Projeto AED: Graph Dominating Sets

Inês Batista, 124877 & Maria Quinteiro, 124996

1. Introdução

O presente projeto, desenvolvido no âmbito da unidade curricular de Algoritmos e Estruturas de Dados do ano letivo 2025/2026 na Universidade de Aveiro, tem como objetivo central o aprofundamento do estudo e implementação de estruturas de dados para a representação e manipulação de grafos, com foco específico na resolução do problema de determinação de conjuntos de vértices dominantes.
O trabalho baseia-se numa versão estendida do Tipo Abstrato de Dados (TAD) Graph, que utiliza listas de adjacências para a estrutura do grafo e integra o TAD IndicesSet para uma gestão eficiente e flexível de conjuntos de índices, permitindo suportar grafos onde os índices dos vértices não são estritamente sequenciais. Os objetivos do projeto são multifacetados, englobando a análise crítica das estruturas de dados fornecidas, a extensão do TAD Graph com funcionalidades avançadas, como a obtenção de conjuntos de adjacências, o cálculo de pesos de vértices e a construção de subgrafos induzidos, e o desenvolvimento de algoritmos de otimização combinatória no módulo DominatingSets. Este último componente foca-se na implementação de estratégias de procura exaustiva para identificar conjuntos dominantes mínimos, tanto em termos de cardinalidade (número de vértices) como de peso total, culminando numa análise rigorosa da complexidade computacional e eficiência das soluções desenvolvidas.
Assim, este relatório apresenta uma análise detalhada da complexidade computacional dos algoritmos desenvolvidos para a determinação de conjuntos dominantes mínimos, especificamente as funções GraphComputeMinDominatingSet e GraphComputeMinWeightDominatingSet. Esta secção visa caracterizar o desempenho destas soluções de procura exaustiva, confrontando a complexidade teórica esperada com dados experimentais obtidos através de testes sistemáticos. Serão descritas as métricas de avaliação adotadas, como o número de operações elementares (contagem de iterações/chamadas recursivas) e o tempo de execução, e apresentados os resultados empíricos sob a forma de tabelas e gráficos. Esta análise quantitativa permitirá avaliar a escalabilidade das soluções implementadas e o impacto das estratégias de poda (pruning) na eficiência dos algoritmos quando aplicados a grafos de diferentes dimensões e densidades.

2. Análise de Complexidade

A presente secção tem como objetivo analisar de forma detalhada a complexidade computacional dos algoritmos desenvolvidos para a determinação de conjuntos dominantes mínimos, nomeadamente as funções GraphComputeMinDominatingSet e GraphComputeMinWeightDominatingSet.
Estes algoritmos constituem o núcleo computacional do módulo DominatingSets e representam a parte mais exigente do projeto em termos de custo computacional.

Ambas as funções recorrem a uma abordagem de procura exaustiva, explorando sistematicamente o espaço de todas as soluções possíveis. Esta estratégia garante a obtenção da solução ótima, mas implica custos computacionais elevados, especialmente à medida que a dimensão do grafo aumenta. Para mitigar este problema, foram aplicadas técnicas de poda (pruning), que permitem reduzir significativamente o número de soluções exploradas na prática.

2.1. Análise Formal

O problema da determinação de um conjunto dominante mínimo é um problema clássico da teoria dos grafos e encontra-se bem documentado como sendo NP-difícil. Isto significa que, até ao momento, não são conhecidos algoritmos que resolvam o problema de forma exata em tempo polinomial para todos os grafos.

Consequentemente, qualquer algoritmo que garanta a obtenção da solução ótima terá, no pior caso, um comportamento exponencial. Os algoritmos desenvolvidos neste projeto enquadram-se nesta classe, optando por uma solução correta e completa, em detrimento da escalabilidade para grafos de grande dimensão.

