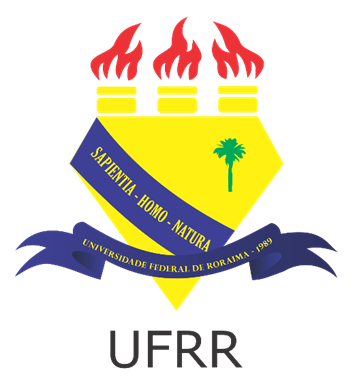
****

**UNIVERSIDADE FEDERAL DE RORAIMA**

**CIÊNCIA DA COMPUTAÇÃO - GRADUAÇÃO**

**RELATÓRIO DO PROJETO: Problema do Clique**

**Boa vista – Roraima**

**2025**

**MARCUS VINÍCIUS MAIA DOS SANTOS**

**FILIPE GABRIEL TOMAZ BRITO**

**RELATÓRIO DO PROJETO: Problema do Clique**

Relatório técnico apresentado ao Prof. Dr. Herbert Oliveira Rocha, como requisito de obtenção de nota parcial na disciplina DCC606 - ANÁLISE DE ALGORITMOS.

**Boa vista – Roraima**

**2025**

**RESUMO**

Este trabalho aborda o problema do clique em grafos, com aplicação prática sobre interações de usuários no Twitter. Foi implementado um algoritmo guloso para detecção de cliques em uma matriz de adjacência gerada a partir de um dataset real. Além disso, foi utilizada uma instância benchmark (Zachary's Karate Club) para avaliação experimental e comparação. A análise inclui aplicação prática, complexidade computacional e considerações sobre o artigo "Finding All Maximal Connected s-Cliques in Social Networks" (EDBT 2018).

**1. INTRODUÇÃO**

O problema do clique é clássico na teoria dos grafos e possui aplicações em diversas áreas, como redes sociais, bioinformática e comunicação. Este trabalho visa detectar cliques em redes de interações no Twitter para identificar grupos coesos de usuários. A abordagem é baseada em um algoritmo guloso simples, capaz de processar grafos reais de forma eficiente.

**2. ANÁLISE DO ARTIGO PROPOSTO**

O artigo “Finding All Maximal Connected s-Cliques in Social Networks” propõe algoritmos eficientes para detectar s-cliques, onde cada par de vértices está conectado com distância máxima s. Diferente do clique tradicional, a s-clique é mais tolerante a ausência de conexões diretas.

Dois algoritmos principais são propostos:

* Naïve-sClique: verifica todas as combinações possíveis (ineficiente);
* Smart-sClique: aplica poda e ordenação inteligente para encontrar s-cliques de forma rápida.

Os autores apresentam experimentos em datasets reais de redes sociais e comparam desempenho, cobertura de cliques e escalabilidade. O foco principal é a eficiência em encontrar subconjuntos densos e bem conectados que não sejam necessariamente cliques completos.

O algoritmo guloso deste projeto pode ser visto como uma especialização do caso em que **s =** 1, ou seja, cliques completos onde todos os pares estão diretamente conectados. Apesar de simples, ele é eficiente e aplicável em situações reais. A simplicidade do algoritmo é compensada por sua aplicabilidade imediata e facilidade de implementação em sistemas reais de análise de redes.

**3. ANÁLISE DE COMPLEXIDADE**

O algoritmo guloso percorre cada vértice e tenta adicionar outros vértices à clique corrente, desde que estejam conectados a todos os membros já presentes. Para cada vértice u, percorremos todos os outros vértices v, e verificamos se v está conectado a todos os elementos da clique parcial. Essa verificação custa até n acessos, e é feita n vezes. Segue abaixo a análise:  
Vamos chamar:

* T(n) = tempo total do algoritmo.
* C(u) = custo de executar greedy\_clique() a partir de um vértice u.

Passo 1: Laço externo

Executa n vezes:

Passo 2: Laço interno por vértice v

Para cada u, no pior caso, o laço:

for (int v = 0; v < n; v++)

Executa n vezes.

