Conjunto Dominante

PAULO LUIZ M. SOUZA, Universidade Federal de Ouro Preto, Brazil

1 Descrição

O problema do conjunto dominante é um problema clássico da teoria dos grafos. Dado um grafo G = (V, E), onde V é o conjunto de vértices e E é o conjunto de arestas, um conjunto dominante é um subconjunto de vértices D V tal que todo vértice de V está ou em D ou é adjacente a pelo menos um vértice de D.

Exemplo: Considere o seguinte grafo com 5 vértices e 6 arestas:

- Vértices: V = A, B, C, D, E
- Aresta: E = (A, B), (A, C), (B, D), (C, D), (C, E), (D, E)

Para este grafo, um possível conjunto dominante é D = A, D, porque:

- O vértice A domina B e C,
- O vértice D domina B, C e E,
- Todos os vértices estão dominados por A ou D, satisfazendo a condição do conjunto dominante.

Uma solução para o problema do conjunto dominante é um subconjunto dominate D de G, tal que o número de vértices de D é o menor possível. Para o exemplo, D é uma solução.

2 Solução

Para resolver o problema proposto, será adotado o paradigma de projeto baseado em algoritmos gulosos. Como a função objetivo do Problema do Conjunto Dominante é encontrar o menor conjunto dominante em um grafo, ou seja, minimizar o tamanho do conjunto, ele se enquadra na categoria de problemas de otimização. Dada a natureza desse problema, a escolha do algoritmo guloso como estratégia de solução é apropriada, pois busca construir uma solução eficiente selecionando localmente, em cada etapa, a melhor opção disponível, na esperança de que essa abordagem conduza a um resultado globalmente ótimo. Além do algoritmo guloso, também foi implementada uma meta-heurística para possibilitar uma comparação dos resultados obtidos.

(1) **Inicialização:**
   - 1.1 Crie um conjunto vazio para armazenar o conjunto dominante.
   - 1.2 Marque todos os vértices como não visitados.

(2) **Iteração:**
   - 2.1 Enquanto houver vértices não visitados:
     - 2.1.1 **Seleção Gulosa:** Encontra o vértice com o **maior número de vizinhos não visitados**. Este é o critério guloso: a cada passo, escolhe-se o vértice que "cobre" mais vértices ainda não cobertos.
     - 2.1.2 **Adição ao Conjunto Dominante:** Adiciona o vértice selecionado ao conjunto dominante.
     - 2.1.3 **Marcação:** Marca o vértice selecionado e todos os seus vizinhos como visitados, pois agora estão cobertos pelo conjunto dominante.

(3) **Retorno:**
   - 3.1 Retorna o conjunto dominante construído.

Author's Contact Information: Paulo Luiz M. Souza, Universidade Federal de Ouro Preto, João Monlevade, Minas Gerais, Brazil.

Em resumo: A estratégia do algoritmo guloso é iterativamente selecionar o vértice que cobre o maior número de vértices não cobertos até que todos os vértices do grafo estejam cobertos, seja por estarem no conjunto dominante ou por serem vizinhos de um vértice no conjunto dominante.

Observação: A escolha gulosa em cada passo não garante a otimalidade global, ou seja, o conjunto dominante encontrado pode não ser o menor possível.

3 Geração das Instâncias

A estratégia de geração de instâncias utilizada no código GeradorDeInstancias.java é baseada na criação de grafos aleatórios com as seguintes características:

- Número de Vértices: O número de vértices do grafo é definido pelo parâmetro numVertices passado para a função gerarGrafoAleatorio.
- Probabilidade de Aresta: A probabilidade de existir uma aresta entre dois vértices distintos é definida pelo parâmetro probAresta. Para cada par de vértices, um número aleatório entre 0 e 1 é gerado. Se esse número for menor que probAresta, uma aresta é adicionada entre os vértices.
- Grafos Não Direcionados: O código gera grafos não direcionados, ou seja, se existe uma aresta entre os vértices A e B, também existe uma aresta entre os vértices B e A.
- Formato de Saída: O grafo gerado é escrito em um arquivo de texto no formato de lista de adjacências. Cada linha do arquivo representa um vértice e seus vizinhos. Por exemplo, a linha 0 -> [1, 3, 4] indica que o vértice 0 possui arestas com os vértices 1, 3 e 4.

