Clique OF size k – Análise de Soluções de Decisões de Problemas

André Almeida Oliveira

***Resumo* - O problema de encontrar um clique de tamanho k num grafo consiste em identificar um subconjunto de vértices do grafo, no qual todos os vértices estão conectados entre si e cujo tamanho seja exatamente k. Este problema tem grande relevância em áreas como redes sociais, bioinformática e análise de redes, onde identificar subgrupos densamente conectados pode trazer insights valiosos. Neste artigo, exploram-se duas abordagens distintas para a solução do problema: pesquisa exaustiva e busca voraz. Foi realizada uma análise formal e experimental de cada uma delas de forma a compará-las quanto aos parâmetros considerados.**

***Abstract* - The problem of finding a clique of size k in a graph consists of identifying a subset of vertices in which all vertices are mutually connected, and the subset has exactly k vertices. This problem is highly relevant in fields such as social networks, bioinformatics, and network analysis, where identifying densely connected subgroups can yield valuable insights. This article explores two distinct approaches to solve the problem: exhaustive search and greedy search. A formal and experimental analysis of each approach was conducted to compare them with respect to the parameters considered.**

Keywords - Clique Problem, Exhaustive Search, Greedy Search, Graph Theory

I. INTRODUÇÃO

A. Contextualização

O problema do clique em grafos, definido como a tarefa de encontrar um subconjunto específico de vértices totalmente conectados, tem gerado interesse considerável devido à sua aplicabilidade e complexidade. Este problema surge de maneira natural em contextos onde é necessário identificar grupos densamente conectados em redes, como comunidades em redes sociais, agrupamentos de genes em redes biológicas, ou grupos de cooperação em redes de comunicação. Um exemplo prático seria a identificação de um grupo de amigos nas redes sociais que interagem frequentemente, ou a descoberta de moléculas em redes biológicas que interagem de maneira consistente.

Formalmente, o objetivo é encontrar um clique de tamanho k, ou seja, um grupo de k vértices de um grafo, onde cada vértice está conectado a todos os outros dentro do grupo. No entanto, à medida que k aumenta, o número total de vértices aumenta e a densidade de arestas diminui, a dificuldade de resolução desse problema cresce significativamente, pois o número de combinações possíveis para examinar torna-se exponencial em n. Por essa razão, o problema de encontrar cliques de tamanho k é classificado como NP-completo, uma classe de problemas onde a solução é fácil de verificar, mas difícil de encontrar.

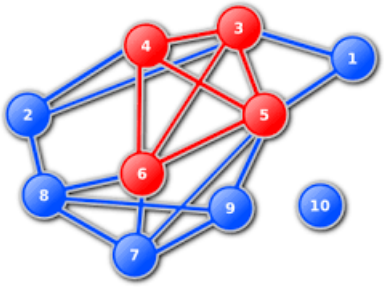

Fig. 1 - Exemplo visual de um clique de tamanho 4 (vértices 3, 4, 5 e 6) num grafo

B. Desafios e Abordagens

Devido à sua natureza complexa, diversas abordagens têm sido propostas para encontrar cliques de tamanho k de maneira eficiente. Cada uma dessas técnicas oferece um compromisso entre precisão e tempo de execução, o que é particularmente relevante em grafos de grande escala.

- **Pesquisa Exaustiva**

Esta é a abordagem mais direta, onde todas as combinações de vértices são verificadas para ver se formam um clique. Apesar de ser precisa, o seu custo computacional é muito elevado para grafos grandes, pois o número de combinações cresce exponencialmente com o número de vértices.

- **Pesquisa Voraz – Simple Greedy**

Esta técnica utiliza uma heurística simples, como a seleção de vértices com maior grau (número de conexões), para construir um clique rapidamente. Apesar de ser bastante mais rápida, eficiente e com uma alta precisão, teoricamente, não garante uma solução sempre correta.

- **Pesquisa Voraz – Greedy Intersection**

Esta abordagem é uma variação da acima, que organiza os vértices por grau e inicia o processo de construção de um clique a partir de um vértice com alto grau. A técnica

baseia-se na interseção de vizinhança, escolhendo o próximo vértice a ser adicionado ao clique com base numa maximização das conexões compartilhadas entre os seus vizinhos e o conjunto parcial de vértices do clique. Este método permite priorizar vértices mais conectados, acelerando a formação de cliques densos e oferecendo um equilíbrio entre eficiência e precisão.

II. PESQUISA EXAUSTIVA

O algoritmo de pesquisa exaustiva, consiste em explorar todas as combinações possíveis de vértices até identificar uma solução que satisfaça as condições do problema. Neste caso, buscamos apenas um subconjunto de k vértices no qual todos os vértices estejam conectados entre si, formando assim um clique.

A abordagem adotada percorre todos os vértices do grafo, gerando todas as combinações possíveis de k vértices e verificando se eles formam um clique. A cada combinação testada, o algoritmo confirma se existe uma conexão direta entre todos os pares de vértices no subconjunto atual. Assim que um clique de tamanho k é encontrado, o algoritmo encerra a busca.

Para colocar em prova este conceito foi desenvolvido código a partir do pseudo-código seguinte:

```
FUNÇÃO exhaustive_clique_search(grafo, tamanho_clique):
 lista_nodes ← lista de nodes do grafo
 soluções_testadas ← 0
 operações_realizadas ← 0

 PARA cada combinação em todas as combinações de
lista_nodes com tamanho igual a tamanho_clique:
  soluções_testadas ← soluções_testadas + 1
  operações_realizadas ← operações_realizadas + 1
  SE é_clique(grafo, combinação):
   RETORNAR combinação, operações_realizadas,
soluções_testadas

 RETORNAR Nulo, operações_realizadas, soluções_testadas
```

A. Análise Formal

A complexidade do algoritmo pode ser analisada da seguinte forma. O algoritmo utiliza um único loop que percorre todas as combinações de k vértices entre os n vértices do grafo, o que resulta num número total de combinações $O\left(\binom{n}{k}\right)$. Para cada combinação, a função is_clique() é chamada, e a sua complexidade é $O(k^2)$, já que ela verifica as arestas entre todos os pares de vértices no subconjunto de k vértices.

