Clique of size k – Análise de Algoritmos Aleatórios para Problemas de Decisão

André Almeida Oliveira

Resumo **- O problema de encontrar um clique de tamanho *k* num grafo consiste em identificar um subconjunto de vértices onde todos os vértices estão conectados entre si, com tamanho 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 trabalho, são exploradas diferentes abordagens probabilísticas, incluindo algoritmos baseados em Monte Carlo e Las Vegas. Estas integram geração aleatória de soluções com heurísticas, otimizando a busca por cliques em grafos grandes. Foi realizada uma análise formal e experimental para avaliar as abordagens, comparando-as em termos de eficiência computacional e precisão dos resultados.**

Abstract **- The problem of finding a k-sized clique in a graph involves identifying a subset of vertices where all vertices are mutually connected, with a size exactly equal to k. This problem holds significant relevance in fields such as social networks, bioinformatics, and network analysis, where identifying densely connected subgroups can provide valuable insights. In this work, different probabilistic approaches are explored, including algorithms based on Monte Carlo and Las Vegas methods. These approaches combine random solution generation with heuristics, optimizing the search for cliques in large graphs. A formal and experimental analysis was conducted to evaluate the approaches, comparing them in terms of computational efficiency and result accuracy.**

Keywords - Clique Problem, Randomized Algorithms, 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 deste 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.

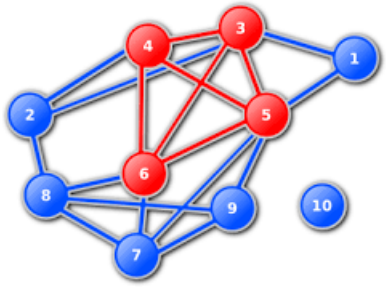

Fig. 1 - Exemplo visual de um clique de tamanho 4 (vértices 3, 4, 5 e 6) num grafo

B. Desafios e Abordagens

Um algoritmo de pesquisa aleatória é uma técnica eficiente e não determinística que explora o espaço de soluções de forma limitada, através da geração aleatória de possíveis cliques num grafo. Este tipo de abordagem é especialmente útil em problemas combinatórios complexos, onde o espaço de busca cresce exponencialmente com o tamanho da entrada. Neste caso específico, implementaram-se diversos algoritmos probabilísticos que selecionam subconjuntos de vértices de forma aleatória ou com heurísticas, para encontrar cliques de tamanho *k* num grafo.

Apesar te terem sido implementadas cinco abordagens, apenas serão detalhadas as três abaixo. Os resultados de todas as abordagens estarão disponíveis no *excel* na pasta *results*.

- **Método de *Las Vegas***

Esta abordagem garante soluções corretas com custo de maior tempo de execução, quando comparado aos métodos probabilísticos simples, pois escolhe subconjuntos de vértices aleatórios para ver se forma um clique.

- **Método de *Monte Carlo***

Esta abordagem utiliza o método de *Monte Carlo*. O processo começa selecionando um vértice aleatório e, a partir dele, expande iterativamente o subconjunto do clique escolhendo vértices vizinhos.

- ***Randomized Heuristic***

Esta abordagem combina geração aleatória de subconjuntos com heurísticas para encontrar cliques de tamanho num grafo. Inicialmente, os vértices são ordenados com base numa heurística que prioriza aqueles com maior número de conexões, formando um conjunto candidato mais promissor. Em seguida, subconjuntos de tamanho *k* são amostrados aleatoriamente desse conjunto reduzido.

II. MÉTODO DE LAS VEGAS

O algoritmo *Las Vegas Clique* é uma abordagem probabilística para encontrar um clique de tamanho *k* num grafo, utilizando aleatoriedade para reduzir o espaço de busca. Em vez de explorar todas as combinações possíveis, o algoritmo seleciona candidatos aleatoriamente, garantindo que a solução final seja válida. Ele verifica inicialmente se o grafo possui vértices suficientes para formar o clique desejado. Em cada iteração, os vértices são baralhados, e o clique é construído incrementalmente, adicionando vértices conectados aos já presentes no subconjunto. Caso um clique de tamanho *k* seja encontrado, o algoritmo retorna a solução, caso contrário, interrompe após um número limite de operações, tentativas ou tempo.

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

```
FUNÇÃO las_vegas_clique(grafo, tamanho_clique,
max_tentativas):
 lista_nodes ← lista de nodes do grafo
 SE número de vértices < tamanho_clique:
  RETORNAR Nulo, 0, 0

 operações_realizadas ← 0
 soluções_testadas ← 0

 PARA cada tentativa em 1 ATÉ max_tentativas:
  SE operações_realizadas > limite_operacional:
   INTERROMPER

  subset ← []
  candidatos ← lista_nodes embaralhada aleatoriamente

  PARA cada node em candidatos:
   SE operações_realizadas > limite_operacional:
    INTERROMPER
   SE tamanho(subset) < tamanho_clique E TODOS os
vértices de subset estão conectados a node:
    adicionar node a subset
    operações_realizadas ← operações_realizadas +
(tamanho(subset) - 1)

  soluções_testadas ← soluções_testadas + 1

  SE tamanho(subset) = tamanho_clique:
   RETORNAR subset, operações_realizadas,
soluções_testadas

 RETORNAR Nulo, operações_realizadas, soluções_testadas
```

A. Análise Formal

Este algoritmo apresenta uma complexidade que depende principalmente do número de vértices *n* e do tamanho do clique *k*. A inicialização do algoritmo, que inclui a criação de uma lista de vértices e a verificação da viabilidade de um clique, possui complexidade $O(n)$. Em cada tentativa, a lista de vértices é baralhada $O(n)$ e percorre-se os vértices para construir o clique incrementalmente. Cada iteração verifica a conectividade entre os vértices, o que tem complexidade $O(k)$, resultando em $O(n \cdot k)$ por tentativa.

Considerando todas as tentativas, a complexidade total no pior caso é $O(num\_trials \cdot n \cdot k)$. No melhor caso, o algoritmo encontra um clique nas primeiras tentativas, encerrando a execução com um custo proporcional ao número de operações realizadas até esse ponto.

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 1000, 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, apesar de neste relatório apenas haver gráficos para os valores 5, 10 e 15. Foram utilizadas três métricas para a avaliação: tempo de execução, número de operações e número de soluções testadas.

Cada teste teve um limite dinâmico de tempo e operações que podia usar, caso contrário, o teste parava e avançava para o próximo, assumindo que não foi capaz de resolver o problema num tempo e numa quantidade de operações aceitável para o número de vértices que o grafo tem.