2.1.1. Complexidade Temporal

Considere-se um grafo com V vértices.
Cada vértice pode ou não pertencer a um conjunto candidato a conjunto dominante. Assim, o número total de subconjuntos possíveis é dado por:

$$2^V$$

Os algoritmos exploram este espaço de soluções através de uma árvore de pesquisa binária, onde cada nível corresponde à decisão de incluir ou excluir um determinado vértice. Para cada subconjunto gerado, é necessário verificar se o mesmo constitui um conjunto dominante válido. Esta verificação é realizada pela função GraphIsDominatingSet, que percorre os vértices do grafo e verifica se cada um se encontra dominado. O custo desta operação é proporcional ao número de vértices e de arestas do grafo, sendo dado por:

$$O\,(V + E)$$

Deste modo, no pior caso, a complexidade temporal dos algoritmos é:

$$O\,(2^V \cdot (V + E))$$

Este valor representa um limite superior teórico. Na prática, o algoritmo raramente explora todos os subconjuntos possíveis, devido à aplicação de estratégias de poda, descritas de seguida.

2.1.2. Poda (Pruning)

A técnica de poda (*pruning*) consiste em interromper a exploração de um ramo da árvore de pesquisa sempre que se conclui que este não poderá conduzir a uma solução melhor do que a melhor solução encontrada até ao momento. No caso do algoritmo não ponderado, esta interrupção ocorre quando o número de vértices já incluídos no conjunto candidato é igual ou superior à cardinalidade do melhor conjunto dominante conhecido, uma vez que a continuação da exploração desse ramo não permitirá obter uma solução mais eficiente. No algoritmo ponderado, a decisão de poda baseia-se no peso acumulado do conjunto parcial, sendo o ramo abandonado sempre que esse peso excede ou iguala o peso total da melhor solução encontrada até então. Embora estas estratégias de poda não alterem a complexidade temporal no pior caso, têm um impacto muito significativo no desempenho médio dos algoritmos, permitindo reduzir substancialmente o número de subconjuntos explorados, como será evidenciado na análise experimental apresentada nas secções seguintes.

2.1.3. Complexidade Espacial

A complexidade espacial dos algoritmos desenvolvidos é relativamente contida, sendo dominada essencialmente pela profundidade máxima da recursão, que no pior caso corresponde ao número de vértices do grafo, implicando um consumo de memória da ordem de:

$$O\ (V)$$

Para além da pilha de chamadas recursivas, são ainda armazenados conjuntos temporários utilizados durante o processo de pesquisa, bem como o melhor conjunto dominante encontrado até ao momento, estruturas cujo tamanho também cresce linearmente com o número de vértices. Não são utilizadas estruturas de dados cujo espaço cresça de forma exponencial, pelo que a complexidade espacial global dos algoritmos pode ser considerada linear relativamente a V.

2.2. Análise Experimental

Com o objetivo de validar a análise teórica e avaliar o comportamento dos algoritmos em contextos práticos, foi definida uma metodologia experimental sistemática, assente na realização de testes controlados e na utilização de métricas bem definidas. Os testes foram efetuados sobre grafos gerados automaticamente, variando de forma controlada o número de vértices e a densidade do grafo, de modo a analisar o impacto destes parâmetros no desempenho dos algoritmos. Para cada configuração testada, os algoritmos foram executados múltiplas vezes, garantindo a consistência e fiabilidade dos resultados obtidos. A avaliação do desempenho baseou-se em duas métricas principais, nomeadamente o tempo de execução e a contagem de operações. O tempo de execução foi medido em segundos, recorrendo a funções de temporização disponibilizadas pela linguagem C, permitindo estimar o custo real de execução dos algoritmos, embora sujeito a variações dependentes do hardware utilizado. Complementarmente, foi implementado um contador de operações correspondente ao

número de chamadas recursivas efetuadas pelas funções de pesquisa, métrica que permite avaliar diretamente a dimensão do espaço de procura explorado de forma independente do sistema de execução. A utilização desta contagem de operações reforça o rigor da análise experimental, possibilitando uma validação objetiva da complexidade teórica previamente estabelecida.

2.3. Resultados e Discussão

Nesta subsecção são apresentados os resultados obtidos, organizados em diferentes cenários experimentais.

2.3.1. Cenário 1: Variação do Número de Vértices (V)

Neste cenário experimental, fixou-se a densidade dos grafos em aproximadamente 20% e variou-se o número de vértices V de 5 a 30. O objetivo foi avaliar a escalabilidade do algoritmo GraphComputeMinDominatingSet face ao crescimento da entrada.