Portanto, a complexidade total do algoritmo é $O\left(\binom{n}{k} \cdot k^2\right)$, que pode ser aproximada para $O\left(\frac{n^k}{k!} \cdot k^2\right)$. Isso implica que o algoritmo apresenta um crescimento exponencial em relação ao número de vértices n, especialmente quando k é grande.

Dado que o algoritmo realiza uma busca exaustiva, a complexidade difere ligeiramente entre os melhores, médios e piores casos. No melhor caso, o algoritmo para assim que encontra um clique de tamanho k, o que pode ocorrer antes de todas as combinações serem verificadas, resultando num tempo de execução inferior. Contudo, no caso médio e no pior caso, especialmente em grafos onde cliques de tamanho k são raros ou inexistentes, o algoritmo percorrerá uma fração significativa ou todas as combinações de k vértices. Assim, a estrutura do grafo influencia o número de combinações testadas, mas, em termos de complexidade assintótica, continua a ser $O\left(\binom{n}{k} \cdot k^2\right)$.

B. Análise Experimental

Para verificar os resultados estipulados no processo de análise formal, foram gerados grafos com diferentes configurações. A quantidade de vértices variou entre 1 e 256, e as arestas foram distribuídas com densidades de 12,5%, 25%, 50% e 75% em relação ao número máximo de arestas possíveis num grafo com n vértices. Em vez de posicionar os vértices num espaço físico específico, foram criados grafos “livres de coordenadas”, sem restrições geográficas, permitindo uma análise puramente estrutural e topológica. Os parâmetros de teste também incluíram o tamanho do clique (denotado como k), variando entre os valores 5, 6, 7, 8, 9, 10 e 15. Foram utilizadas três métricas para a avaliação: tempo de execução, número de operações e número de soluções testadas.

Neste algoritmo, cada iteração do loop representa uma tentativa de combinação de vértices para formar um clique de tamanho k. As operações a ter em conta para este algoritmo foram as comparações, sendo que existe uma para cada solução testada, e o número de verificações da existência de ligação em cada par de vértices. O tempo de execução acabou por limitar o número máximo de vértices a testar, para este algoritmo, uma vez que os tempos de execução a partir de certo ponto são demasiado elevados para serem aprazíveis.

