Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
TEORIA DOS GRAFOS

Algoritmos construtivos para o problema problema de particionamento de grafos ponderados por vértices

Integrantes do Grupo:

Felipe Vilela - 202465557B

Luiz Guilherme - 201676050

Professora: Luciana Brugiolo Gonçalves

Relatório do trabalho final da disciplina DCC059 - Teoria dos Grafos, parte integrante da avaliação da mesma.

Juiz de Fora

Setembro de 2024

1 Introdução

Este relatório tem como objetivo apresentar soluções para o problema de particionamento de grafos conhecido como Minimum Gap Graph Partitioning Problem (MGGPP). Esse problema envolve dividir um grafo não direcionado em subgrafos conectados, buscando minimizar a diferença entre os maiores e menores pesos dos vértices dentro de cada subgrafo, chamada de "gap". No desenvolvimento do trabalho, foram consideradas as características principais do problema, como a estrutura do grafo, as restrições de conectividade dos subgrafos e o objetivo de reduzir ao máximo a soma dos gaps em todos os subgrafos.

Foram implementados três tipos de algoritmos para resolver o MGGPP: um algoritmo guloso, um algoritmo guloso randomizado adaptativo (GRASP), e uma variação adaptativa reativa (GRASP-R). Esses métodos foram testados em diferentes instâncias com base no artigo de Cordone e Costa (2021), intitulado Minimum gap graph partitioning problem, publicado no European Journal of Operational Research, Volume 132. Os resultados obtidos foram comparados entre as três abordagens e com aqueles já documentados em estudos anteriores, com o objetivo de avaliar a eficiência das soluções desenvolvidas.

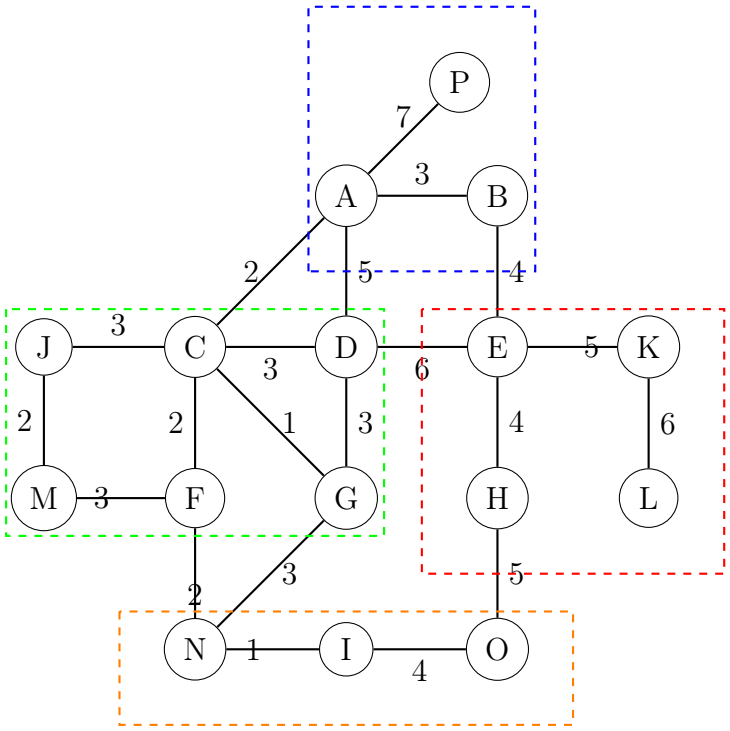
2 Descrição do problema

O MGGPP envolve um grafo não direcionado $G = (V, E)$, onde V representa os vértices e E representa as conexões entre eles. Cada vértice possui um peso associado w_v, que pode ser um valor que represente características como custo, demanda ou outro atributo relevante.