4 Resultados

4.1 Análise do Algoritmo Guloso de Conjunto Dominante

Entrada:

O algoritmo trabalha com um grafo representado por uma lista de adjacência, onde:

- V é o número de vértices (nós).
- E é o número de arestas.

Vamos analisar a complexidade em termos do número de vértices V e arestas E.

(1) Complexidade de Inicialização: A primeira parte da função principal (`encontrarConjuntoDominante`) cria um array booleano `visitado` de tamanho V e uma lista `conjuntoDominante`. A inicialização desses dois objetos tem complexidade $O(V)$.

(2) Loop Principal (Selecionar Vértices): O loop principal executa enquanto nem todos os vértices foram cobertos, o que depende do número de vértices e do tamanho do conjunto dominante. No pior caso, o conjunto dominante pode conter até V vértices.
   - **Laço Externo (while loop)**
     - Esse laço pode rodar no máximo V vezes, pois em cada iteração pelo menos um vértice é marcado como visitado.
     - Isso nos dá uma contribuição de $O(V)$ iterações no pior caso.
   - **Escolher o Melhor Vértice (laço for)**

  - Para cada iteração do laço principal, o algoritmo verifica todos os vértices que ainda não foram visitados. A verificação de cada vértice percorre sua lista de adjacência, o que tem uma complexidade proporcional ao grau desse vértice.
  - No pior caso, o laço percorre todos os vértices e suas listas de adjacência, resultando em uma complexidade de $O(V + E)$ para cada iteração.

(3) Marcar Vizinhos como Visitados: Para o vértice escolhido como o "melhor", o algoritmo percorre todos os seus vizinhos e os marca como visitados. O número de vizinhos de um vértice é limitado pelo grau do vértice. Em um grafo esparso, o grau é pequeno; em um grafo denso, pode ser proporcional a V. A complexidade desse passo é $O(\text{grau}(v))$ para cada vértice v escolhido, e no total, ao longo de todo o algoritmo, será no máximo $O(E)$, já que todas as arestas serão percorridas no final.

Complexidade Total: A complexidade de cada iteração do loop principal é $O(V + E)$. No pior caso, o laço externo pode rodar até V vezes, o que resulta em uma complexidade total de:

$$O(V) \times O(V + E) = O(V(V + E)) = O(V^2 + VE)$$

Complexidade Final

- Para grafos esparsos, onde o número de arestas E é pequeno (proporcional a V), a complexidade será aproximadamente $O(V^2)$.
- Para grafos densos, onde E é proporcional a V^2, a complexidade pode crescer até $O(V^3)$.

A complexidade de tempo do algoritmo guloso para o problema do conjunto dominante, conforme implementado, é $O(V^2 + VE)$. Dependendo da densidade do grafo, essa complexidade pode ser reduzida a $O(V^2)$ para grafos esparsos, ou crescer até $O(V^3)$ para grafos densos.