Passo 3: Cada chamada de is\_clique() verifica conexão de v com os elementos atuais da clique

Supondo que o clique tem tamanho k, e que no pior caso k = n:

* Cada verificação custa O(k) = O(n)
* Como fazemos isso para cada v, o custo total do laço interno é:

Passo 4: Laço externo repetido para cada u ⇒ n vezes

Portanto:

Assim:

* Laço externo: O(n)
* Laço interno: O(n)
* Verificação de clique: O(n)

Complexidade total no pior caso: **O(n³)**.  
Uso de memória: **O(n²)**, devido à matriz de adjacência.

Em termos de desempenho prático, o algoritmo se mostrou viável para grafos com até alguns milhares de vértices. Em grafos muito densos ou muito grandes, no entanto, a complexidade cúbica pode se tornar um fator limitante, especialmente em sistemas com pouca RAM ou sem paralelização.

**4. APLICAÇÃO PRÁTICA**

O projeto tem como aplicação a análise de redes de atendimento ao cliente no Twitter. Cada usuário do Twitter foi representado por um vértice, e uma aresta foi criada entre dois vértices quando houve uma interação entre eles (resposta, reply ou menção). Ao identificar cliques nesse grafo, é possível:

* Detectar grupos de usuários com problemas semelhantes;
* Agrupar reclamações automaticamente;
* Priorizar cliques maiores como possível indício de falhas sistêmicas;
* Sugerir respostas padronizadas para grupos semelhantes;
* Identificar comportamentos coordenados (como bots ou spam).

Essa abordagem pode ser expandida para redes maiores e adaptada para sistemas automatizados de atendimento inteligente em redes sociais corporativas. A identificação automática de clusters bem conectados é uma ferramenta poderosa para reduzir o tempo de resposta e alocar recursos de forma eficiente

**5. IMPLEMENTAÇÃO DOS PROGRAMAS**

Para iniciar com a implementação, nós primeiros analisamos o dataset disponível em <https://www.kaggle.com/datasets/thoughtvector/customer-support-on-twitter> e então definimos o conjunto que analisamos, sendo analisados até 1.000 tweets, visto que 10.000 iria poluir a saída do programa principal e a partir de 1.000.000 de tweets o arquivo twcs.csv congelava o computador. Adiante implementamos o pseudocódigo representando nosso programa principal:

Para cada vértice u não visitado:

Inicie uma nova clique com u

Marque u como visitado

Para cada vértice v não visitado:

Se v está conectado a todos os vértices da clique atual:

Adicione v à clique

Marque v como visitado

Imprima a clique encontrada

Definidas tais etapas, passamos a implementar os programas:

* grafo.py (gerador de matriz): Lê o twcs.csv, extrai até os 1000 primeiros tweets por limitação de performance e gera uma matriz de adjacência simétrica (grafo não direcionado) com base nas interações entre author\_id e text.
* click.c: Responsável pela leitura da matriz de adjacência e aplicação do algoritmo guloso. O código identifica cliques locais ao longo dos vértices, imprimindo cada clique encontrado e seu tamanho. A saída é exibida no terminal e pode ser redirecionada para arquivo para análises posteriores.
* matriz.c: Auxiliar para testes de leitura e verificação de integridade da matriz gerada. Garante que a matriz de entrada está corretamente formatada e acessível. Também pode ser usado para debugar problemas de entrada no algoritmo principal.

**6. AVALIAÇÃO EXPERIMENTAL**

O benchmark Zachary’s Karate Club foi escolhido por ser amplamente conhecido e utilizado em estudos de grafos sociais. Trata-se de um grafo pequeno, com apenas 34 vértices e 78 arestas, o que facilita tanto a leitura quanto a interpretação dos resultados. Ele é frequentemente adotado para testar algoritmos relacionados à detecção de comunidades e cliques, justamente por sua estrutura bem definida e amplamente documentada na literatura. Além disso, o grafo está disponível em diversos formatos, como lista de adjacência ou lista de arestas, o que facilita sua integração com diferentes ferramentas. Outro fator relevante é que os cliques encontrados nesse grafo podem ser validados visualmente com o auxílio de ferramentas como Gephi ou bibliotecas como NetworkX, o que reforça sua utilidade como benchmark confiável para testes experimentais.

