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

Algoritmos para o Problema de Particionamento de Grafos com Mínima Diferença (MGGPP)

Integrantes do Grupo:

Breno Montanha - 202265513B

Lucas Henrique Nogueira - 202265515B

Pedro Henrique de Souza Rodrigues - 202165508B

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

O relatório tem como objetivo implementar e avaliar três algoritmos construtivos para resolver o Problema de Particionamento de Grafos com Mínima Diferença (Minimum Gap Graph Partitioning Problem - MGGPP). O MGGPP é um problema de otimização que visa particionar grafos ponderados por vértices em subgrafos conectados, de modo a minimizar a diferença (gap) entre o maior e o menor peso de cada subgrafo. Este problema é relevante em diversas áreas, como design de redes, balanceamento de carga e particionamento de sistemas, onde a distribuição uniforme de recursos ou demandas entre diferentes partes é desejável.

Ao longo desse trabalho, foram desenvolvidos três algoritmos construtivos: um algoritmo guloso, um algoritmo guloso randomizado adaptativo e um algoritmo guloso randomizado adaptativo reativo. Esses algoritmos foram aplicados a um conjunto de instâncias do problema e seus resultados foram comparados tanto em termos de qualidade de solução (gap) quanto de tempo de execução. Além disso, o desempenho obtido foi confrontado com resultados da literatura, como os apresentados por [3].

Por fim, a estrutura do relatório se encontra na seguinte disposição: na Seção 2, o problema é descrito formalmente através de um modelo de grafos; a Seção 3 descreve as abordagens propostas para o problema, enquanto a Seção 4 apresenta os experimentos computacionais, onde se detalha o design dos experimentos e as instâncias utilizadas, bem como se realiza uma análise comparativa dos resultados obtidos pelos algoritmos desenvolvidos; por fim, a Seção 5 traz as conclusões e as propostas para trabalhos futuros.

2 Descrição do problema

O *Minimum Gap Graph Partitioning Problem* (MGGPP) é um problema de particionamento de grafos ponderados por vértices que visa minimizar a diferença entre os pesos dos vértices em subgrafos resultantes. Formalmente, o MGGPP é definido em um grafo não direcionado $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas. Cada vértice $v \in V$ possui um peso associado w_v, que pode representar um atributo como demanda, custo, ou qualquer outra característica relevante.

O objetivo do MGGPP é particionar G em p subgrafos conexos, onde p é um valor predefinido tal que $1 < p < n$, e $n = |V|$ é o número total de vértices. Cada subgrafo resultante $G_r = (V_r, E_r)$ deve conter pelo menos dois vértices, e a diferença entre o maior e o menor peso de vértices dentro de cada subgrafo, denominada *gap*, deve ser minimizada. O *gap* de um subgrafo G_r é dado por:

$$\gamma_r = \max_{v \in V_r} w_v - \min_{v \in V_r} w_v$$

A função objetivo do problema visa minimizar a soma dos gaps em todos os subgrafos:

$$\text{minimizar} \sum_{r=1}^{p} \gamma_r$$

As principais restrições do MGGPP incluem:

- **Subgrafos Conexos**: Todos os subgrafos devem ser conexos, ou seja, deve haver um caminho entre qualquer par de vértices dentro de cada subgrafo.
- **Partição com Dois ou Mais Vértices**: Nenhum subgrafo pode conter menos de dois vértices.
- **Minimização do Gap Total**: O objetivo final é minimizar a soma dos gaps de todos os subgrafos, buscando subgrafos com distribuições de pesos o mais homogêneas possível.

Esse problema tem aplicações em diversos campos, como a setorização de redes de distribuição de água e o nivelamento de terrenos agrícolas, onde o objetivo é particionar a rede ou a área de maneira que as diferenças de pressão ou de elevação sejam minimizadas.