O objetivo principal do MGGPP é dividir o conjunto de vértices V em um número fixo de subgrafos $S_1, S_2, \ldots, S_p$. Para que essa divisão seja válida, cada subgrafo deve ser conectado, ou seja, deve ser possível chegar de um vértice a outro dentro do mesmo subgrafo sem sair dele. Além disso, cada subgrafo deve conter pelo menos dois vértices; não é permitido que um subgrafo seja formado apenas por um único vértice.

O critério para a partição é minimizar a diferença entre o maior e o menor peso dos vértices em cada subgrafo chamada de "gap". Portanto, o que queremos é que os subgrafos resultantes tenham distribuições de pesos o mais uniformes possíveis, ou seja, queremos evitar subgrafos onde um vértice é muito mais pesado que os outros.

A Figura 1 a seguir ilustra um exemplo de grafo, mostrando como os vértices são conectados e como eles podem ser particionados em subgrafos.



Pesos das Arestas: A-B: 3, A-C: 2, A-D: 5, A-P: 7, B-E: 4, C-D: 3, C-F: 2, D-G: 3, E-H: 4, G-C: 1, D-E: 6, C-J: 3, E-K: 5, K-L: 6, F-M: 3, F-N: 2, I-O: 4, H-O: 5, G-N: 3, J-M: 2, N-I: 1

Os vértices foram distribuídos em quatro subgrafos, e os pesos totais de cada um foram calculados da seguinte forma:

1. **Subgrafo 1** (A, B, P): Peso Total: $3 + 7 = 10$
2. **Subgrafo 2** (C, D, F, J, G): Peso Total: $2 + 3 + 2 + 1 + 3 + 3 + 3 = 17$
3. **Subgrafo 3** (E, K, H, L): Peso Total: $4 + 5 + 4 + 6 = 19$
4. **Subgrafo 4** (I, N, O): Peso Total: $1 + 2 + 4 + 5 = 12$

O gap total é definido como a diferença entre o peso do subgrafo mais pesado e o peso do subgrafo mais leve. No exemplo acima, temos:

- Peso Mínimo: 10 (Subgrafo 1)
- Peso Máximo: 19 (Subgrafo 3)

Portanto, o gap total é calculado como:

Gap Total = 19 - 10 = 9

3 Abordagens gulosas para o problema

Nesta parte, vamos descrever os algoritmos que criamos para resolver o problema de particionamento de grafos MGGPP. Vamos explicar cada abordagem e justificar por que usamos as estratégias escolhidas, além de como atualizamos a lista de candidatos.

3.1 Algoritmo guloso

O algoritmo Guloso visa particionar um grafo em p subgrafos, garantindo a conectividade e respeitando restrições de tamanho. Inicia-se com uma verificação para assegurar que o número de subgrafos p não ultrapasse o total de nós do grafo. Se essa condição não for atendida, uma mensagem de erro é exibida, e o algoritmo é encerrado.

Após essa validação, o algoritmo prepara um vetor para armazenar os subgrafos, inicializando cada um com valores de peso máximo, mínimo e total. Um conjunto denominado visited é utilizado para rastrear quais nós já foram explorados, enquanto todos os nós do grafo são coletados em um vetor para facilitar a seleção aleatória durante a construção dos subgrafos.

A construção dos subgrafos começa com a seleção aleatória de um nó não visitado, que servirá como ponto de partida para cada um dos p subgrafos. Uma busca em profundidade (DFS) é utilizada para coletar os nós conectados. Os nós são adicionados a uma pilha de exploração, e, enquanto essa pilha não estiver vazia, os nós são removidos e, se ainda não visitados, adicionados ao subgrafo, junto com seus vizinhos não visitados.

Após a formação dos subgrafos, o algoritmo verifica se algum deles possui menos de dois vértices, ajustando-se para garantir que todos mantenham um número mínimo de vértices. Por fim, o "gap"de cada subgrafo é calculado e impresso, junto com o gap total, refletindo a qualidade da partição resultante. A combinação da busca em profundidade com ajustes dinâmicos assegura uma partição que otimiza tanto a conectividade quanto a distribuição dos vértices nos subgrafos.