Gráficos de tempo de execução por tamanho da instância

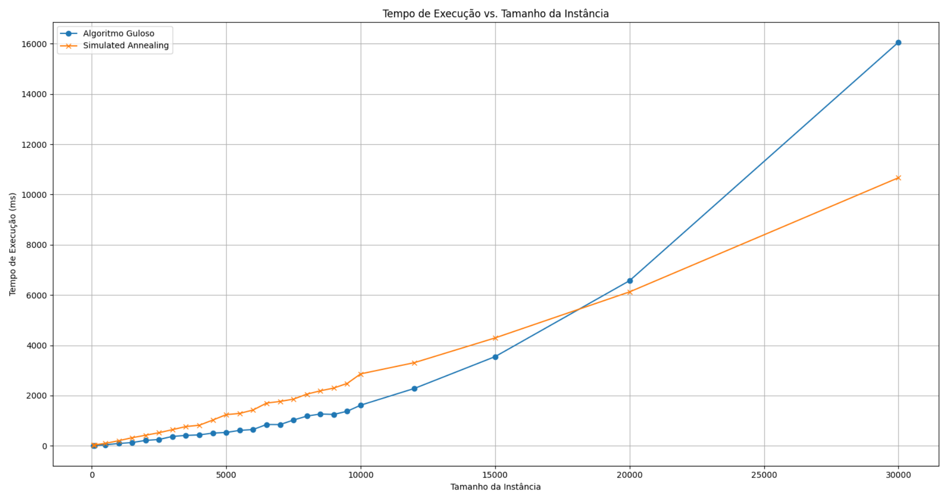

Fig. 1. Quantidade de arestas igual a 10% do total de arestas possíveis (grafo mais esparso).

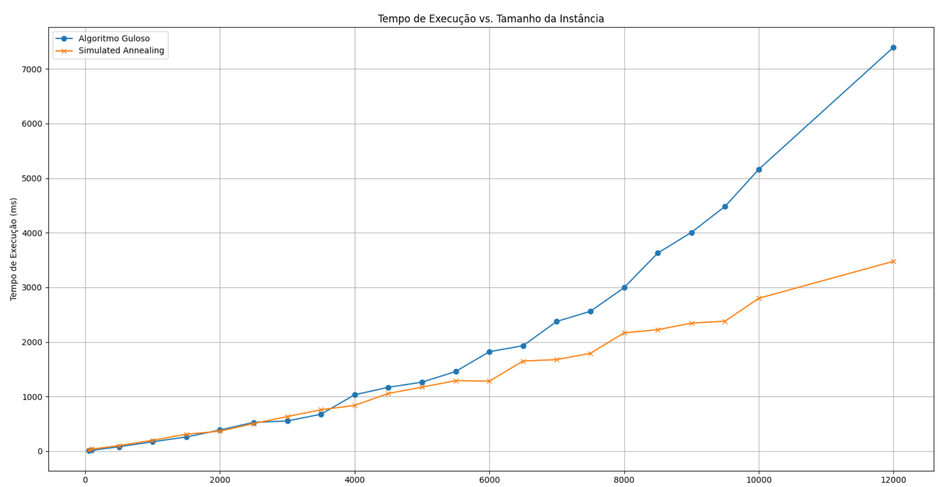

Fig. 2. Quantidade de arestas igual a 50% do total de arestas possíveis.

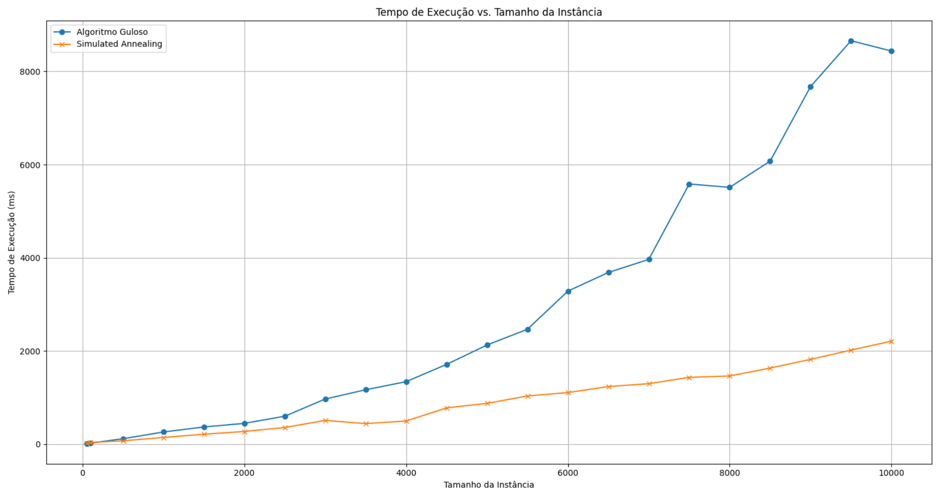

Fig. 3. Quantidade de arestas igual a 90% do total de arestas possíveis (grafo mais denso).