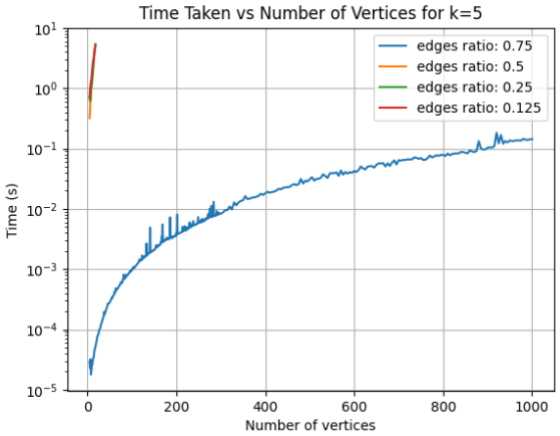

Fig. 2 -Las Vegas: Tempo de execução por grafo com k=5

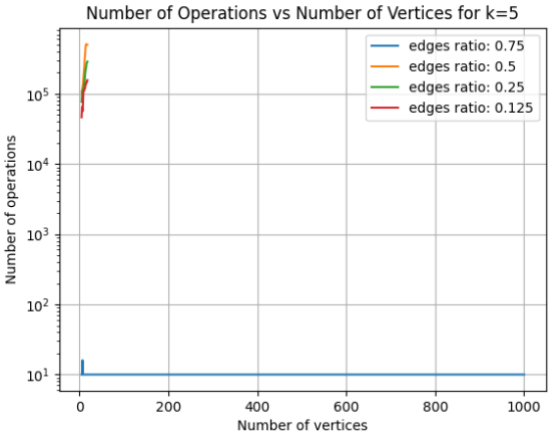

Fig. 5 - Las Vegas: Número de operações básicas por grafo com k=5

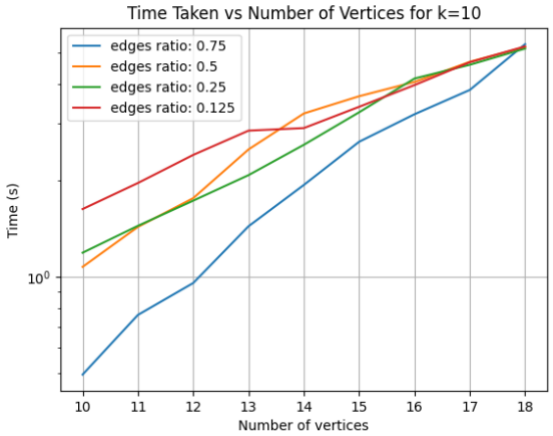

Fig. 3 - Las Vegas: Tempo de execução por grafo com k=10

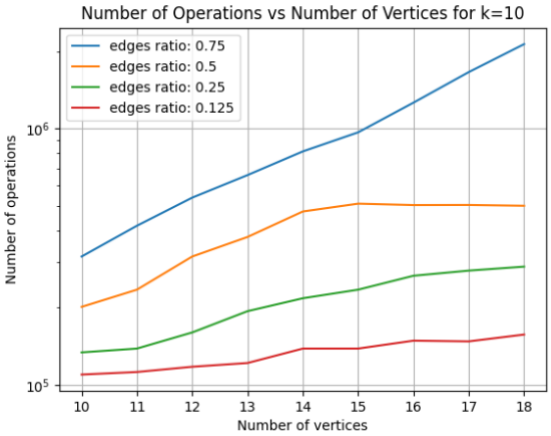

Fig. 6 - Las Vegas: Número de operações básicas por grafo com k=10

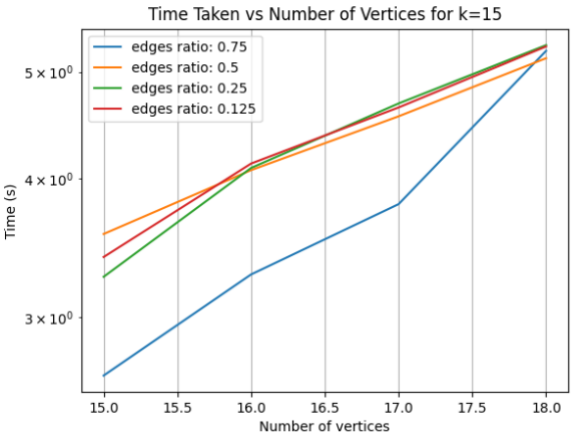

Fig. 4 - Las Vegas: Tempo de execução por grafo com k=15

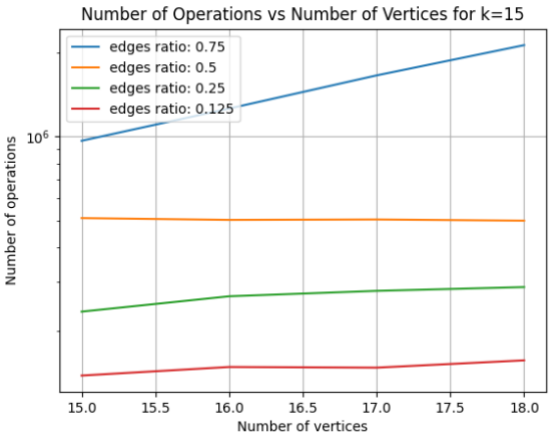

Fig. 7 - Las Vegas: Número de operações básicas por grafo com k=15

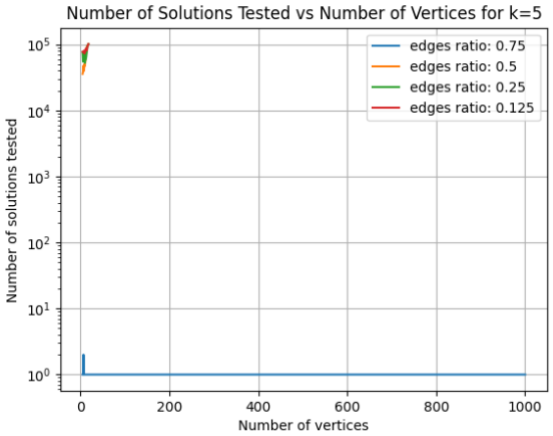

Fig. 8 - Las Vegas: Número de soluções testadas por grafo com k=5

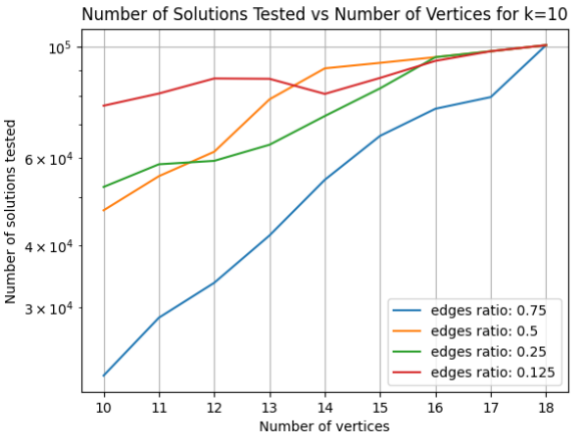

Fig. 9 - Las Vegas: Número de soluções testadas por grafo com k=10

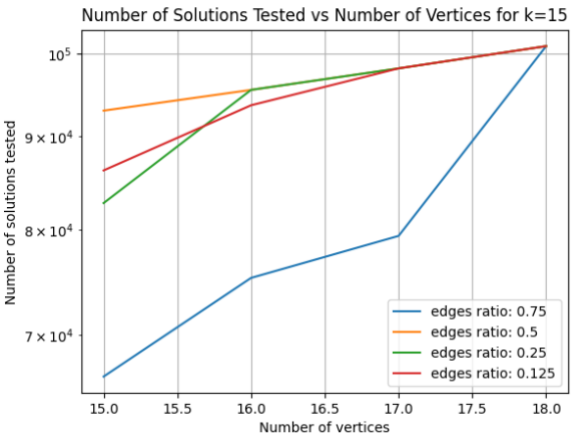

Fig. 10 - Las Vegas: Número de soluções testadas por grafo com k=15

O método de *Las Vegas* apresentou um desempenho interessante, mas não mostrou melhorias óbvias em relação à pesquisa exaustiva em diversas métricas avaliadas. Embora utilize aleatoriedade para reduzir o espaço de busca, o algoritmo não conseguiu superar a pesquisa exaustiva, especialmente para valores maiores de k, como $k = 10$ e $k = 15$. Nestes casos, este método até testou menos vértices e encontrou menos soluções em comparação com a abordagem exaustiva, o que demonstra limitações na sua capacidade de exploração eficiente do espaço de soluções.

Adicionalmente, embora o número de operações realizadas e o tempo de execução tenham sido moderados em algumas situações, a densidade do grafo mostrou um impacto significativo no desempenho. Grafos densos facilitaram a busca por soluções, mas não compensaram a inconsistência do método em grafos esparsos ou para valores elevados de k. Assim, apesar da tentativa de aproveitar a aleatoriedade, este método, neste cenário, não conseguiu demonstrar vantagens claras sobre a pesquisa exaustiva, sendo menos eficiente em casos onde k e o número de vértices são maiores. Melhorias futuras poderiam explorar a integração de heurísticas ou técnicas de otimização que ajudem a superar estas limitações.

III. MÉTODO DE MONTE CARLO

O algoritmo *Monte Carlo with Heuristic Clique* é uma abordagem probabilística, integrando heurísticas para otimizar o processo de seleção de vértices. Este método utiliza uma combinação de geração aleatória de soluções e priorização baseada em conectividade local, o que reduz o espaço de busca e aumenta a eficiência em comparação com abordagens determinísticas exaustivas.