3.2 Algoritmo guloso randomizado

O algoritmo Guloso Randomizado Adaptativo começa verificando se o número de clusters desejado é maior que o número de vértices do grafo, retornando um erro se for o caso. Ele distribui os vértices aleatoriamente entre os subgrafos usando uma abordagem baseada no parâmetro "alpha"para escolher os candidatos a cada passo. Cada subgrafo é inicialmente preenchido selecionando vértices conectados, priorizando a seleção aleatória com base em pesos e conexões, até atingir um tamanho desejado.

Na segunda fase, o algoritmo tenta garantir que todos os subgrafos tenham pelo menos dois vértices, ajustando os subgrafos menores ao realocar vértices de outros subgrafos ou adicionando vértices não visitados conectados. Isso assegura que a estrutura do grafo seja mantida e que cada subgrafo seja funcional.

Por fim, qualquer vértice remanescente é alocado em subgrafos existentes, assegurando a conectividade. O algoritmo calcula o "gap"de cada subgrafo e retorna o "gap"total como uma medida de qualidade da partição, garantindo que os subgrafos sejam o mais equilibrados e conectados possível.

3.3 Algoritmo guloso randomizado reativo

A abordagem gulosa randomizada adaptativa reativa combina os elementos dos dois algoritmos anteriores, introduzindo uma estratégia reativa para aprimorar a partição do grafo. O algoritmo utiliza uma lista de valores predefinidos de "alpha"que são ajustados dinamicamente durante a execução, permitindo uma seleção mais eficaz dos vértices em cada iteração. Inicialmente, subgrafos são formados de maneira semelhante ao método randomizado adaptativo, com o critério de minimizar o gap entre os subgrafos.

Após a formação inicial dos subgrafos, o algoritmo realiza múltiplas iterações, onde ajusta e avalia a qualidade da partição em relação ao gap total obtido. Em cada iteração, o desempenho de cada valor de "alpha"é monitorado e ajustado com base na eficácia de redução do gap, permitindo ao algoritmo selecionar probabilisticamente os alphas que apresentam melhor desempenho. Essa adaptabilidade possibilita que o algoritmo explore diferentes combinações para otimizar a solução.

Além disso, se algum subgrafo apresentar um gap superior a um limite predefinido, o algoritmo busca reestruturar os subgrafos trocando vértices entre eles para reduzir a diferença de pesos. Assim, a função critério utilizada se mantém na minimização do gap, mas inclui um fator de penalização para subgrafos que excedem o limite estabelecido. Essa abordagem reativa permite que o algoritmo se adapte e melhore sua solução ao longo do tempo, promovendo uma partição mais eficaz e equilibrada.

3.4 Pseudocodigos

A seguir, apresentamos os pseudocódigos das três funções implementadas, correspondentes às heurísticas descritas anteriormente. Cada pseudocódigo detalha a sequência de passos executados para a resolução do problema de particionamento de grafos em suas respectivas abordagens: gulosa, gulosa randomizada e gulosa randomizada adaptativa reativa.

Algorithm 1: Algoritmo Guloso

Input: p
Output: Total gap

```
1  if p > número total de nós then
2      Exibir "Erro: Número de clusters não pode ser maior que o número de
         vértices.";
3      return -1;
4  end
5  Inicializar lista de subgrafos vazia com p subgrafos;
6  Inicializar conjunto 'visitados' para acompanhar os nós já explorados;
7  COLETAR todos os nós do grafo em uma lista 'nós';
8  for cada subgrafo de 1 até p do
9      Escolher aleatoriamente um nó não visitado como ponto inicial;
10     Adicionar este nó a uma pilha de exploração;
11     while a pilha não estiver vazia e o subgrafo não estiver completo do
12         Remover um nó da pilha;
13         if o nó não foi visitado then
14             Marcar nó como visitado e adicioná-lo ao subgrafo atual;
15             Adicionar os vizinhos não visitados do nó à pilha;
16         end
17     end
18     if o subgrafo tem menos de 2 vértices then
19         Mover vértices de outros subgrafos para garantir pelo menos 2 vértices;
20     end
21 end
22 CALCULAR e IMPRIMIR o "gap"total de todos os subgrafos;
23 return o "gap"total;
```