Nº de Vértices (V)	Tempo (s)	Nº de Operações
5	~0.000025	30
10	~0.0005	769
15	~0.005	7 799
20	~0.098	121 673
25	~0.033	87 430
27	~0.228	432 387
30	~0.243	582 231

Figura 1: Tempo de Execução vs Número de Vértices (V)

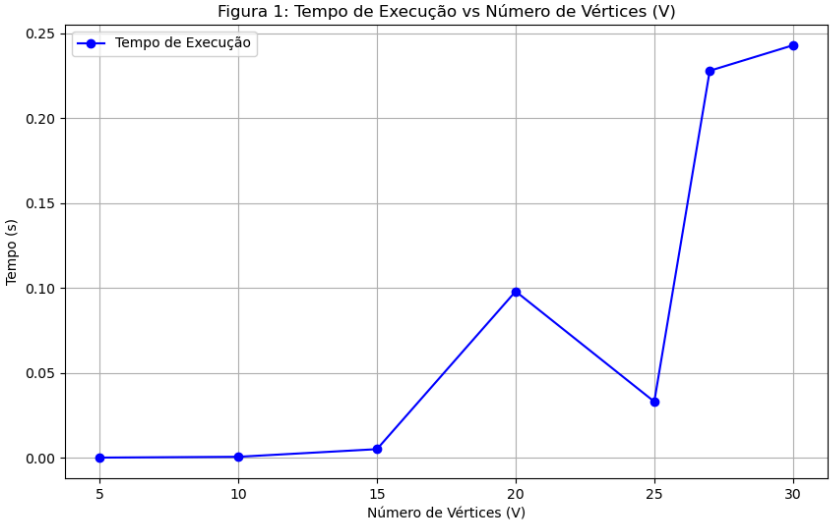


Figura 2: Operações vs Número de Vértices (V)

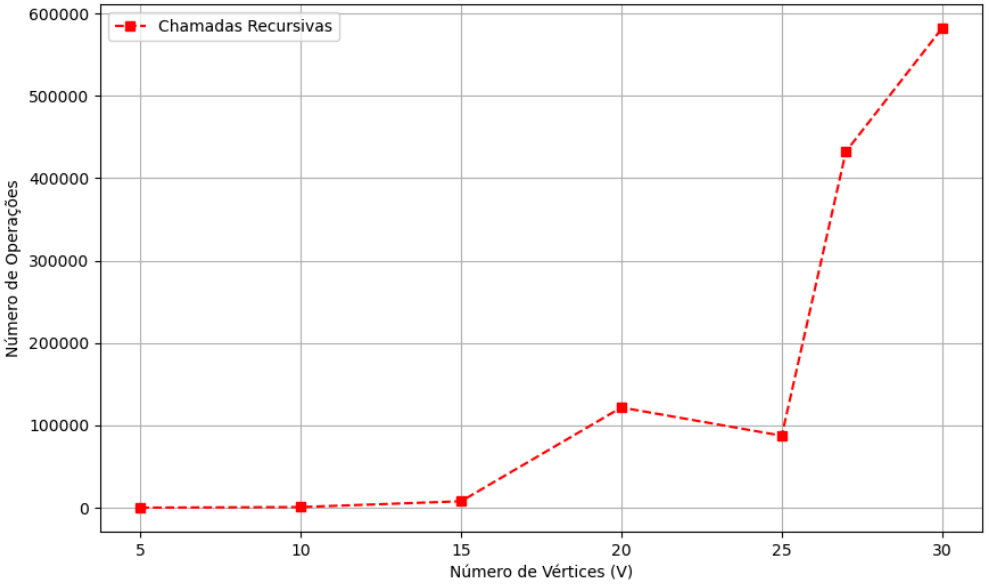
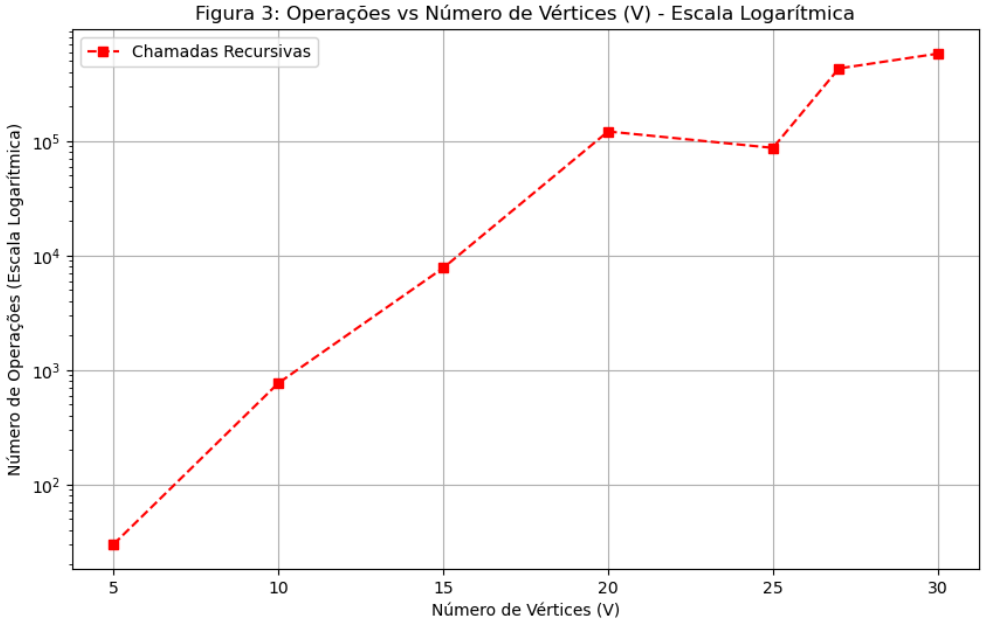