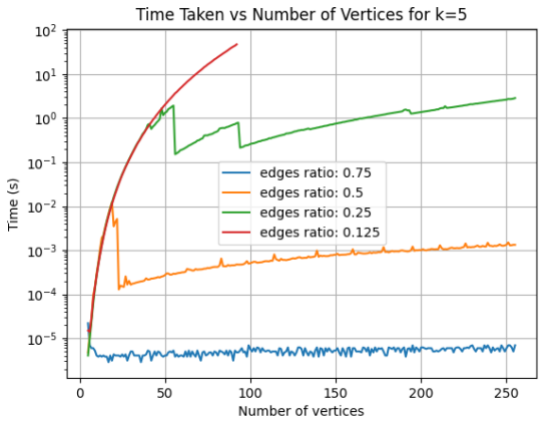

Fig. 2 - Pesquisa Exaustiva: Tempo de execução por grafo com k=5

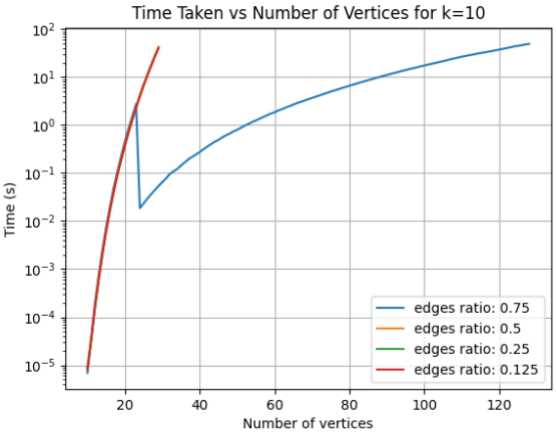

Fig. 3 - Pesquisa Exaustiva: Tempo de execução por grafo com k=10

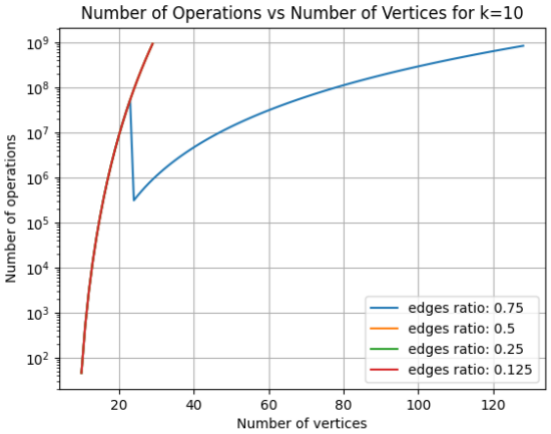

Fig. 6 - Pesquisa Exaustiva: Número de operações básicas por grafo com k=10

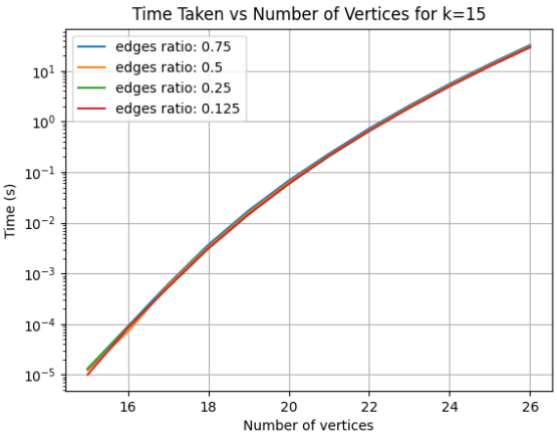

Fig. 4 - Pesquisa Exaustiva: Tempo de execução por grafo com k=15

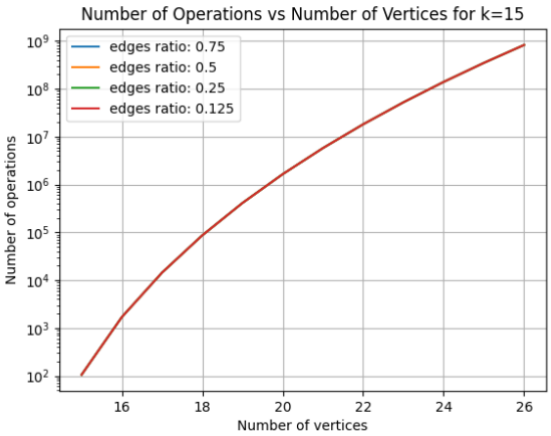

Fig. 7 - Pesquisa Exaustiva: Número de operações básicas por grafo com k=15

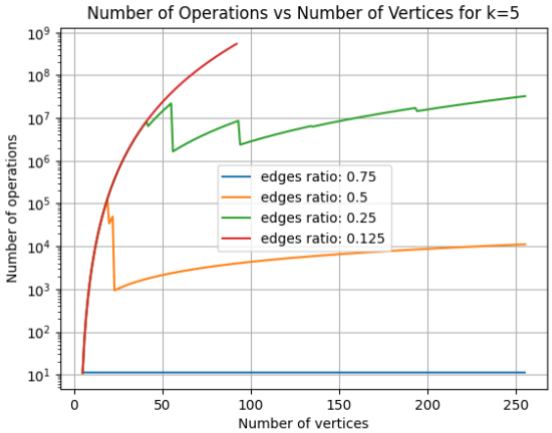

Fig. 5 - Pesquisa Exaustiva: Número de operações básicas por grafo com k=5

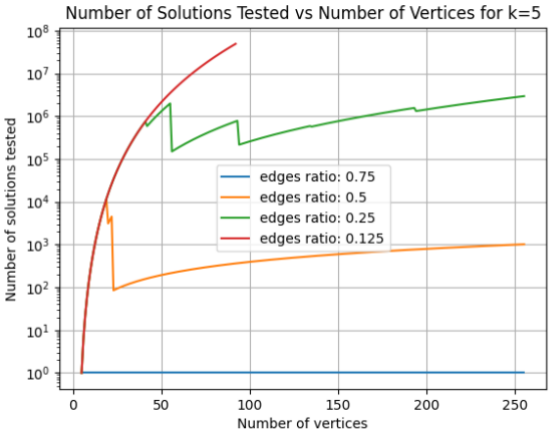

Fig. 8 - Pesquisa Exaustiva: Número de soluções testadas por grafo com k=5

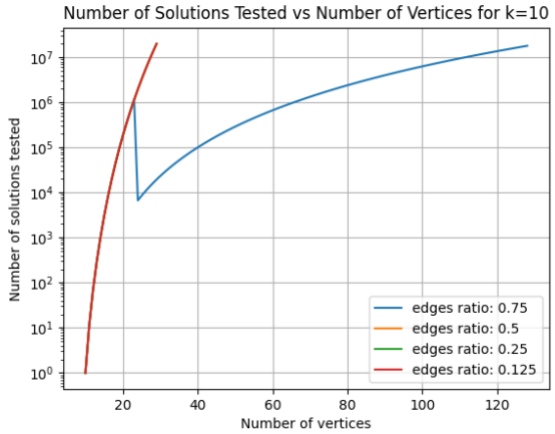

Fig. 9 - Pesquisa Exaustiva: Número de soluções testadas por grafo com k=10

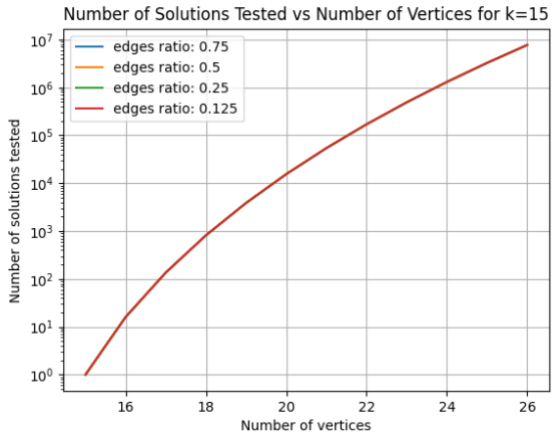

Fig. 10 - Pesquisa Exaustiva: Número de soluções testadas por grafo com k=15

Todos os gráficos usam uma escala logarítmica no eixo y, sendo que é possível verificar um crescimento exponencial em cada um dos gráficos já que este se assemelha a um crescimento linear, mas numa escala de logaritmo.

Observa-se que a pesquisa exaustiva é muito sensível tanto ao tamanho do clique quanto à densidade de arestas do grafo. Para cliques menores (como de tamanho 5), uma alta densidade de arestas ajuda a manter o tempo de execução mais baixo, mas essa vantagem reduz à medida que o tamanho do clique aumenta. Quando o clique cresce, o tempo de execução aumenta drasticamente, independentemente da densidade, tornando a pesquisa exaustiva impraticável para cliques grandes em grafos grandes. Este comportamento, também se reflete no número máximo de vértices dos grafos que o algoritmo conseguiu testar à medida que o tamanho do clique aumentou.

Além disso, é possível perceber que o tempo de execução está diretamente relacionado com o número de operações realizadas pelo algoritmo. Embora o número de operações e o número de soluções testadas não sejam idênticos (pois as operações incluem verificações adicionais, como a operação de verificação de existência de arestas entre os vértices), é claro que a variação desses valores depende mais do tamanho do clique a procurar e da densidade de arestas.

Esta relação entre tempo de execução e número de operações demonstra que, apesar das verificações de arestas, o principal fator determinante para a complexidade do algoritmo é o tamanho do clique a procurar e da densidade de arestas.

Nos gráficos, observa-se que grafos com um número muito pequeno de vértices apresentam métricas mais altas, pois é menos provável encontrar um clique do tamanho desejado, fazendo com que o algoritmo precise de explorar mais combinações possíveis. Esse fenômeno é particularmente percetível quando o número de vértices ultrapassa ligeiramente os 20, momento em que ocorre uma queda acentuada nas diferentes métricas. Isso indica que, com um maior número de vértices, a formação de cliques torna-se mais provável, permitindo ao algoritmo identificar cliques mais rapidamente.