Validação Empírica da Complexidade do Algoritmo Guloso

Para verificar se o tempo de execução do algoritmo guloso realmente segue a complexidade teórica $O(V^2 + VE)$, foram aplicadas três abordagens de análise experimental:

(1) **Regressão Polinomial:**
A primeira abordagem consiste em aplicar regressão polinomial sobre os dados de tempo de execução, com o objetivo de identificar o grau do polinômio que melhor se ajusta ao comportamento observado.
  1.1 Para grafos esparsos, aplicou-se uma regressão de segundo grau ($O(V^2)$), resultando em um coeficiente de determinação R^2 próximo de 1. Isso indica que os tempos medidos crescem aproximadamente como o quadrado do número de vértices.
  1.2 Para grafos densos, utilizou-se uma regressão de terceiro grau ($O(V^3)$), também com ajuste quase perfeito, confirmando o crescimento cúbico esperado.

Fig. 4. Regressão Quadrática (grafo mais esparso).

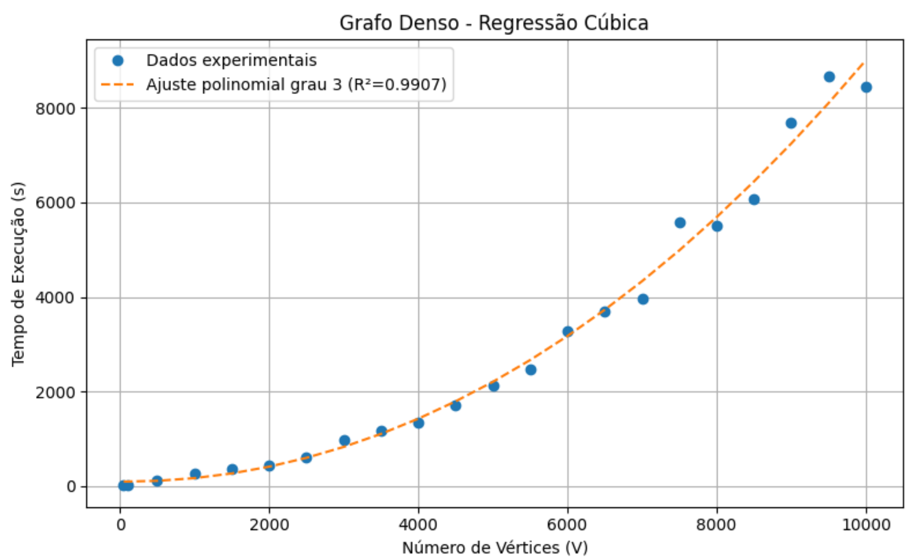

Fig. 5. Regressão Cúbica (grafo mais denso).

(2) **Normalização por Complexidade Esperada**

- Para grafos esparsos, foi calculado $\frac{T(V)}{V^2}$
- Para grafos densos, foi calculado $\frac{T(V)}{V^3}$

Se o algoritmo realmente possui essa ordem de complexidade, os valores normalizados devem permanecer aproximadamente constantes.