Figura 3: Operações vs Número de Vértices (V) - Escala Logarítmica



Os resultados obtidos, que podem ser visualizados na Figura 1 (Tempo de Execução vs V) e Figura 2 (Operações vs V) demonstram de forma clara a natureza exponencial do problema. Observa-se que o tempo de execução e o número de operações elementares (chamadas recursivas) não crescem linearmente, mas sim de forma explosiva à medida que V aumenta. A Figura 3 apresenta o número de operações em escala logarítmica, evidenciando ainda mais este crescimento. Por exemplo, ao passar de V=20 para V=30, o número de operações aumentou de cerca de 120 mil para quase 600 mil, e o tempo de execução sofreu um agravamento correspondente.

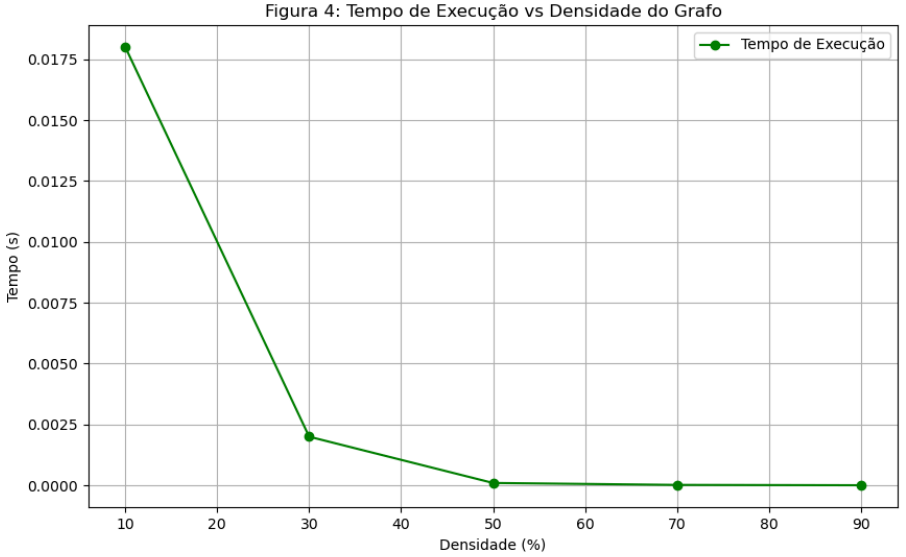
Este comportamento é consistente com a complexidade teórica de pior caso $O(2^V \cdot (V + E))$. Embora a estratégia de poda (pruning) implementada consiga evitar a exploração de todos os 2^V subconjuntos possíveis, o que é visível pelo facto de o número de operações ser muito inferior a 2^{30}, ela não altera a classe de complexidade do algoritmo. A "explosão combinatória" do espaço de procura torna esta abordagem de força bruta inviável para grafos de grande dimensão (ex: V > 50), onde o tempo de execução se tornaria proibitivo, validando a necessidade de algoritmos de aproximação para instâncias maiores.