Algorithm 2: Algoritmo Guloso Randomizado Adaptativo

Input: p, α

Output: Total gap

if $p >$ *número de vértices* **then**
    Exibir "Erro e retorna -1";
    return *-1*;
end
Inicializar lista de subgrafos vazia com p subgrafos;
Inicializar conjunto 'visitados' para acompanhar os nós já explorados;
Coletar todos os nós do grafo em uma lista 'nós';
for *cada subgrafo de* 1 *até* p **do**
    Escolher aleatoriamente um nó não visitado como ponto inicial;
    Adicionar este nó ao subgrafo atual e marcar como visitado;
    Adicionar nós conectados ao ponto inicial a uma lista de candidatos;
    while *a lista de candidatos não estiver vazia e o subgrafo não estiver completo* **do**
        Selecionar aleatoriamente um nó da lista de candidatos com base no parâmetro α;
        Adicionar este nó ao subgrafo atual e marcar como visitado;
        Adicionar nós conectados a este nó à lista de candidatos, se ainda não visitados;
    end
    if *o subgrafo tem menos de* 2 *vértices* **then**
        Ajustar movendo vértices de outros subgrafos ou procurando vértices não visitados;
    end
end
for *cada nó não visitado ainda* **do**
    Tentar adicioná-lo a um subgrafo existente garantindo a conectividade;
end
Verificar se todos os subgrafos têm pelo menos 2 vértices e ajustar se necessário;
Calcular e imprimir o "gap"total de todos os subgrafos;
return *o "gap"total*;

Algorithm 3: Algoritmo Guloso Randomizado Adaptativo Reativo

Input: p, max_iter
Output: Total gap

if $p >$ *número de vértices* **then**
    Exibir "Erro e retorna -1";
    return *-1*;
end
Definir alphas $\leftarrow [0.1, 0.2, 0.3, 0.4, 0.5]$;
Inicializar subgrafos_best e gap_total como infinito;
for *iter de* 1 *até max_iter* **do**
    Inicializar subgrafos e visitados;
    Coletar todos os nós em 'nós';
    for *i de* 1 *até p* **do**
        Escolher aleatoriamente um nó não visitado;
        Adicionar ao subgrafo e marcar como visitado;
        Inicializar pilha com o nó;
        while *pilha não vazia e subgrafo não completo* **do**
            Remover nó do topo;
            Adicionar ao subgrafo e marcar como visitado;
            Construir RCL com nós conectados não visitados;
            Filtrar RCL por peso $\geq$ (max_weight $\times \alpha$);
            if *RCL não vazia* **then**
                Escolher aleatoriamente um nó e adicionar à pilha;
            end
        end
        if *subgrafo tem < 2 vértices* **then**
            Ajustar adicionando vértices não visitados;
        end
    end
    Calcular gap atual dos subgrafos;
    Atualizar gap_total e subgrafos_best se gap atual $<$ gap_total;
end
Imprimir subgrafos_best e gap_total;
return *gap_total*;

4 Experimentos computacionais

Nesta seção, descrevemos os experimentos realizados para avaliar a eficácia dos algoritmos aplicados ao problema de particionamento de grafos MGGPP. A descrição das instâncias utilizadas, o ambiente computacional, e o conjunto de parâmetros testados são detalhados nas subseções a seguir.