III. PESQUISA VORAZ

Com o objetivo de encontrar cliques em grafos maiores, com mais vértices e arestas, e de comparar o desempenho do algoritmo de pesquisa exaustiva com abordagens alternativas, foram desenvolvidos dois algoritmos de pesquisa voraz. Cada um deles utiliza uma heurística diferente para maximizar a eficiência na busca de cliques e testar a viabilidade desses métodos em comparação com a abordagem exaustiva.

IV. PESQUISA VORAZ – SIMPLE GREEDY

O primeiro algoritmo de pesquisa voraz, referido como Simple Greedy, usa uma heurística básica que organiza os vértices por ordem decrescente de grau (número de conexões) e, em seguida, tenta formar um clique, adicionando iterativamente vértices que estão conectados a todos os vértices do clique parcial atual. Este método foi projetado para uma prova de conceito e busca cliques de forma direta, baseando-se na conectividade local.

Para colocar em prova este conceito foi desenvolvido código a partir do pseudo-código seguinte:

```
FUNÇÃO greedy_clique_search(grafo, tamanho_clique):
  operações_realizadas ← 0
  soluções_testadas ← 0
  lista_nodes_ordenada ← lista de nodes do grafo
ordenada por grau em ordem decrescente

  PARA cada node em lista_nodes_ordenada:
    clique_atual ← conjunto contendo apenas node
      PARA cada node_potencial em lista_nodes_ordenada:
        operações_realizadas ← operações_realizadas + 1

        SE node_potencial ≠ node E TODOS os nodes em
clique_atual estão conectados a node_potencial:
          ADICIONAR node_potencial a clique_atual
```

```
        SE tamanho de clique_atual = tamanho_clique:
          soluções_testadas ← soluções_testadas + 1
          RETORNAR clique_atual como lista,
operações_realizadas, soluções_testadas

  RETORNAR Nulo, operações_realizadas, soluções_testadas
```

A. Análise Formal

A complexidade do algoritmo pode ser analisada da seguinte forma. Primeiro, os vértices do grafo são ordenados por grau em ordem decrescente, uma operação com complexidade $O(n \log n)$, onde n é o número de vértices. Em seguida, o algoritmo percorre essa lista ordenada, tentando formar um clique de tamanho k a partir de cada vértice.

Para cada vértice inicial, o algoritmo verifica os demais vértices, adicionando-os ao clique se todos os vértices no clique atual estiverem conectados ao novo vértice. A verificação de conectividade entre um vértice candidato e o clique atual tem complexidade $O(k)$ em cada iteração, já que precisa verificar todas as arestas entre o vértice candidato e os vértices já presentes no clique.

Dessa forma, a complexidade do algoritmo pode ser representada como $O(n \cdot k)$ para o loop principal de tentativa de formação de clique, somado a $O(n \log n)$ para a ordenação inicial. Assim, a complexidade total é $O(n \log n + n \cdot k)$.

No melhor caso, o algoritmo para ao encontrar o primeiro clique de tamanho k, o que reduz significativamente o tempo de execução. No entanto, no caso médio e no pior caso, especialmente em grafos esparsos onde cliques de tamanho k podem ser raros ou inexistentes, o algoritmo pode percorrer uma grande quantidade de vértices e realizar inúmeras verificações de conectividade. Ainda assim, em termos de complexidade assintótica, permanece $O(n \log n + n \cdot k)$, embora o comportamento real dependa da densidade do grafo e da presença de cliques do tamanho desejado.

B. Análise Experimental

Os grafos utilizados para teste foram gerados da mesma forma do que os utilizados no algoritmo de pesquisa exaustiva. No entanto, para este algoritmo, foi possível avaliar grafos com até 256 vértices de maneira eficiente, já que a sua execução é significativamente mais rápida e menos complexa do que a do algoritmo anterior.

Além das três métricas inicialmente analisadas, este estudo também considera a precisão de cada uma das heurísticas dos algoritmos, removendo o número de soluções testadas, pois aqui apenas a solução final de tamanho k é verificada.