O algoritmo começa verificando se o grafo possui vértices suficientes para formar um clique de tamanho *k*. Caso contrário, retorna imediatamente um valor nulo. Em seguida, realiza até *num_trials* iterações, onde cada iteração segue os seguintes passos: um vértice inicial é escolhido aleatoriamente, e o conjunto de vizinhos desse vértice é identificado. A partir desse ponto, o clique é construído incrementalmente, utilizando uma estratégia que prioriza a escolha de vértices vizinhos com maior grau de conectividade. Esses vizinhos são ordenados por ordem decrescente de grau, e um vértice é selecionado aleatoriamente da fração superior do conjunto ordenado. O subconjunto do clique é atualizado iterativamente para conter apenas os vértices que continuam conectados a todos os outros no clique parcial.

O processo é interrompido caso o número de operações ou o tempo ultrapasse o limite predefinido, o conjunto de vizinhos fique vazio, um clique de tamanho *k* seja formado ou chegue ao número máximo de tentativas.

Para validar o conceito, foi desenvolvido o seguinte pseudo-código:

```
FUNÇÃO monte_carlo_with_heuristic_clique(grafo,
tamanho_clique, max_tentativas):
  lista_nodes ← lista de nodes do grafo
  SE número de vértices < tamanho_clique:
    RETORNAR Nulo, 0, 0

  soluções_testadas ← conjunto vazio
  operações_realizadas ← 0

  PARA cada tentativa em 1 ATÉ max_tentativas:
    subset ← []
    node ← escolher aleatoriamente um vértice em
lista_nodes
    adicionar node a subset
    neighbors ← conjunto de vizinhos de node
    operações_realizadas ← operações_realizadas + 1

    ENQUANTO tamanho(subset) < tamanho_clique:
      SE neighbors estiver vazio OU operações_realizadas
> limite_operacional:
        INTERROMPER
      neighbors ← ordenar neighbors por grau de
conectividade (decrescente)
      candidate ← escolher aleatoriamente de uma fração
superior de neighbors
      adicionar candidate a subset
      neighbors ← interseção de neighbors e conjunto de
vizinhos de candidate
      operações_realizadas ← operações_realizadas + 1

    subset_id ← ordenar subset e converter para
identificador único
    operações_realizadas ← operações_realizadas +
tamanho(subset)
    SE subset_id estiver em soluções_testadas:
      CONTINUAR
    adicionar subset_id a soluções_testadas
    operações_realizadas ← operações_realizadas + 1

    SE tamanho(subset) = tamanho_clique E subset for um
clique:
      operações_realizadas ← operações_realizadas +
combinações do subset
      RETORNAR subset, operações_realizadas,
tamanho(soluções_testadas)

  RETORNAR Nulo, operações_realizadas,
tamanho(soluções_testadas)
```

A. Análise Formal

Este algoritmo utiliza uma abordagem probabilística com heurísticas, baseando-se em tentativas aleatórias com priorização de vértices. A sua complexidade depende do número de vértices *n*, do tamanho do clique *k*, do número de tentativas e da densidade do grafo. Inicialmente, o algoritmo verifica se o grafo possui vértices suficientes para formar um clique de tamanho *k*, com complexidade $O(n)$.

