Trabalho de Grafos - Parte 3

Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

Guloso, Guloso Randomizado e Reativo

Davi Kirchmaier Paiva

Felipe Souza Magalhães Sant'Anna

Rayssa Amaral Gomes

Vitória Cristina Nunes dos Santos

Lucas Henrique Arruda

16 de março de 2025

Sumário

1 Descrições

1.1 Descrição do Problema

1.1.1 Descrição Geral

O problema de cobertura de arestas em grafos consiste em encontrar um conjunto mínimo de vértices de um grafo de modo que cada aresta do grafo esteja conectada a pelo menos um desses vértices. Ou seja, buscamos um subconjunto de vértices que cubra todas as arestas do grafo.

1.1.2 Casos de Uso

Esse problema **é relevante** em diversas aplicações, como:

- **Redes de comunicação:** Minimizar o número de servidores em uma rede enquanto mantém conectividade.
- **Otimização de logística:** Escolha de pontos estratégicos para distribuição de recursos.
- **Bioinformática:** Análise de interações genéticas e redes biológicas.

A solução **ótima** pode ser obtida através de métodos exatos, como programação linear e algoritmos de emparelhamento máximo, ou por meio de heurísticas para casos em que a complexidade computacional **é um fator limitante**, já que são soluções aproximadas mais eficientes para grandes instâncias.

1.1.3 Complexidade

A complexidade do problema varia dependendo da estrutura do grafo. Em grafos bipartidos, por exemplo, a cobertura de arestas pode ser encontrada de maneira eficiente a partir de um emparelhamento máximo. No entanto, em grafos gerais, encontrar a solução **ótima** pode ser mais desafiador, já que exige abordagens heurísticas para viabilizar a solução em tempo hábil.

2 Descrição das Instâncias

2.1 Descrição Geral

Durante a pesquisa e procura por instâncias feita pelo grupo, chegamos à conclusão que o problema de cobertura de arestas, proposto como tema do trabalho, apresentava instâncias muito grandes, com a maioria dos problemas se tratando de situações relacionadas a redes sociais, com média de mais de **700 mil nós**, o que seria extremamente lento e demorado para uma rodagem. Portanto, visando conseguir atingir nosso objetivo alvo em tempo hábil, foi criado um script em Python (presente dentro do repositório no arquivo `main.py`) que gerou **10 instâncias aleatórias** com número de nós entre 5000 e 7501, para facilitar o desenvolvimento de uma solução para o tema.

3 Descrição dos Métodos Implementados

3.1 Descrição Geral

Para resolver o problema de cobertura de arestas, implementamos três abordagens heurísticas:

3.1.1 Algoritmo Guloso

O algoritmo guloso toma decisões baseadas apenas no benefício imediato, escolhendo sempre a melhor opção local sem considerar consequências futuras. **Passos do algoritmo:**

- Inicializa uma solução vazia.
- Seleciona o vértice que cobre o maior número de arestas ainda não cobertas.
- Repete o processo até que todas as arestas sejam cobertas.

Vantagens:

- Rápido e eficiente para pequenas instâncias.
- Fácil de implementar.

Desvantagens:

- Pode ficar preso em **ótimos locais**.
- Não garante a melhor solução global.

3.1.2 Algoritmo Randomizado

Esse método introduz aleatoriedade no processo de escolha, evitando que o algoritmo fique preso em uma única solução fixa. **Passos do algoritmo:**

- Em vez de escolher sempre a melhor opção, seleciona aleatoriamente entre um conjunto de boas opções.
- O processo **é repetido várias vezes** para gerar diferentes soluções.
- A melhor solução **encontrada** entre múltiplas execuções **é escolhida**.

Vantagens:

- Evita **mínimos locais**.
- Pode gerar soluções melhores do que o guloso.

Desvantagens:

- Tempo de execução maior, pois precisa rodar várias vezes.
- A qualidade da solução depende do número de repetições.

3.1.3 Algoritmo Reativo

O método reativo se adapta dinamicamente às condições do problema, ajustando suas escolhas com base no histórico de execuções. **Passos do algoritmo:**

- Inicializa um conjunto de estratégias e mede o desempenho de cada uma ao longo do tempo.
- Ajusta dinamicamente a estratégia escolhida com base nos resultados obtidos.
- Adapta-se **ás características do grafo** durante a execução.