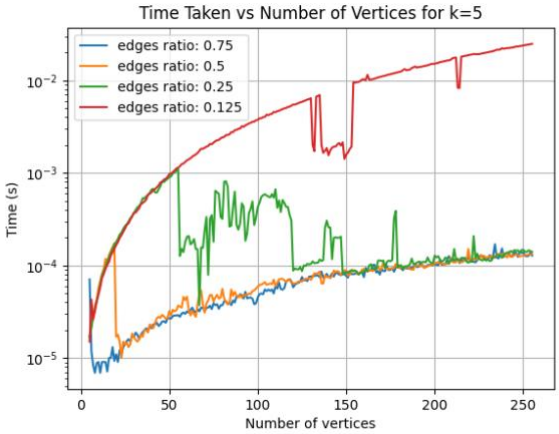

Fig. 11 - Simple Greedy: Tempo de execução por grafo com k=5

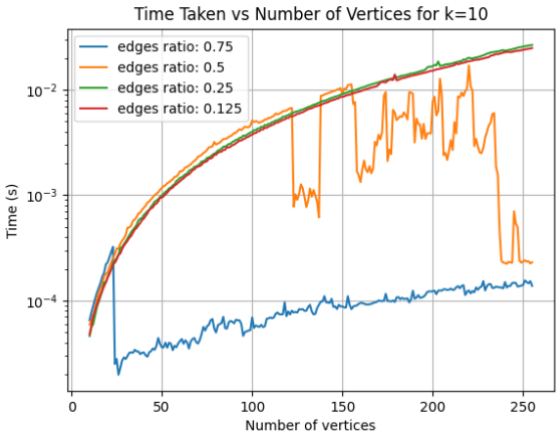

Fig. 12 - Simple Greedy: Tempo de execução por grafo com k=10

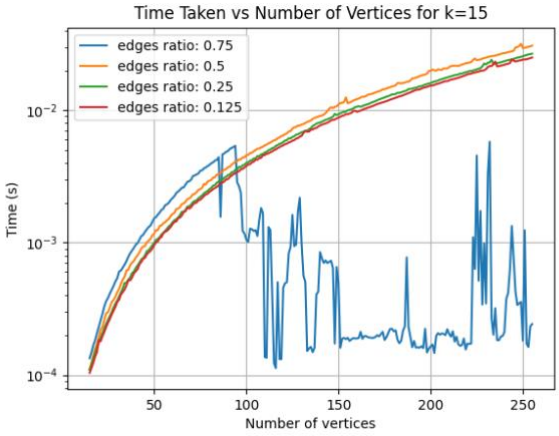

Fig. 13 - Simple Greedy: Tempo de execução por grafo com k=15

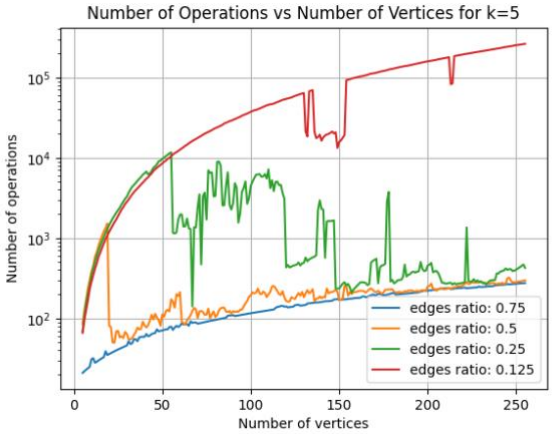

Fig. 14 - Simple Greedy: Número de operações básicas por grafo com k=5

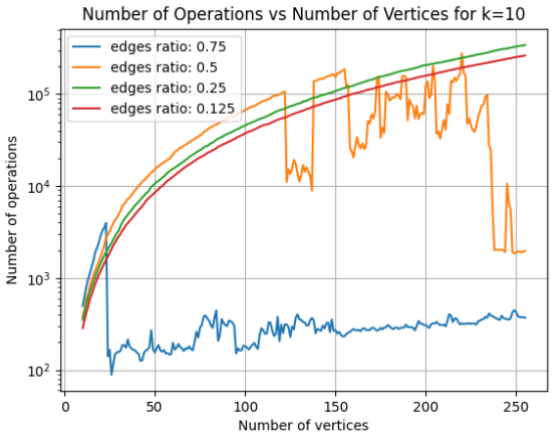

Fig. 15 - Simple Greedy: Número de operações básicas por grafo com k=10

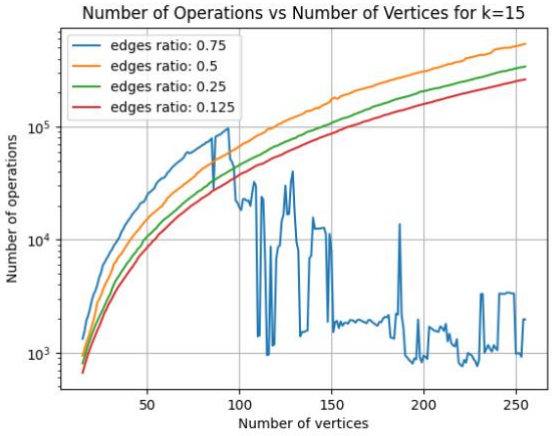

Fig. 16 - Simple Greedy: Número de operações básicas por grafo com k=15

Os gráficos, apresentados em escala logarítmica como anteriormente, mostram que a utilização de um algoritmo que ordena os vértices por grau antes de iniciar a busca torna o processo significativamente mais rápido. Com essa otimização, é evidente que o tempo de execução reduz drasticamente em comparação com a pesquisa exaustiva, especialmente para grafos de maior complexidade. A relevância do tamanho do clique diminui consideravelmente, uma vez que o algoritmo consegue encontrar cliques de forma mais eficiente, reduzindo a influência do tamanho do clique sobre o tempo de execução.

Ainda assim, a densidade de arestas continua a ser um fator muito relevante. Grafos com alta densidade de arestas facilitam a identificação de cliques, mantendo o tempo de execução mais baixo. Observa-se novamente o fenômeno de grandes quedas de tempo até a primeira deteção de cliques, quando o número de vértices aumenta. Este comportamento reflete a eficiência do algoritmo em explorar as conexões mais densas de forma prioritária, encontrando cliques mais rapidamente.

Além disso, o aumento no número de vértices dos grafos que agora podem ser testados com essa abordagem evidencia a eficácia da ordenação prévia dos vértices por grau. Isso possibilita ao algoritmo explorar grafos maiores, ampliando a escala da análise e permitindo a verificação de cliques em grafos que seriam impraticáveis de testar com a pesquisa exaustiva tradicional.

V. PESQUISA VORAZ – GREEDY INTERSECTION

O segundo algoritmo, chamado Greedy Intersection, adota uma abordagem mais refinada para a construção do clique. Ele também organiza os vértices por grau e inicia a formação do clique a partir de um vértice de maior grau. Em seguida, usa uma técnica de interseção de vizinhança para escolher o próximo vértice a ser adicionado ao clique. Cada vértice é selecionado com base numa maximização da interseção dos vizinhos do vértice com o conjunto de vértices no clique parcial, o que ajuda a priorizar vértices mais conectados e potencialmente acelera a formação de cliques densos.