Em cada tentativa, um vértice inicial é escolhido aleatoriamente ($O(1)$) e o conjunto de vizinhos é obtido ($O(d)$, onde *d* é o grau do vértice selecionado). Os vizinhos são ordenados por grau de conectividade, e os vértices são selecionados com base nesta ordenação, priorizando aqueles com maior grau. O clique é construído incrementalmente, com custo $O(k \cdot d)$ por tentativa, considerando as operações de interseção e ordenação realizadas durante o processo. No pior caso, para tentativas, a complexidade total é $O(num\_trials \cdot k \cdot d)$. Em grafos densos *d*, pode ser proporcional a *n*, enquanto em grafos esparsos, *d* é reduzido, diminuindo o custo.

No melhor caso, um clique é identificado rapidamente, reduzindo significativamente o número de operações realizadas.

B. Análise Experimental

Na análise deste método, foi utilizado o mesmo método de análise experimental aplicado ao método de *Las Vegas*. No entanto, apesar se ter sido implementado um algoritmo sem heurística e outro com heurística, apenas serão apresentados os resultados referentes à variação com heurística, dado que este demonstrou um desempenho significativamente superior. Apesar disso, os resultados completos foram devidamente guardados para consulta e comparação futura, como já dito anteriormente.

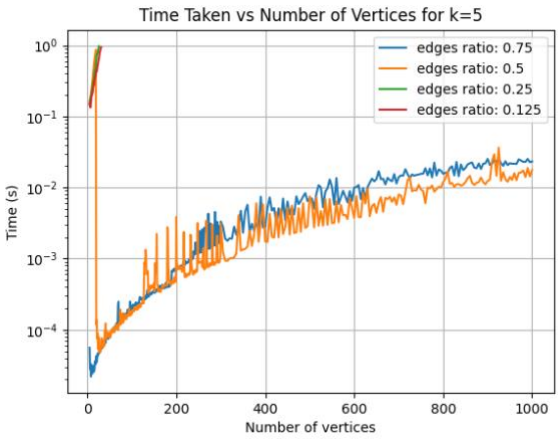

Fig. 11 – Monte Carlo: Tempo de execução por grafo com k=5

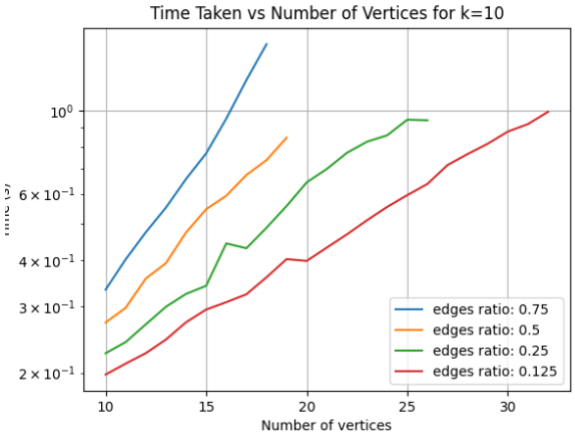

Fig. 12 – Monte Carlo: Tempo de execução por grafo com k=10

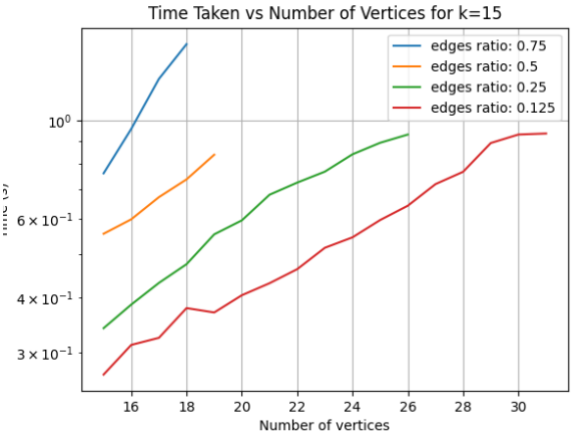

Fig. 13 –Monte Carlo: Tempo de execução por grafo com k=15

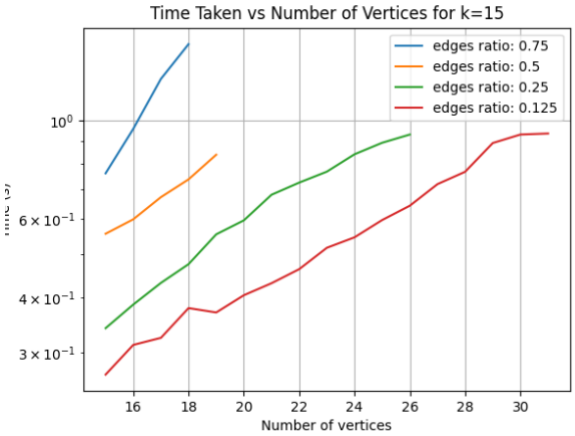

Fig. 16 – Monte Carlo: Número de operações básicas por grafo com k=15

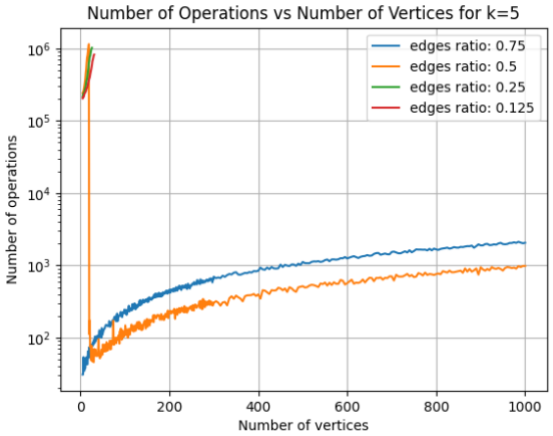

Fig. 14 – Monte Carlo: Número de operações básicas por grafo com k=5

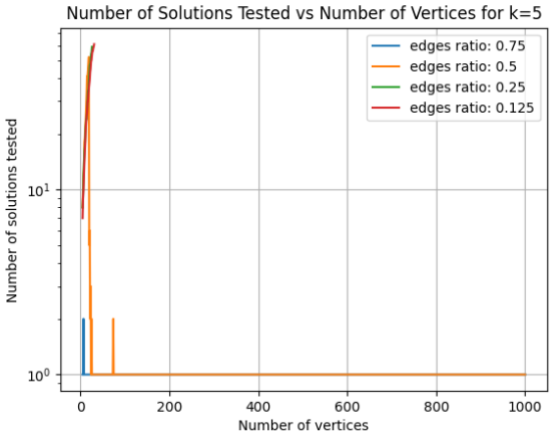

Fig. 17 – Monte Carlo: Número de soluções testadas por grafo com k=5

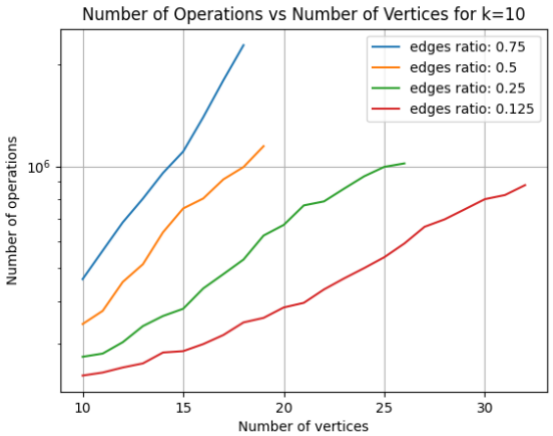

Fig. 15 – Monte Carlo: Número de operações básicas por grafo com k=10

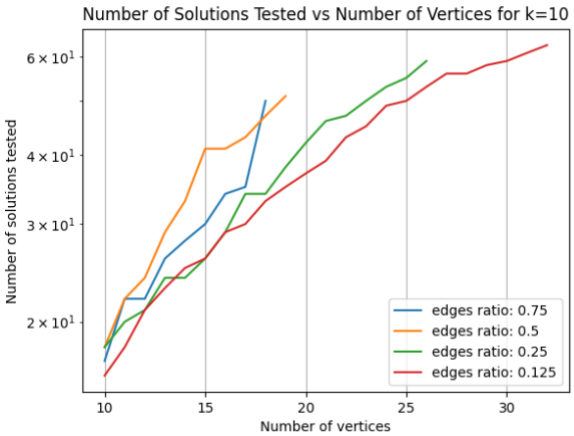

Fig. 18 – Monte Carlo: Número de soluções testadas por grafo com k=10

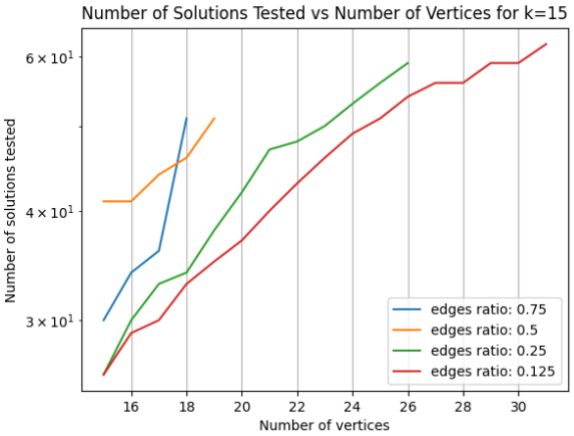

Fig. 19 – Monte Carlo: Número de soluções testadas por grafo com k=15

Este algoritmo demonstrou um desempenho significativamente superior ao algoritmo usando o método de *Las Vegas* em várias métricas avaliadas, destacando-se como uma abordagem mais robusta. Para valores menores de k, como $k = 5$, o número de operações realizadas foi menor em comparação ao de *Las Vegas*, excluindo quando a densidade do grafo é 0.75, e a utilização de heurísticas resultou em tempos de execução menores, podendo reparar que quando $k = 5$, esse tempo até é menor com uma densidade de grafo de 0.5 do que com densidade de 0.75.