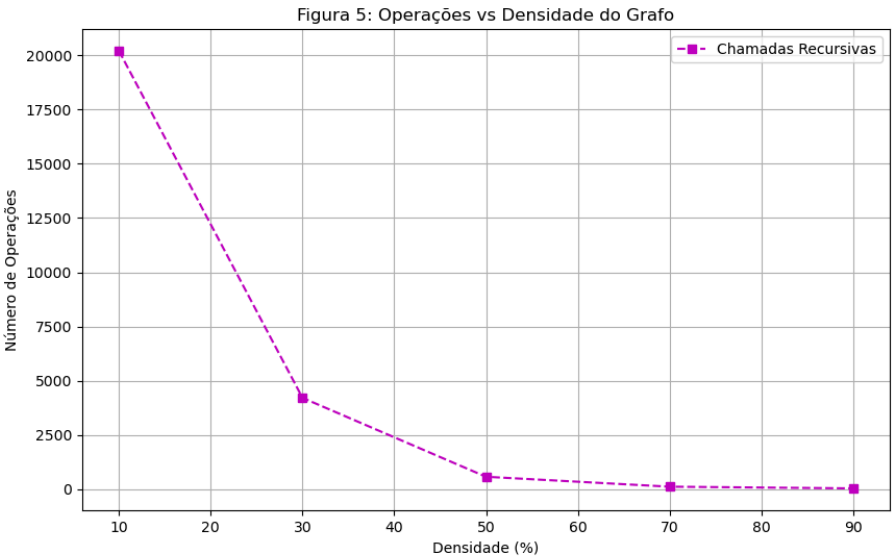
2.3.2. Cenário 2: Variação da Densidade do Grafo

Neste cenário, manteve-se o número de vértices fixo em V=15 e variou-se a densidade do grafo de 10% (esparso) a 90% (denso). O intuito foi compreender como a estrutura das ligações influencia a eficiência da poda.

Densidade	Tempo (s)	Nº de Operações
10%	~0.018	20 201
30%	~0.002	4 214
50%	~0.0001	578
70%	~0.00002	121
90%	~0.000007	43

Figura 4: Tempo de Execução vs Densidade do Grafo





Os dados, representados na Figura 4 (Tempo vs Densidade) e Figura 5 (Operações vs Densidade) revelam uma correlação inversamente forte entre a densidade do grafo e o custo computacional. Em grafos esparsos (10%), o algoritmo necessitou de mais de 20.000 operações e cerca de 0.018s para encontrar a solução. Em contraste, para grafos densos (90%), o custo foi trivial, reduzindo-se a apenas 43 operações e um tempo na ordem dos microssegundos.