Para colocar em prova este conceito foi desenvolvido código a partir do pseudo-código seguinte:

```
FUNÇÃO greedy_intersection_clique_search(grafo,
tamanho_clique):
  operações_realizadas ← 0
  soluções_testadas ← 0
  lista_nodes_ordenada ← lista de nodes do grafo
ordenada por grau em ordem decrescente

  PARA cada node_inicial em lista_nodes_ordenada:
        clique_atual ← conjunto contendo apenas
node_inicial
    ENQUANTO tamanho de clique_atual < tamanho_clique:
      candidatos ← lista de nodes no grafo que NÃO
estão em clique_atual E estão conectados a TODOS os
nodes em clique_atual

      SE candidatos estiver vazio:
        PARAR o loop ENQUANTO

      próximo_node ← node em candidatos que maximiza a
interseção de vizinhos com clique_atual
```

```
      ADICIONAR próximo_node a clique_atual
      operações_realizadas ← operações_realizadas + 1
      soluções_testadas ← soluções_testadas + 1

    SE tamanho de clique_atual = tamanho_clique:
      RETORNAR clique_atual como lista,
operações_realizadas, soluções_testadas

  RETORNAR Nulo, operações_realizadas, soluções_testadas
```

A. Análise Formal

A complexidade deste algoritmo é ligeiramente diferente. Apesar de também iniciar com a ordenação dos vértices por grau, com complexidade $O(n \log n)$, para cada vértice inicial, o algoritmo tenta expandir o clique usando uma estratégia baseada na interseção de vizinhos.

Em cada iteração de expansão do clique, o algoritmo verifica todos os candidatos potenciais que estão conectados ao clique atual. A seleção do vértice que possui a maior interseção de vizinhos com o clique leva $O(n)$ no pior caso. Esta seleção de candidatos continua até que o clique atinja o tamanho k.

Portanto, a complexidade para a construção de um clique a partir de cada vértice inicial é aproximadamente $O(n \cdot k)$, sendo necessário, em cada iteração, verificar a conectividade e calcular as interseções. A complexidade total do algoritmo é $O(n \log n + n^2 \cdot k)$.

No melhor caso, o algoritmo para assim que encontra um clique de tamanho k, o que pode ocorrer com poucas operações. No entanto, no caso médio e no pior caso, o algoritmo testa múltiplos vértices e realiza várias operações de interseção, especialmente em grafos grandes ou esparsos. Em termos de complexidade assintótica, o algoritmo mantém-se em $O(n \log n + n^2 \cdot k)$, mas o seu desempenho prático pode variar significativamente com a estrutura do grafo.

B. Análise Experimental

Na análise desta variação, foi utilizado o mesmo método de análise experimental do algoritmo anterior.