Para valores maiores de k, como $k = 10$ e $k = 15$, o algoritmo com heurísticas mostrou uma melhor capacidade de explorar o espaço de soluções, testando um maior número de combinações em grafos de menor densidade e alcançando soluções válidas com maior frequência. Em contraste, o método de *Las Vegas* demonstrou limitações, tanto para grafos densos como esparsos, onde testou menos vértices e encontrou menos soluções. A abordagem com heurísticas conseguiu mitigar essas limitações, priorizando vértices mais conectados e aumentando a eficiência geral.

Em termos de operações realizadas e tempo de execução, o *Monte Carlo* com heurística apresentou uma escalabilidade mais consistente em grafos grandes. Embora o número de operações cresça com k e com o número de vértices, os gráficos indicam que esse crescimento é menos abrupto em comparação ao método de *Las Vegas*. Isso reforça a importância das heurísticas na otimização do processo, permitindo uma exploração mais direcionada e eficaz.

No geral, este algoritmo destacou-se pela sua eficiência em grafos mais esparsos e pela capacidade de lidar com valores maiores de k. Comparado ao de *Las Vegas*, mostrou-se mais eficiente, especialmente em cenários onde a conectividade local favorece a formação de cliques. No entanto, em grafos esparsos e com tamanhos de clique maiores, ambas as abordagens enfrentam desafios, ainda que as heurísticas tenham proporcionado resultados melhores. Este método demonstra que a integração de heurísticas é uma estratégia eficaz para superar as limitações observadas em algoritmos puramente aleatórios.

V. RANDOMIZED HEURISTIC

O algoritmo *Randomized Heuristic Clique* é uma abordagem probabilística que combina geração aleatória com heurísticas. Diferentemente de métodos puramente aleatórios, este algoritmo também utiliza informações sobre a estrutura do grafo para otimizar a escolha de vértices, o que o torna mais eficiente e direcionado. A principal estratégia consiste em ordenar os vértices com base no grau de conectividade, formando subconjuntos candidatos mais promissores para a construção do clique.

O algoritmo inicia verificando se o grafo possui vértices suficientes para formar um clique de tamanho k. Caso contrário, retorna imediatamente um valor nulo. De seguida, realiza até *num_trials* iterações. Durante cada iteração, os vértices do grafo são ordenados com base no seu grau de conectividade (heurística). Após a ordenação, é formada uma *pool* de candidatos, composta pelos vértices com maiores graus, representando uma fração superior da lista ordenada. A partir dessa *pool*, é selecionado aleatoriamente um subconjunto de k vértices.

Esse subconjunto é testado para verificar se é um clique válido. Caso seja, o algoritmo retorna a solução imediatamente. Para evitar redundâncias, subconjuntos já testados são armazenados num conjunto de soluções rastreadas. Caso o limite de operações ou de tempo seja atingido ou as iterações se esgotem sem encontrar um clique, o algoritmo retorna um valor nulo.