4.1 Descrição das instâncias

As instâncias utilizadas nos experimentos foram retiradas do Departamento de Ciência da Computação da Universidade de Milão. O conjunto de instâncias engloba grafos com diferentes tamanhos e características, e foram selecionadas de forma a representar uma variedade de condições do problema. A seleção das instâncias foi baseada na tentativa de testar ao menos uma instância de cada tamanho, além de realizar escolhas aleatórias para cobrir uma gama maior de configurações possíveis.
A Tabela 1 apresenta as instâncias utilizadas nos experimentos, com a respectiva descrição e o tamanho do grafo.

Tabela 1: Instâncias utilizadas nos experimentos computacionais

Instância	**Número de vértices (n)**	**Descrição**
n100d03p1i2	100	5 partições e 3300 arestas
n100d03p2i2	100	10 partições e 3300 arestas
n100d03p3i2	100	22 partições e 3300 arestas
n100d06p1i3	100	5 partições e 6600 arestas
n100d06p2i3	100	10 partições e 6600 arestas
n100d06p3i3	100	22 partições e 6600 arestas
n100plap1i1	100	5 partiçã e 570 arestaso
n100plap2i1	100	10 partições e 570 arestas
n100plap3i1	100	22 partições e 570 arestas
n200d03p1i5	200	5 partições e 13266 arestas
n200d03p2i5	200	14 partições e 13266 arestas
n200d03p3i5	200	38 partições e 13266 arestas
n200plap1i4	200	5 partição e 1140 arestas
n200plap2i4	200	14 partições e 1140 arestas
n200plap3i4	200	38 partições e 1140 arestas
n300d06p2i2	300	17 partições e 59800 arestas
n300plap1i1	300	6 partição e 1736 arestas
n300plap2i4	300	17 partições e 1704 arestas
n300plap3i5	300	53 partições e 1718 arestas

4.2 Ambiente computacional do experimento e conjunto de parâmetros

Os experimentos foram realizados em um ambiente Linux, utilizando a linguagem de programação C++ com o compilador `g++`, parte do GCC (GNU Compiler Collection). A máquina de teste foi equipada com um processador Intel Core i5 9400f, o que garantiu um desempenho adequado para a execução dos algoritmos, mesmo em instâncias de maior tamanho.

Para a geração de números aleatórios, foi utilizado o gerador de números pseudoaleatórios `rand()` disponível na biblioteca padrão do C++. Esse gerador foi suficiente para a execução dos testes de maneira estável.

Conjunto de parâmetros

Os parâmetros utilizados nos experimentos variaram de acordo com o algoritmo testado. A seguir, detalhamos as configurações principais utilizadas:

- **Número de iterações**: O número de iterações foi definido pelo usuário diretamente no terminal durante a execução dos algoritmos. Esse número impacta diretamente no desempenho dos algoritmos, especialmente para as abordagens mais complexas como o GRASP.

- **Valores de α no algoritmo guloso randomizado**: No caso do algoritmo guloso randomizado, foram testados os seguintes valores fixos de α: {0.1f, 0.2f, 0.3f, 0.4f, 0.5f}. Esses valores controlam o grau de aleatoriedade na construção das soluções e foram escolhidos de forma a permitir um bom equilíbrio entre exploração e exploração do espaço de soluções.

- **Faixa de valores de α e tamanho do bloco de atualização no algoritmo guloso randomizado reativo (GRASP-R)**: No caso do algoritmo reativo, o tamanho do bloco de atualização foi ajustado de maneira a equilibrar a exploração do espaço de soluções. Esse ajuste foi necessário para garantir que as probabilidades associadas aos valores de α fossem atualizadas de maneira eficiente, melhorando o desempenho do algoritmo ao longo do tempo.

Esses parâmetros foram selecionados com base em experimentos preliminares, buscando maximizar a eficiência do processo de particionamento e garantir que os algoritmos fossem testados em uma ampla gama de condições.

