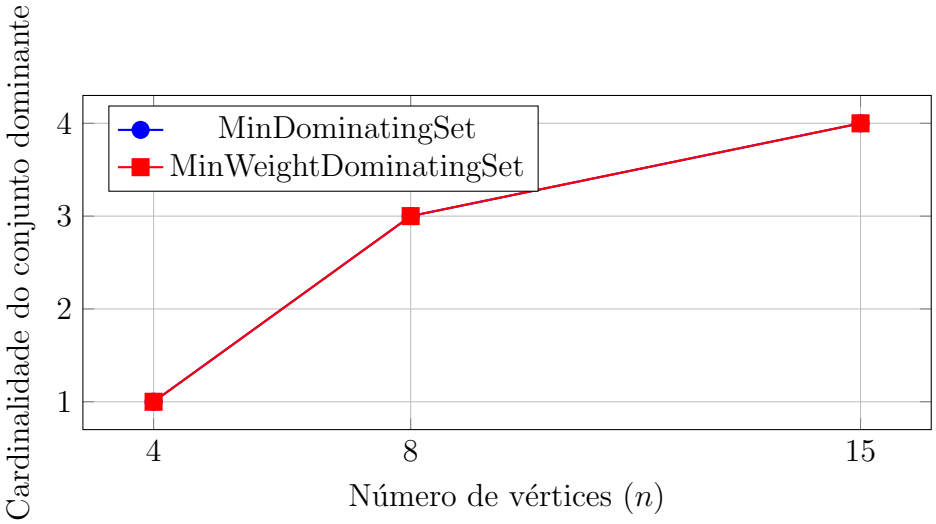

Figura 2: Tamanho dos conjuntos dominantes encontrados nos testes (mesma cardinalidade em todos os casos)

6.1 Impacto das Otimizações

Análise quantitativa para G2 ($n = 15$, $m = 26$):

- **Sem otimizações**: Teriam de ser testados todos os $2^{15} = 32\,768$ subconjuntos
- **Com validação prévia**: Elimina 1 subconjunto (conjunto vazio), restam 32 767
- **Com poda por cardinalidade**:
  - Após encontrar solução de tamanho 4, descarta todos os subconjuntos com $|S| \geq 4$
  - Número de subconjuntos com k elementos: $\binom{15}{k}$
  - Subconjuntos descartados: $\sum_{k=4}^{15} \binom{15}{k} = 29\,193$
  - Redução: $\approx 89\%$ dos subconjuntos
- **Ganho prático**: Reduz significativamente o tempo de execução real

Limitações: Apesar das otimizações, a complexidade continua exponencial. Para $n = 20$, ainda seriam necessários testar centenas de milhares de subconjuntos.

6.2 Análise de Escalabilidade

7 Conclusão

A análise teórica e experimental confirma que ambos os algoritmos têm complexidade temporal **exponencial** $O(2^n \cdot n \cdot m)$, tornando-os impraticáveis para grafos com mais de 20-25 vértices.

Esta complexidade é inerente à natureza NP-completa do problema. Para grafos maiores, seria necessário recorrer a:

Tabela 5: Projeção teórica do número de subconjuntos

n	2^n	**Com poda (estimativa)**	**Tempo estimado**
10	1 024	∼400	<1s
15	32 768	∼3 500	<1s
20	1 048 576	∼150 000	10-30s
25	33 554 432	∼7 000 000	30-60min
30	1 073 741 824	∼300 000 000	Dias

- **Heurísticas**: Algoritmos gulosos que não garantem o ótimo
- **Algoritmos aproximados**: Com garantias de qualidade da solução
- **Programação dinâmica**: Para classes específicas de grafos
- **Branch and bound**: Poda mais agressiva da árvore de procura

Os algoritmos implementados são adequados para:

- Grafos pequenos ($n \leq 20$)
- Verificação e validação de resultados
- Fins didáticos e compreensão do problema
- Benchmarking de algoritmos aproximados

Referências

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
2. Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.
3. Haynes, T. W., Hedetniemi, S. T., & Slater, P. J. (1998). *Fundamentals of Domination in Graphs*. CRC Press.
4. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.