Para validar o conceito, foi desenvolvido o seguinte pseudo-código:

```
FUNÇÃO randomized_heuristic_clique(grafo,
tamanho_clique, max_tentativas):
  lista_nodes ← lista de nodes do grafo
  SE número de vértices < tamanho_clique:
    RETORNAR Nulo, 0, 0

  soluções_testadas ← conjunto vazio
  operações_realizadas ← 0

  DEFINIR heurística(node):
    RETORNAR grau do node no grafo

  FUNÇÃO gerar_candidato():
    ordenar lista_nodes por heurística (decrescente)
    operações_realizadas ← operações_realizadas +
tamanho(lista_nodes)
    candidate_pool ← fração superior da lista ordenada
    SE tamanho(candidate_pool) < tamanho_clique:
      RETORNAR Nulo
    subset ← selecionar aleatoriamente tamanho_clique
elementos de candidate_pool
    operações_realizadas ← operações_realizadas +
tamanho_clique
    RETORNAR subset

  PARA cada tentativa em 1 ATÉ max_tentativas:
    SE operações_realizadas > limite_operacional:
      INTERROMPER

    candidato ← gerar_candidato()
    SE candidato for Nulo:
      CONTINUAR
    candidate_id ← ordenar candidato e converter para
identificador único
    operações_realizadas ← operações_realizadas +
tamanho(candidato)
    SE candidate_id estiver em soluções_testadas:
      CONTINUAR
    adicionar candidate_id a soluções_testadas
    operações_realizadas ← operações_realizadas + 1
```

```
    SE candidato for um clique:
      operações_realizadas ← operações_realizadas +
combinações do candidato
      RETORNAR candidato, operações_realizadas,
tamanho(soluções_testadas)

  RETORNAR Nulo, operações_realizadas,
tamanho(soluções_testadas)
```

A. Análise Formal

A complexidade deste algoritmo depende do número de vértices *n*, do tamanho do clique *k*, do número de tentativas *num_trials* e da densidade do grafo. Inicialmente, o algoritmo verifica se o grafo possui vértices suficientes para formar um clique de tamanho *k*, com complexidade $O(n)$.

Durante cada tentativa, os vértices do grafo são ordenados com base no grau de conectividade ($O(n \cdot log\ n)$) e uma fração superior dos vértices mais conectados é selecionada para formar a *pool* de candidatos. Deste conjunto, vértices são escolhidos aleatoriamente para formar um subconjunto, com custo $O(k)$. O clique resultante é testado quanto à validade ($O(k^2)$) e, caso já tenha sido testado antes, é descartado para evitar redundância. O processo é repetido até atingir o número máximo de tentativas, atingir o limite de tempo, atingir o limite de operações ou até encontrar um clique válido.

No pior caso, para *num_trials* tentativas, a complexidade total é $O(num\_trials \cdot (n \cdot \log n + k^2))$. Em grafos densos, onde o grau médio dos vértices (*d*) é proporcional a *n*, a ordenação e a construção do clique tornam-se mais custosas, mas a probabilidade de encontrar cliques válidos aumenta devido à maior conectividade. Já em grafos esparsos, é pequeno, o que reduz o custo de processamento, mas também diminui a chance de sucesso, tornando o algoritmo menos eficiente nesses cenários.

No melhor caso, um clique é identificado rapidamente, reduzindo significativamente o número de operações realizadas.