Vantagens:

- Melhora os resultados ao longo do tempo.
- Se adapta às características do problema.

Desvantagens:

- Maior custo computacional devido ao monitoramento contínuo.
- Implementação mais complexa.

4 Análises

4.1 Análise de Tempo de Execução entre Lista e Matriz

4.1.1 Descrição Geral

Os três algoritmos foram comparados em relação ao tempo de execução, e os resultados foram os seguintes:

- **Algoritmo Guloso:** Muito rápido.
- **Algoritmo Randomizado:** Tempo médio, dependendo do número de repetições.
- **Algoritmo Reativo:** Mais lento devido à adaptação dinâmica.

O Guloso teve a menor complexidade, mas com soluções de menor qualidade. O Randomizado apresentou desempenho intermediário, enquanto o Reativo gerou os melhores resultados, porém com maior custo computacional.

4.2 Análise de Resultado com Teste de Hipótese entre os Métodos

4.2.1 Descrição Geral

Realizamos uma análise estatística para comparar o desempenho das abordagens:

- H_0 (Hipótese Nula): Não há diferença significativa entre os algoritmos.
- H_1 (Hipótese Alternativa): Existe diferença significativa de desempenho.

Resultados dos Testes Estatísticos (ANOVA):
Os testes mostraram que:

Tabela 1: Resultados do Teste ANOVA

Métrica Estatística	Valor Calculado	Resultado
Estatística F	13.13	–
Valor p	0.0001	Rejeita-se H_0 ($p < 0.05$)

Tabela 2: Resultados Da Cobertura De Arestas

Execução	Guloso	Randomizado	Reativo
1	56	58	56
2	69	62	55
3	78	50	36
4	64	50	58
5	60	63	46
6	57	60	40
7	78	43	36
8	70	47	55
9	56	63	35
10	75	42	46

- O Guloso apresentou os piores resultados.
- O Randomizado obteve melhores soluções, mas sua qualidade variava.
- O Reativo teve a melhor performance estatística.

O teste confirmou que H_0 foi rejeitada, ou seja, os métodos têm diferenças estatisticamente significativas.

5 Conclusões

5.1 Conclusão

5.1.1 Descrição Geral

Com base nos testes realizados, concluímos que:

- O Guloso **é o mais rápido**, mas gera soluções de menor qualidade.
- O Randomizado melhora os resultados, mas exige várias execuções, logo, mais tempo.
- O Reativo **é o mais eficiente na qualidade da solução**, mas tem maior custo computacional.

A escolha do algoritmo depende da prioridade entre velocidade e qualidade da solução para a aplicação específica.

5.1.2 Limitações do Código

O código conta com algumas limitações, tais como:

- **Uso de matriz de adjacência em grafos grandes:** Usa $O(V^2)$ de memória, o que torna inviável trabalhar com grafos muito grandes (ex.: 1 milhão de vértices exigiria 3.7 TB).
- **Tempo de execução em instâncias menores:** O reativo pode ser lento para centenas de milhares de vértices.
- **Qualidade das soluções aproximadas:** O guloso pode gerar soluções ruins; o randomizado depende de múltiplas execuções; o reativo ainda pode ser subótimo.
- **Ausência de algoritmos exatos:** Não há métodos como Programação Linear ou Branch and Bound.
- **Falta de paralelismo:** Não aproveita multithreading.
- **Ausência de gerenciamento de memória eficiente:** Sem estruturas compactas ou liberação automática.
- **Tratamento de erros e robustez:** Sem validação de entradas ou logs detalhados.
- **Suporte limitado a diferentes tipos de grafos:** Não suporta grafos dinâmicos ou hipergrafos.
- **Dependência de formato de entrada:** Requer conversão manual para outros formatos.

Embora eficiente para instâncias médias, o código enfrenta desafios para grandes volumes de dados. Melhorias em memória, paralelismo e algoritmos exatos podem torná-lo mais robusto.