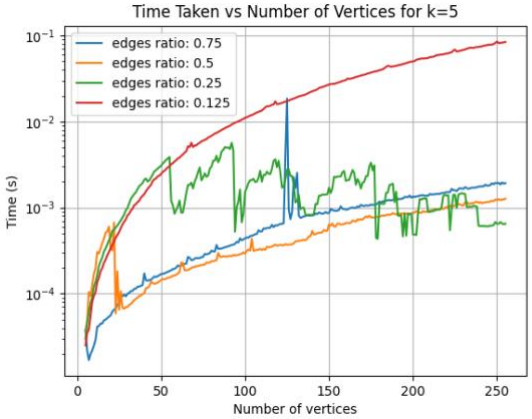

Fig. 17 - Intersection Greedy: Tempo de execução por grafo com k=5

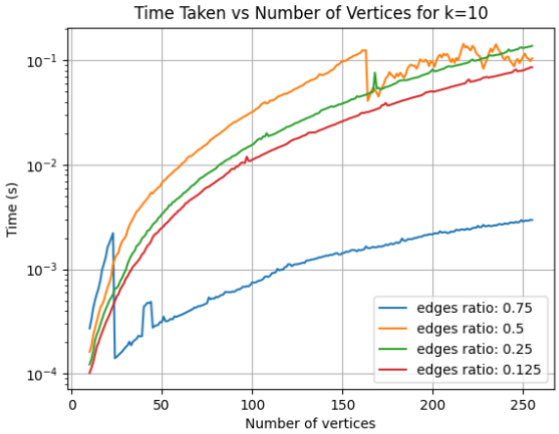

Fig. 18 - Intersection Greedy: Tempo de execução por grafo com k=10

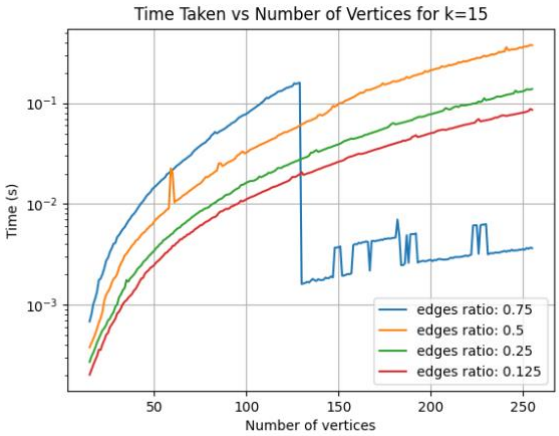

Fig. 19 - Intersection Greedy: Tempo de execução por grafo com k=15

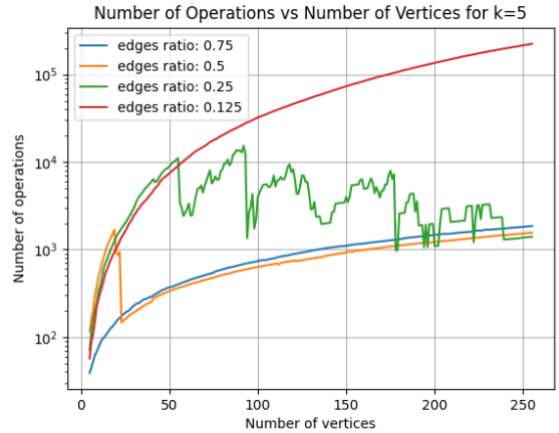

Fig. 20 - Intersection Greedy: Número de operações básicas por grafo com k=5

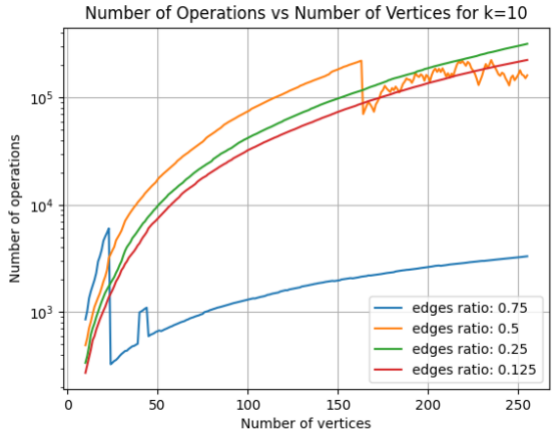

Fig. 21 - Intersection Greedy: Número de operações básicas por grafo com k=10

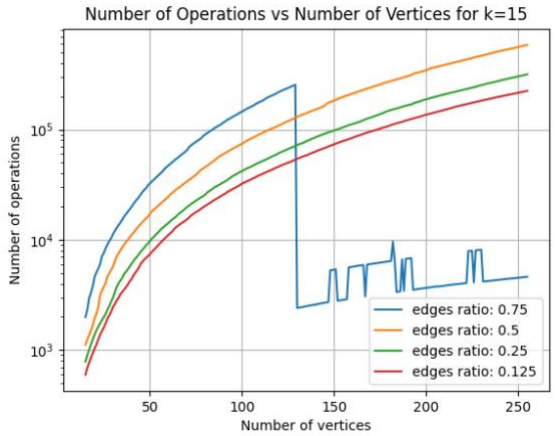

Fig. 22 - Intersection Greedy: Número de operações básicas por grafo com k=15

Nos gráficos em escala logarítmica, as conclusões para o este algoritmo são similares às do algoritmo anterior, apesar de este método apresentar um ligeiro aumento no tempo de execução. Isto ocorre porque utiliza uma técnica mais complexa de interseção de vizinhança (como já explicado) para selecionar o próximo vértice a ser adicionado ao clique, o que, embora, teoricamente, aumente a precisão na construção do clique, acrescenta um custo adicional de processamento.

Ainda assim, este algoritmo continua a ser muito mais eficiente que a pesquisa exaustiva, demonstrando uma redução substancial no tempo de execução. Neste caso, o tamanho do clique torna-se novamente menos relevante para o desempenho do algoritmo, uma vez que a interseção de vizinhança ajuda a identificar vértices mais conectados de forma mais eficiente. No entanto, a densidade de arestas do grafo permanece um fator crítico. Grafos com maior densidade facilitam a identificação de cliques, resultando em tempos de execução mais baixos.

Observa-se novamente o fenômeno inicial de grandes quedas de tempo à medida que o número de vértices aumenta, momento em que a formação de cliques se torna mais provável. Com essa abordagem, o algoritmo também é capaz de testar grafos com um número maior de vértices em relação à pesquisa exaustiva, demonstrando a vantagem da técnica de ordenação por grau seguida da interseção de vizinhança. Em resumo, o este algoritmo oferece uma boa combinação de precisão e eficiência, mesmo que apresente um leve aumento de tempo em comparação com a versão anterior mais simples.

VI. EXPLORAÇÃO DE RESULTADOS

A. Visão Geral

A precisão dos resultados obtidos pela pesquisa voraz foi validada comparando-os com os resultados obtidos por meio da pesquisa exaustiva. Cabe ressaltar que a comparação foi realizada apenas com os casos cujos resultados foram possíveis de serem calculados pela pesquisa exaustiva. A precisão do método Simple Greedy foi de 100%, enquanto a precisão do método Intersection Greedy foi de 99,76%.

Não foram encontrados benchmarks específicos na internet para os métodos utilizados nesta pesquisa. Foi apenas possível encontrar benchmarks relacionados ao problema de encontrar o clique máximo, que, embora similar, não aborda diretamente este caso de estudo.

B. Outros Resultados

Para complementar a análise, serão apresentados, para o algoritmo Simple Greedy e Greedy Intersection, novos resultados com grafos de maior dimensão (10000, 20000 e 30000 vértices) e uma densidade de arestas de 25%. Esta configuração foi escolhida para demonstrar a eficiência dos algoritmos ao lidar com grafos grandes e relativamente esparsos, onde a busca por cliques é mais desafiadora devido à menor conectividade entre os vértices. As tabelas que acompanham esta análise ilustrarão como os algoritmos se comportam em cenários mais complexos, reforçando a viabilidade das suas aplicações em grafos de grande escala e baixa densidade.

- **Simple Greedy**

10000 vértices

	5	6	7	8	9	10	15
Resultado	Sim	Sim	Sim	Sim	Sim	Sim	Null
Tempo(s)	0.007	0.006	0.006	0.0	0.6	23	80
Operações	10968	11430	18024	53560	5186434	215466925	761960701

20000 vértices

	5	6	7	8	9	10	15
Resultado	Sim	Sim	Sim	Sim	Sim	Sim	Null
Tempo(s)	0.02	0.02	0.02	0.03	0.3	7.7	391
Operações	20879	22457	23591	79095	1247475	58177285	3227651433

30000 vértices

	5	6	7	8	9	10	15
Resultado	Sim	Sim	Sim	Sim	Sim	Sim	Não
Tempo	0.6s	0.05s	0.04s	0.2s	0.15s	15s	>16h
Operações	32257	45295	48088	174536	261107	20204732	?

- **Greedy Intersection**

10000 vértices

	5	6	7	8	9	10	15
Resultado	Sim	Sim	Sim	Sim	Sim	Null	Null
Tempo	0.2s	0.2s	0.2s	3s	8s	2291s	2292s
Operações	53543	63557	73560	925498	2415165	744746643	744746643

20000 vértices

	5	6	7	8	9	10	15
Resultado	Sim	Sim	SIm	SIm	Sim	Sim	Não
Tempo	1s	1s	1s	4s	4s	32s	>16h
Operações	106914	126939	146942	460907	647879	4995020	?

30000 vértices

	5	6	7	8	9	10	15
Resultado	Sim	Sim	Sim	Sim	Sim	Sim	Não
Tempo	3s	4s	3s	5s	26s	39s	>16h
Operações	160305	190336	220340	470784	2163043	3414525	?

Os resultados obtidos mostram que ambos mantêm um desempenho eficiente, mesmo em cenários desafiadores com grafos grandes e esparsos. No entanto, os dados evidenciam diferenças significativas entre os dois algoritmos em termos de tempo de execução e operações realizadas.