4.3 Resultados quanto à qualidade e tempo

A tabela a seguir contém os resultados de qualidade e tempo de execução para três diferentes variantes de algoritmos: o **Algoritmo Guloso**, o **Algoritmo Guloso Randomizado** e o **Algoritmo Guloso Randomizado Reativo**. Esses resultados são apresentados em termos de **qualidade da solução** (valores associados à função de otimização) e **tempo de execução** (em segundos).

				Algoritmo guloso		Algoritmo guloso randomizado		Algoritmo guloso randomizado reativo	
Instance	p	n	m	QUALIDADE	TEMPO (s)	QUALIDADE	TEMPO (s)	QUALIDADE	TEMPO (s)
n100plap1i1	5	100	570	410	0,0002454	386	0,0013443	377	0,0040990
n100plap2i1	10	100	570	674	0,00032950	562	0,0022017	785	0,0038469
n100plap3i1	22	100	570	1084	0,00036460	980	0,0072877	1449	0,0047011
n100d03p1i2	5	100	3300	323	0,00067240	332	0,0022435	470	0,0154897
n100d03p2i2	10	100	3300	542	0,00072800	480	0,0019178	889	0,0117510
n100d03p3i2	22	100	3300	1041	0,00061890	714	0,0016149	1755	0,0118906
n100d06p1i3	5	100	6600	353	0,00079700	268	0,0026421	468	0,0266172
n100d06p2i3	10	100	6600	617	0,00113530	476	0,0034419	858	0,0241332
n100d06p3i3	22	100	6600	965	0,00119190	908	0,0034295	1654	0,0258903
n200plap1i4	5	200	1140	871	0,00054580	729	0,0118979	861	0,0045653
n200plap2i4	14	200	1140	2112	0,00418500	1667	0,0028358	2230	0,0301087
n200plap3i4	38	200	1140	3898	0,00077660	3908	0,0014620	5842	0,0088235
n200d03p1i5	5	200	13266	721	0,00177240	615	0,0033069	899	0,0572544
n200d03p2i5	14	200	13266	1870	0,00202270	1503	0,0525753	2396	0,0603772
n200d03p3i5	38	200	13266	4324	0,00217440	3014	0,0449394	5776	0,0724463
n300plap1i1	6	300	1736	1518	0,00562500	1368	0,0085239	1602	0,0966310
n300plap2i4	17	300	1704	3959	0,00066510	3202	0,0111143	4148	0,0111661
n300plap3i5	53	300	1718	10405	0,00119240	7920	0,0123379	12843	0,0209248
n300d06p2i2	17	300	59800	3240	0,00981670	2670	0,0194629	4735	0,2621680

Cada linha da tabela representa uma instância de teste, identificada pela coluna *Instância*, que indica o nome da instância do problema. As colunas p, n, e m representam, respectivamente, o número de partições, o número de vértices e o número de arestas presentes em cada instância. Para cada algoritmo, são fornecidos os resultados de *qualidade* e *tempo de execução*.

Significado das colunas

- **QUALIDADE**: Representa o valor do gap minimo total para a instância avaliada. Quanto menor o valor, melhor é a qualidade da solução.
- **TEMPO (s)**: O tempo de execução do algoritmo para aquela instância, medido em segundos.

Análise dos Resultados

Algoritmo Guloso: Apresentou resultados aceitáveis em termos de minimização do gap, mas em muitos casos não conseguiu atingir a mesma qualidade que o algoritmo guloso randomizado.Em relação ao tempo de execução, o algoritmo guloso foi o mais rápido em praticamente todas as instâncias, o que é esperado por ser uma abordagem determinística e não envolver cálculos adicionais, como geração de números aleatórios ou avaliação de múltiplas opções.

Algoritmo Guloso Randomizado:Em geral, o algoritmo guloso randomizado apresentou uma qualidade melhor em relação ao guloso puro na maioria dos testes, indicando que a aleatoriedade introduzida permitiu explorar soluções que o algoritmo guloso não conseguiu encontrar.Entretanto,como esperado, o tempo de execução foi maior do que o do algoritmo guloso puro, devido à necessidade de avaliar múltiplas escolhas aleatórias. No entanto, essa diferença de tempo é relativamente pequena comparada à melhoria na qualidade do resultado.