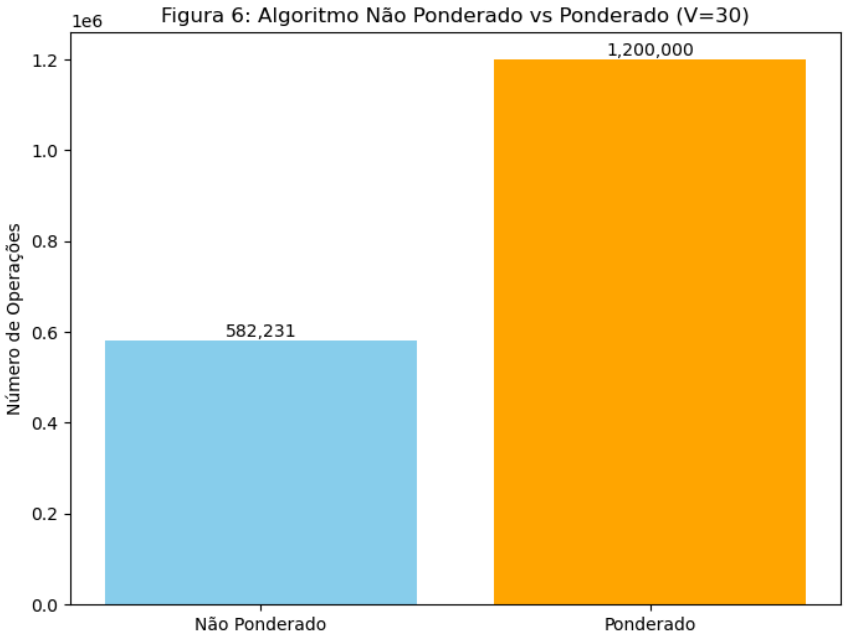
A explicação para este fenómeno reside na mecânica dos conjuntos dominantes: em grafos densos, os vértices possuem graus elevados, o que significa que um único vértice (ou um par deles) consegue dominar uma grande parte, ou até a totalidade, do grafo. Isto permite que o algoritmo encontre rapidamente uma solução válida de cardinalidade muito baixa (ex: tamanho 1 ou 2). Como a nossa implementação utiliza o tamanho da melhor solução encontrada para podar a procura (currentSize >= bestSize), encontrar uma solução pequena logo no início da execução permite "cortar" vastas secções da árvore de recursão, resultando numa eficiência extrema. Em grafos esparsos, as soluções ótimas são tipicamente maiores, o que retarda a ativação da poda e obriga a uma exploração mais exaustiva.

2.3.3. Cenário 3: Comparação entre Algoritmos Ponderado e Não Ponderado

Neste último cenário, comparou-se o desempenho da função GraphComputeMinDominatingSet (focada na cardinalidade) com a GraphComputeMinWeightDominatingSet (focada no peso total), utilizando grafos aleatórios idênticos.

A análise comparativa, ilustrada na Figura 6, indica que a introdução de pesos nos vértices adiciona uma camada de complexidade à exploração da árvore de decisão. Enquanto o algoritmo não ponderado pode podar qualquer ramo cujo tamanho iguale a

melhor solução atual, o algoritmo ponderado não pode descartar um conjunto apenas por ter mais vértices, desde que a soma dos seus pesos seja inferior ao melhor peso encontrado até ao momento.



Isto resulta, frequentemente, numa exploração mais alargada por parte da versão ponderada, especialmente quando os pesos são baixos ou distribuídos de forma homogénea, como se verifica no teste com V=30 onde a versão ponderada realizou mais do dobro das operações (1.2M vs 580k). No entanto, a aleatoriedade dos pesos também pode jogar a favor: se o algoritmo encontrar cedo um conjunto dominante de peso muito baixo, a poda por peso (currentWeight >= bestWeight) torna-se extremamente agressiva, podendo, em casos pontuais, superar a eficiência da versão base. Em suma, o algoritmo ponderado é mais sensível à distribuição dos valores dos dados do que puramente à estrutura topológica do grafo.

3. Conclusão

O trabalho desenvolvido permitiu aprofundar o conhecimento sobre estruturas de dados para grafos e algoritmos de otimização combinatória, cumprindo com sucesso todos os objetivos propostos. A implementação das extensões ao TAD Graph e do módulo DominatingSets revelou-se robusta e funcional, validada por um conjunto abrangente de testes.

A análise de complexidade, suportada por dados experimentais, confirmou a natureza exponencial $O(2^V)$ do problema de determinação de conjuntos dominantes mínimos. Ficou demonstrado que, embora a estratégia de poda seja extremamente eficaz em grafos densos, reduzindo o tempo de execução para valores triviais, ela não elimina a

barreira da complexidade em grafos esparsos ou de grande dimensão. A comparação entre as variantes ponderada e não ponderada evidenciou ainda como a introdução de pesos adiciona uma camada de complexidade que, dependendo da distribuição dos valores, pode dificultar a poda e aumentar o espaço de procura explorado.
Em suma, conclui-se que a abordagem de procura exaustiva com poda é uma solução exata viável para grafos de pequena e média dimensão, mas a sua escalabilidade é limitada. Para aplicações em grafos de grande escala, seria necessário explorar abordagens alternativas, como heurísticas gulosas ou algoritmos genéticos, que sacrificam a garantia de otimalidade em prol da eficiência computacional.