B. Análise Experimental

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

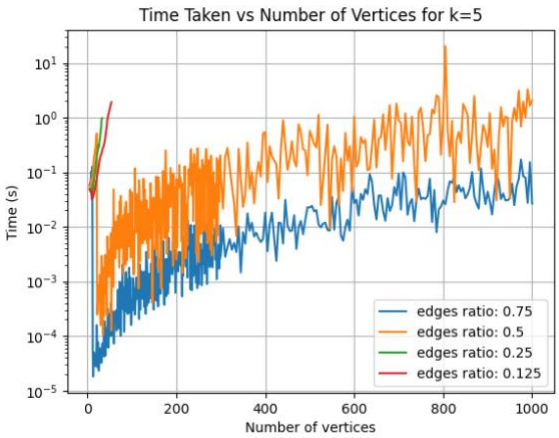

Fig. 20 – Randomized Heuristic: Tempo de execução por grafo com k=5

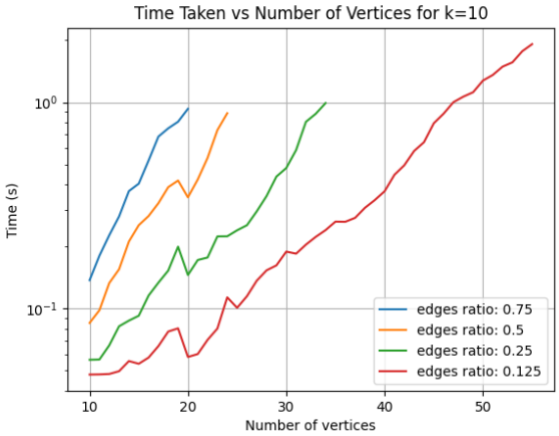

Fig. 21 - Randomized Heuristic: Tempo de execução por grafo com k=10

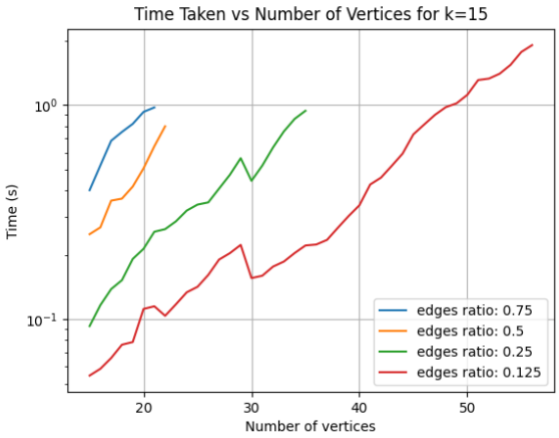

Fig. 22 - Randomized Heuristic: Tempo de execução por grafo com k=15

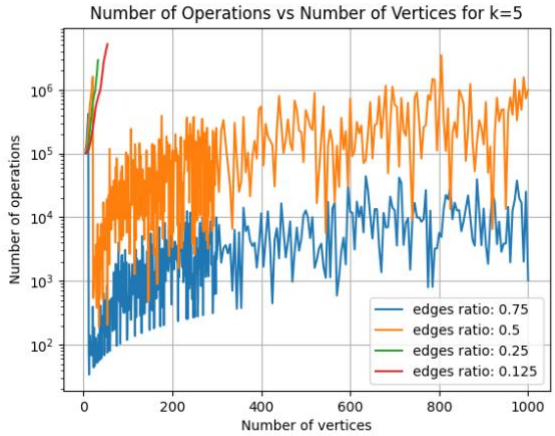

Fig. 23 – Randomized Heuristic: Número de operações básicas por grafo com k=5

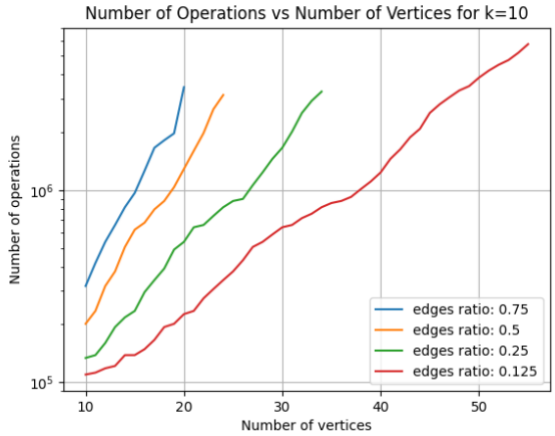

Fig. 24 - Randomized Heuristic: Número de operações básicas por grafo com k=10

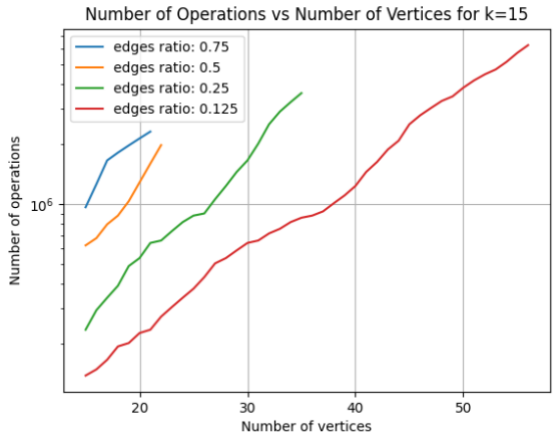

Fig. 25 - Randomized Heuristic: Número de operações básicas por grafo com k=15

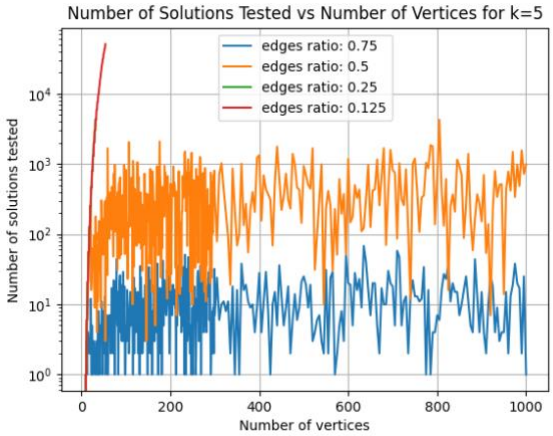

Fig. 26 - Randomized Heuristic: Número de soluções testadas por grafo com k=5

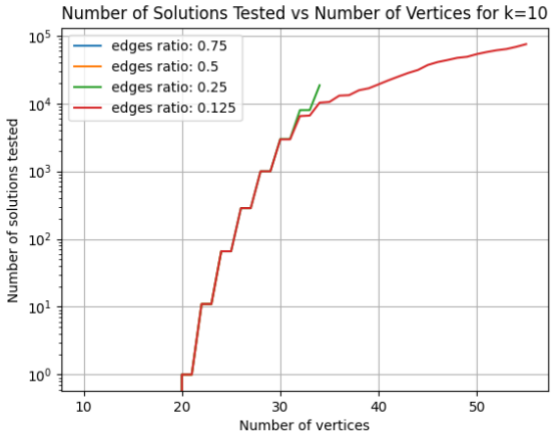

Fig. 27 - Randomized Heuristic: Número de soluções testadas por grafo com k=10

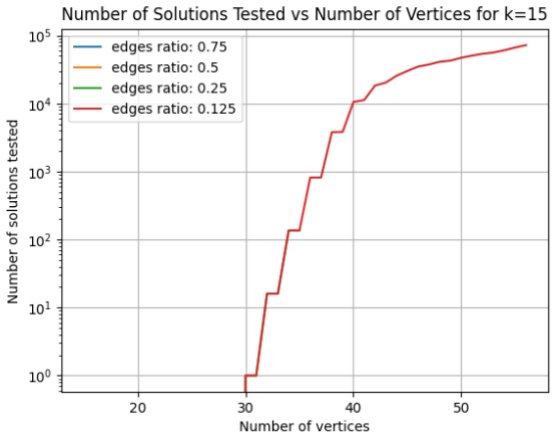

Fig. 28 – Randomized Heuristic: Número de soluções testadas por grafo com k=15

Este método demonstrou bom desempenho, permitindo reduzir o espaço de busca em relação a abordagens puramente aleatórias. Em comparação ao algoritmo utilizando o método de *Monte Carlo*, mostrou-se mais eficiente para grafos menos densos e valores elevados de *k*, mas apresentou um menor valor de precisão (mostrado no tópico seguinte). Contudo, o *Randomized Heuristic* apresentou simplicidade de implementação e um desempenho bastante competitivo.

Quando comparado ao algoritmo com o método de *Las Vegas*, foi consistentemente superior, apresentando menos operações e menos soluções testadas, especialmente para grafos maiores e valores elevados de *k*. O uso de heurística no *Randomized Heuristic* permitiu uma exploração mais direcionada do espaço de busca, superando a abordagem puramente probabilística do de *Las Vegas*.

Em resumo, este algoritmo posiciona-se como uma solução eficiente e intermediária, mais simples que o de *Monte Carlo* com heurística, mas substancialmente mais eficaz que o de *Las Vegas*. É uma opção recomendada para cenários de complexidade moderada, com resultados bastante semelhantes aos do método de *Monte Carlo*, e até melhores em alguns casos, conseguindo a busca em grafos com tamanhos maiores e menos densos. Apesar de ter menos precisão, é tão alta que continua a ser uma opção bastante válida para a resolução deste problema.

VI. EXPLORAÇÃO DE RESULTADOS

A. Visão Geral

A precisão dos resultados obtidos pelos três algoritmos analisados foi validada comparando-os com os resultados provenientes da pesquisa exaustiva. Essa comparação foi realizada apenas para os casos em que a pesquisa exaustiva conseguiu calcular uma solução. Os resultados indicaram que o método *Las Vegas* e o *Monte Carlo* com Heurística atingiram uma precisão de 100%, enquanto o Randomized Heuristic apresentou uma precisão ligeiramente inferior, de 98,87%.

Além desses três métodos, foram avaliadas outras duas abordagens: o *Monte Carlo* normal e o *Random Sampling*. O *Monte Carlo* normal segue a lógica do *Monte Carlo* com heurística, mas sem a priorização baseada na conectividade dos vértices, alcançando uma precisão de 100%. Já o *Random Sampling*, que consiste apenas em selecionar subconjuntos aleatórios de vértices e verificar se formam um clique, obteve uma precisão de 99,85%.