Algoritmo Guloso Randomizado Reativo: Na maioria dos casos, o algoritmo guloso randomizado reativo teve o pior desempenho, indicando que a adaptabilidade do parâmetro não foi eficaz na obtenção de melhores soluções em relação aos outros algoritmos.Apresentou os tempos de execução mais altos, devido à complexidade adicional de ajustar o parâmetro dinamicamente.

Possiveis razões para o desempenho inferior do algoritmo reativo

- A abordagem reativa ajusta o alpha durante a execução, tentando encontrar o melhor valor com base em iterações anteriores. Esse processo pode levar tempo e nem sempre converge para a melhor escolha, especialmente em instâncias grandes e complexas.

- O ajuste reativo pode não garantir a seleção das melhores opções em todas as iterações, especialmente se o processo de aprendizagem do alpha for lento ou for influenciado por escolhas subótimas iniciais. Em contraste, o algoritmo adaptativo com alpha fixo é mais consistente, proporcionando um desempenho melhor.

- Com o número fixo de maximas iterações, se a adaptação do alphanão for suficientemente eficiente,o algoritmo pode terminar sem encontrar uma configuração ótima.omo uma boa opção intermediária entre os outros dois algoritmos.

5 Conclusões

Neste trabalho, abordamos o problema da partição de grafos utilizando algoritmos gulosos e suas variações. O desafio consistiu em dividir um grafo não direcionado em subgrafos ou clusters menores, onde os vértices representam entidades, como **pessoas em redes sociais, locais em mapas urbanos, ou produtos em um sistema de e-commerce**, e as arestas indicam relações entre essas entidades, como **amizade, proximidade geográfica ou similaridade de compra**. O objetivo foi otimizar as partições, maximizando as conexões internas aos subgrafos e minimizando as conexões entre diferentes subgrafos, ao mesmo tempo em que se mantém um equilíbrio entre os tamanhos dos clusters.

Modelamos o problema com grafos diversos, e implementamos três variações de algoritmos: o **Algoritmo Guloso**, o **Algoritmo Guloso Randomizado**, e o **Algoritmo Guloso Randomizado Reativo**. Cada variante tem suas próprias características em termos de exploração do espaço de soluções e balanceamento entre tempo de execução e qualidade das soluções encontradas.

Com os resultados, concluimos que o algoritmo guloso randomizado se destacou por oferecer uma melhor qualidade na minimização do gap em comparação com o guloso puro, mostrando que a introdução de aleatoriedade pode ser benéfica para encontrar soluções de maior qualidade. No entanto, o algoritmo reativo não conseguiu aproveitar seu mecanismo adaptativo para superar os outros dois, apresentando desempenho inferior na qualidade e tempos de execução mais elevados.

Dentre as dificuldades encontradas, uma delas foi ajustar corretamente os parâmetros do **Algoritmo Guloso Randomizado Reativo**, cujo desempenho depende fortemente dos valores de α e do tamanho do bloco de atualização das probabilidades. Além disso, a variação de complexidade entre as instâncias tornou difícil encontrar um único algoritmo que fosse otimizado para todos os casos.

Em conclusão, o trabalho permitiu compreender a eficácia de algoritmos gulosos e suas variações na resolução do problema de partição de grafos. Embora os resultados tenham mostrado a superioridade do **Algoritmo Guloso Randomizado** em termos de qualidade, há um claro trade-off entre qualidade e tempo de execução. Com as propostas de melhoria, espera-se alcançar algoritmos mais robustos e eficientes, aplicáveis em cenários reais e de maior escala.

Link para o programa

https://github.com/Luiizrst/tdgrafos02