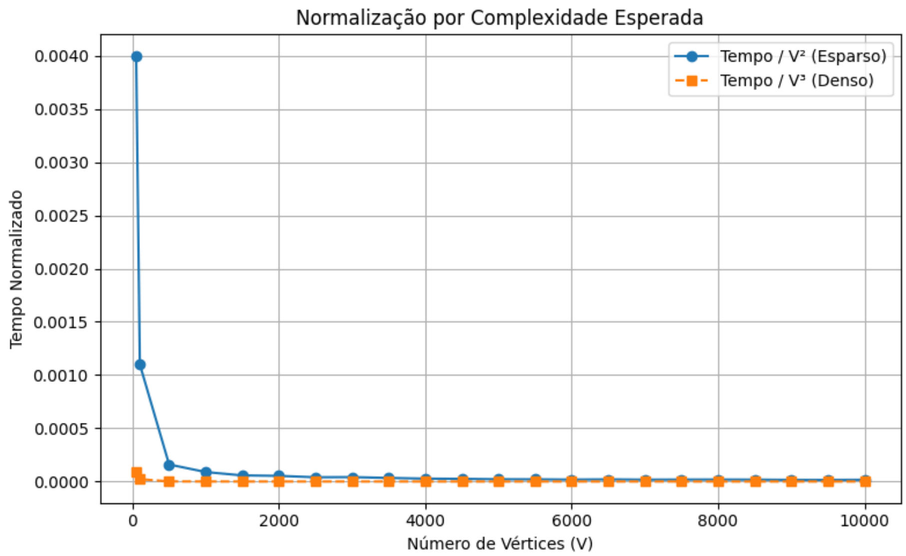

Fig. 6. Normalização.

(3) **Comparação com Curvas Teóricas**

As curvas de V^2 e V^3 foram geradas com base nos própios dados experimentais e sobrepostas aos gráficos de tempo real. Cada curva foi escalada para coincidir com o primeiro e último ponto experimental, permitindo uma comparação direta da tendência de crescimento.

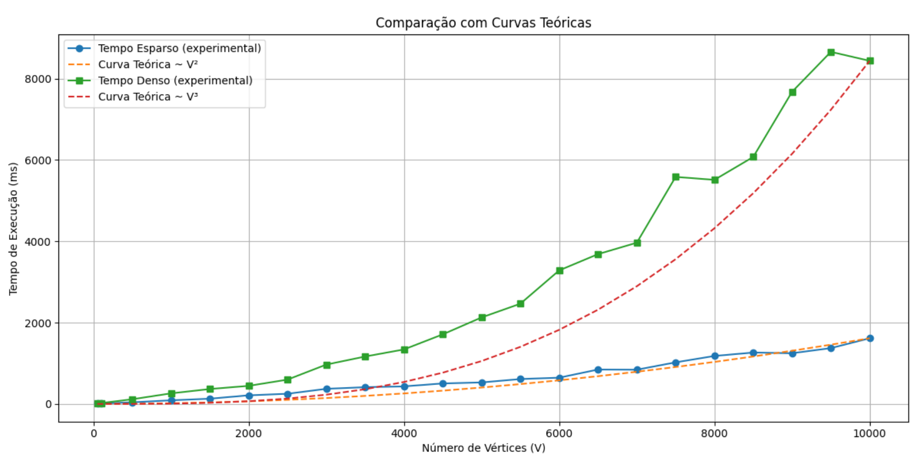

Fig. 7. Comparação experimental com curvas teóricas.

Conclusão:

O método da regressão fornece uma evidência quantitativa de que o comportamento empírico do algoritmo está alinhado com a análise de complexidade teórica. Além disso, os gráficos de tempo normalizado mostraram linhas horizontais e estáveis, indicando que o crescimento dos tempos acompanha de forma consistente as curvas V^2 e V^3, respectivamente. E o resultado da comparação de curvas foi uma sobreposição visual consistente entre os dados e as curvas teóricas, especialmente nas instâncias menores (onde o tempo é mais confiável). Isso reforça a conclusão de que os tempos medidos seguem, respectivamente, $O(V^2)$ e $O(V^3)$ conforme o grau de densidade do grafo.

Os testes realizados confirmaram a análise teórica de complexidade do algoritmo guloso para o problema do conjunto dominante. Observou-se que, em grafos esparsos, o algoritmo apresenta comportamento quadrático O(V^2), sendo relativamente eficiente para um número moderado de vértices. No entanto, à medida que a densidade do grafo aumenta, a complexidade do algoritmo cresce de maneira significativa, tendendo a O(V^3). Esse comportamento torna o algoritmo intratável para grafos densos com um grande número de vértices, exigindo um tempo de execução proibitivo para instâncias maiores. Esses resultados destacam a limitação do uso do algoritmo guloso em cenários de alta densidade, sugerindo a necessidade de métodos alternativos, como heurísticas mais sofisticadas ou técnicas de redução de problemas, para tratar instâncias complexas de forma eficiente.