Foram utilizados dois grafos:

* karate.txt: Grafo clássico com 34 vértices.
* matriz\_adjacencia.txt: Gerado a partir do Twitter com 1066 vértices.

Resumo dos resultados:

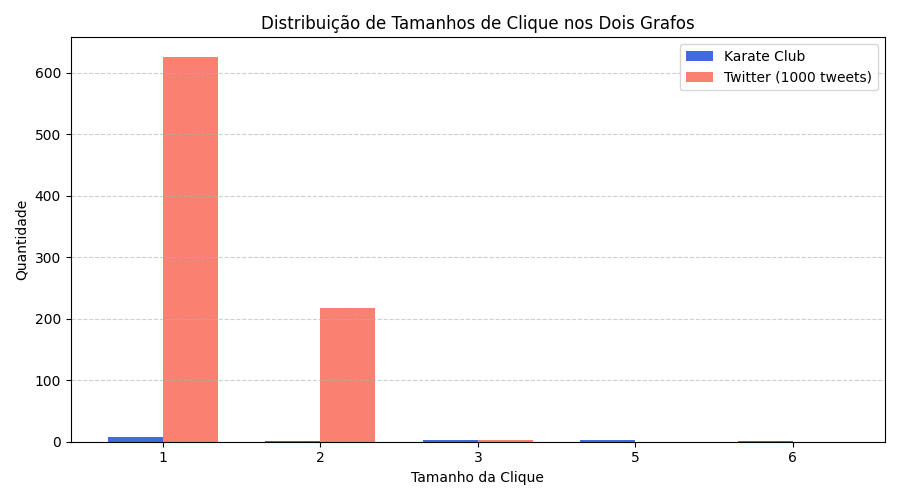
* karate.txt:
  + Total de cliques: 14
  + Tamanho máximo da clique: 6
  + Tempo de execução: 0.03s
* matriz\_adjacencia.txt:
  + Total de cliques: 845
  + Apenas duas cliques de tamanho 3
  + Tempo de execução: 0.111s

Tabela com resultados de cliques da matriz\_adjacencia.txt

| **Tamanho do clique** | **Quantidade de cliques** |
| --- | --- |
| 1 | 626 |
| 2 | 217 |
| 3 | 2 |

Tabela com resultados de cliques do karate.txt:

| **Tamanho do clique** | **Quantidade de cliques** |
| --- | --- |
| 1 | 7 |
| 2 | 1 |
| 3 | 3 |
| 5 | 2 |
| 6 | 1 |



Fonte: Elaborada pelos autores.

Esses resultados demonstram a eficácia do algoritmo em grafos reais, mas também refletem limitações naturais das conexões sociais — em especial no Twitter, onde poucos grupos são densamente conectados. O grafo do Karate Club, por outro lado, apresenta uma maior coesão entre os vértices, refletindo a estrutura de comunidade estudada.

**7. CONSIDERAÇÕES FINAIS**O projeto demonstrou que é possível aplicar um algoritmo guloso simples para detectar cliques em grafos reais, obtendo resultados coerentes com a estrutura das redes. Apesar de sua limitação em encontrar apenas cliques locais e não garantir o clique máximo global, o algoritmo mostrou-se eficiente e com boa aplicação prática. A relação com o artigo é conceitual, pois o algoritmo implementado é equivalente ao caso s = 1 abordado pelos autores.

A abordagem é ideal como ponto de partida para problemas maiores de descoberta de comunidades. O uso de benchmarks conhecidos e de dados reais garante uma avaliação equilibrada entre controle e aplicabilidade. Como extensão, seria possível aplicar técnicas de paralelização ou heurísticas mais avançadas para escalar o algoritmo a grafos com dezenas ou centenas de milhares de vértices.

**8. REPOSITÓRIO**

<https://github.com/marcusv0/FinalProject_DCC606_Tema_5_RR_2025>