É importante destacar que, em nenhum momento, os algoritmos apresentam a ocorrência de falsos positivos. Assim, a medida de precisão refere-se exclusivamente aos casos em que o algoritmo afirma que não há solução, mas, na realidade, a solução existe.

B. Outros Resultados

A fim de testar as abordagens para grafos mais dispendiosos computacionalmente foi percorrido o grafo SWlargeG, com 1 milhão de vértices, para os cinco algoritmos falados anteriormente.

Concluímos que apenas o algoritmo *Monte Carlo* normal e o *Monte Carlo* com heurística foram capazes de resolver este problema, sendo que os restantes passaram o limite de tempo ou de operações.

- **Monte Carlo Normal**

	5	**6**	**7**	**8**	**9**	**10**	**15**
Resultado	Sim	Sim	Sim	Sim	Sim	Sim	Não
Tempo (s)	2	6	4	0.2	2	2	?
Operações	21	79	74	7	49	60	?
Soluções Testadas	4	12	11	1	7	10	?

- **Monte Carlo com Heurística**

	5	**6**	**7**	**8**	**9**	**10**	**15**
Resultado	Sim	Sim	Sim	Sim	Sim	Sim	Não
Tempo (s)	0.5	0.7	0,8	1,2	2,1	1,1	?
Operações	55	86	89	103	170	127	?
Soluções Testadas	1	1	1	2	3	1	?

Os resultados obtidos pelos dois métodos, *Monte Carlo* normal e *Monte Carlo* com heurística, demonstram que ambos são bastante eficientes para encontrar cliques em grafos. No entanto, o uso de heurísticas no *Monte Carlo* revelou-se uma estratégia vantajosa, especialmente no que diz respeito à redução do número de soluções testadas. Enquanto o *Monte Carlo* normal testa um maior número de subconjuntos para identificar um clique, o algoritmo com heurísticas consegue priorizar subconjuntos mais promissores, reduzindo significativamente a quantidade de soluções testadas, como observado para, por exemplo, 1 solução testada com heurística contra 12 no método normal.

Por outro lado, o *Monte Carlo* com heurística apresentou um leve aumento no número de operações em comparação com o método normal, devido ao custo adicional associado à ordenação dos vértices e à aplicação da heurística de seleção baseada em conectividade. Apesar desse aumento, o tempo de execução do método com heurísticas foi consistentemente melhor, evidenciando a sua eficiência geral. Por exemplo, para $k = 6$, o tempo no *Monte Carlo* com heurísticas foi de 0.7 segundos, enquanto no *Monte Carlo* normal foi de 6 segundos.

Portanto, conclui-se que, embora o uso de heurísticas introduza um pequeno custo adicional em termos de operações, ele é amplamente compensado pela significativa redução no número de soluções testadas e pelo melhor desempenho em tempo de execução. Essas vantagens tornam o *Monte Carlo* com heurística uma abordagem mais eficaz, especialmente para grafos maiores ou cliques maiores, onde a eficiência é crucial.

VII. CONCLUSÃO

Neste artigo, foram explorados diferentes algoritmos probabilísticos para encontrar cliques num grafo, comparando o desempenho de três abordagens: *Las Vegas*, *Monte Carlo* com heurística e *Randomized Heuristic*, além de mencionar outras duas abordagens complementares, o *Monte Carlo* normal e o *Random Sampling*. Através de análises formais e experimentais, foi possível identificar as vantagens e limitações de cada método, bem como a sua aplicabilidade em diferentes cenários.

O método de *Las Vegas* demonstrou alta precisão (100%) e eficiência em grafos densos, mas enfrentou dificuldades em grafos mais esparsos e valores maiores de *k*, apresentando tempos de execução elevados e limitações na exploração de soluções válidas. Em contrapartida, o *Monte Carlo* com heurística destacou-se pela capacidade de priorizar subconjuntos mais promissores através de técnicas de ordenação e seleção baseadas na conectividade local, resultando em tempos de execução menores e numa redução expressiva no número de soluções testadas, mantendo uma precisão de 100% mesmo em grafos maiores e mais complexos.

O *Randomized Heuristic* mostrou-se uma solução eficiente e intermediária, combinando simplicidade de implementação com desempenho competitivo. Apesar de apresentar uma precisão ligeiramente inferior (98,87%), este método foi eficaz em grafos menos densos e com tamanhos maiores de *k*, superando consistentemente o método de *Las Vegas* em termos de operações realizadas e soluções testadas.

Foi identificado um fenômeno marcante em que a pesquisa probabilística conseguiu resolver melhor problemas em grafos de menor densidade e maior número de vértices, algo inviável para algoritmos determinísticos. Ao explorar aleatoriedade e heurísticas, os métodos probabilísticos reduziram significativamente o espaço de busca, concentrando esforços em subconjuntos de vértices mais promissores. Isso permitiu superar as limitações dos métodos determinísticos, que enfrentam dificuldades em cenários de baixa conectividade devido à explosão combinatória e a padrões de busca.

Outros métodos, como o *Monte Carlo* normal e o *Random Sampling*, também apresentaram resultados promissores, com precisão de 100% e 99,85%, respetivamente. Contudo, o uso de heurísticas revelou-se essencial para otimizar o desempenho em cenários mais desafiadores, reduzindo o número de soluções testadas sem comprometer a precisão ou aumentar significativamente o número de operações realizadas.

De forma geral, conclui-se que as heurísticas desempenham um papel fundamental na melhoria da eficiência dos algoritmos probabilísticos, permitindo a priorização de vértices mais promissores e aumentando as chances de encontrar soluções válidas de forma eficiente. Além disso, a aleatoriedade mostrou-se crucial para evitar os padrões repetitivos que frequentemente levam algoritmos determinísticos a becos sem saída, especialmente em grafos esparsos.

Os métodos probabilísticos destacaram-se como uma solução prática e escalável para problemas de grande escala, onde métodos determinísticos são computacionalmente inviáveis. Em grafos menos densos, essas abordagens não apenas superaram os métodos tradicionais, mas também demonstraram um desempenho robusto na identificação de cliques. Este estudo evidencia o potencial das técnicas probabilísticas, especialmente quando combinadas com heurísticas, para enfrentar desafios computacionais modernos e resolver problemas combinatórios de alta complexidade.

REFERÊNCIAS

[1] https://www.cos.ufrj.br/~celina/cbm07/12vinicius.pdf
[2] http://vigusmao.github.io/manuscripts/randomizados.pdf
[3] Class's PowerPoints