O Simple Greedy demonstrou ser mais eficiente em termos de tempo, conseguindo encontrar cliques de tamanho 5 a 10 em grafos de até 30.000 vértices em tempos que variam de milissegundos a segundos. Para cliques maiores, como de tamanho 15, o algoritmo começa a enfrentar limitações significativas, com tempos de execução muito maiores e até falha a encontrar uma das soluções.

Por outro lado, o Greedy Intersection apresentou tempos de execução consistentemente mais elevados do que o Simple Greedy, devido à complexidade adicional da interseção de vizinhança. No entanto, conseguiu encontrar soluções para cliques de tamanhos 5 a 10 em tempos que variam de segundos a algumas dezenas de segundos, mesmo para grafos de 30.000 vértices. No entanto, para cliques maiores, o algoritmo enfrenta limitações significativas, apresentando tempos de execução muito mais elevados em certos casos e, ocasionalmente, falhando em encontrar soluções onde o Simple Greedy teve sucesso.

Ambos os algoritmos se mostram eficazes para cliques de tamanho moderado (até 10) em grafos de grande escala e baixa densidade. No entanto, o Simple Greedy destaca-se pela simplicidade e pelo melhor desempenho prático em termos de tempo de execução e precisão, sendo preferível para aplicações onde o objetivo é equilibrar eficiência e resultados satisfatórios. Já o Greedy Intersection, embora introduza uma técnica mais sofisticada, não apresentou vantagens significativas em relação ao Simple Greedy e, em algumas situações, mostrou-se menos eficiente. Assim, para grafos grandes e esparsos, o Simple Greedy continua a ser a escolha mais prática e robusta.

Não foram gerados grafos maiores, pois as limitações computacionais tornaram inviável a sua criação e processamento.

VII. CONCLUSÃO

Ao longo da análise e implementação de diferentes algoritmos para encontrar cliques de um tamanho especificado em grafos, diversas abordagens foram exploradas para otimizar a eficiência da solução, considerando os requisitos de precisão e desempenho de cada método. Cada algoritmo apresentou vantagens e limitações distintas em termos de custo computacional e complexidade.

O algoritmo de Pesquisa Exaustiva demonstrou ser o único capaz de garantir a descoberta exata de um clique do tamanho desejado, tornando-o uma opção precisa, mas custosa em termos de tempo e recursos computacionais. Com uma complexidade de $O\left(\binom{n}{k} \cdot k^2\right)$, a sua aplicabilidade fica limitada a grafos menores, já que o aumento do número de vértices, diminuição da densidade de arestas e o aumento do tamanho de clique torna o seu uso impraticável para instâncias maiores.

Por outro lado, os algoritmos vorazes, como o Simple Greedy e o Greedy Intersection, apresentaram uma complexidade muito menor — $O(n \log n + n \cdot k)$ para o Simple Greedy e $O(n \log n + n^2 \cdot k)$ para o Greedy Intersection. Com estas complexidades, estes algoritmos conseguem lidar com grafos substancialmente maiores de forma eficiente, sacrificando uma parte da precisão em troca de soluções mais rápidas, mas ainda assim satisfatórias para a maioria dos casos.

O Simple Greedy mostrou-se bastante eficiente ao ordenar os vértices pelo grau e, a partir de cada vértice de maior grau, tentar construir um clique adicionando iterativamente vértices conectados a todos os vértices do clique parcial. Este método tem a vantagem de ser rápido e, embora a sua precisão tenha sido de 100%, apenas foi comparado com os valores que a pesquisa exaustiva conseguiu calcular, podendo sofrer alterações para os grafos que faltavam.

O Greedy Intersection teoricamente aprimora a estratégia do Simple Greedy ao utilizar uma técnica de interseção de vizinhança, que visa maximizar a seleção de vértices altamente conectados ao clique parcial, conferindo ao algoritmo uma eficiência potencial em grafos densos. Esta abordagem, embora adicione um custo extra devido às operações de interseção, permite, em teoria, uma identificação mais rápida de cliques de alta conectividade. No entanto, nos testes práticos, esta técnica, apesar de demonstrar uma boa velocidade e eficácia, não atingiu o mesmo nível de precisão do Simple Greedy. Com base nas

métricas coletadas, concluímos que, na prática, o Greedy Intersection não superou as expectativas teóricas, e o Simple Greedy revelou-se uma escolha preferível para atender ao equilíbrio entre precisão e desempenho.

Além disso, a ordenação dos vértices pelo grau antes de iniciar a construção dos cliques provou ser uma otimização eficiente para ambos os algoritmos vorazes, diminuindo o tempo de execução e melhorando a capacidade de formar cliques desejados. Esta melhoria, contudo, depende da estrutura do grafo: em grafos esparsos ou muito pequenos, o tempo de execução ainda é afetado pela probabilidade reduzida de encontrar cliques de tamanho desejado logo no início, o que obriga os algoritmos a explorar combinações adicionais até encontrar uma solução, levando a um aumento nas operações e soluções testadas.

Embora essas otimizações tenham mostrado impacto positivo, é essencial avaliar o custo-benefício de cada tentativa de melhoria, especialmente em cenários onde o ganho em precisão não compensa o aumento do tempo computacional. No geral, os algoritmos vorazes, especialmente com ajustes como a ordenação por grau e a interseção de vizinhança, oferecem uma solução eficiente e prática, conciliando velocidade e precisão para a maioria dos contextos em que uma precisão absoluta não é obrigatória.

REFERÊNCIAS

[1] Jain, Sandeep. “Find all cliques of size K in an undirected graph.” *GeeksforGeeks*, 20 February 2023, https://www.geeksforgeeks.org/find-all-cliques-of-size-k-in-an-undirected-graph/.

[2] “Clique problem.” *Wikipedia*, https://en.wikipedia.org/wiki/Clique_problem.

[3] *An Algorithm to Discover the k-Clique Cover in Networks*, https://repositorio.uac.pt/bitstream/10400.3/2148/1/progress%20in%20AI.pdf.

[4] https://github.com/OnurArdaB/k-clique