3 Abordagens gulosas para o problema

Para desenvolvermos os algoritimos usamos as seguintes funções: Fase Construtiva e Busca Local, onde na fase construtiva são criados os subgrafos pertencentes a solução baseados no α passado e a Busca Local tem a função de alterar os subgrafos de forma a minimizar o gap obtido pela Fase Construtiva alterando a quais subgrafos cada vértice pertence. O código fonte pode ser encontrado no repositório do GitHub [1].

Algorithm 1: Fase Construtiva

Input: α, $grafo$
Output: $listaSubgrafos$

```
Cria n subgrafos com 1 vértice respeitando uma certa distância nos gaps(baseado
  nos máximos, mínimos e número de subgrafos)
Adiciona um novo vértice que altere o mínimo possível o GAP de cada subgrafo
n ← 0
while n < numNodes do
    n ← 0
    Cria uma lista de possíveis adições em todos subgrafos
    Atualiza o n com o número de vértices em cada subgrafo
    Organiza a lista de possíveis adições em ordem crescente de alteração no GAP
    Adiciona de acordo com uma posição aleatória entre os α * 100% da lista de
      posições ou a primeira se o α for 0
end
return listaSubgrafos
```

Algorithm 2: Busca Local

Input: $grafo$, $listaSubgrafos$
Output: *Gap Total*

```
while Existir alterações que reduzam o gap do
    Verifica todas a possíveis alterações que diminuam o GAP
    Ordena as alterações de forma a reduzir ao máximo o GAP
    Faz a alteração que mais reduz o GAP
end
return listaSubgrafos
```

3.1 Algoritmo Guloso

O algorítimo guloso utiliza apenas da função Fase Construtiva passando como parametro α igual a 0, e assim pegando sempre a melhor escolha local, não necessitando o chamado da função Busca Local. Como sempre é escolhido a melhor opção, o resultado será sempre o mesmo independente de quantas vezes seja repetido.

Algorithm 3: Algoritimo Guloso

Input: $grafo$, $listaSubgrafos$

Output: *Gap Total*

1 Chama a função Fase Construtiva (Algoritimo 1)
2 Verifica se a Partição é válida
3 $gapTotal \leftarrow \sum gapSubgrafo$
4 **return** $gapTotal$

3.2 Algoritmo Guloso Randomizado

O algoritimo guloso randomizado é chamado passando um parametro α, para este determinado alpha, são feitas 10 iterações com objetivo de achar um gap médio para aquele determinado α. Este algoritimo utiliza das funções Fase Construtiva e Busca Local, com o objetivo de gerar sempre subgrafos diferentes e minimizar o gap deles, tentando assim se aproximar da solução ótima.

Algorithm 4: Algoritimo Guloso Randomizado

Input: α, $grafo$, $listaSubgrafos$

Output: *Gap Médio*

1 $gapMedio \leftarrow 0$
2 **for** $i \leftarrow 0; i < 10; i += 1$ **do**
3 | Chama a função Fase Construtiva (Algoritmo 1)
4 | Chama a função Busca Local (Algoritmo 2)
5 | Verifica se a partição é válida
6 | $gapTotal \leftarrow \sum gapSubgrafo$
7 | $gapMedio += gapTotal$
8 **end**
9 **return** $gapMedio/10$

3.3 Algoritmo guloso randomizado reativo

O algoritimo guloso randomizado reativo já é chamado passando uma lista de alphas, a qual de inicio possui mesma probabilidade para todos. A cada iteração, ele escolhe um alpha diferente e com esse determinado alpha, é executada as funções Fase Construtiva e Busca Local, assim como no randomizado, mas neste algoritimo, a cada 10 iterações, as probabilidades são recalculadas tendo como objetivo obter melhor resultado para o GAP.

Algorithm 5: Algoritimo Guloso Randomizado Reativo

Input: $listaAlphas$, $grafo$, $listaSubgrafos$
Output: *Gap Médio*

```
1  bestGap ← inf
2  gapMedio ← 0
3  Inicializa as probabilidades de cada alpha for i ← 0; i < 100; i += 1 do
4      Seleciona o α de acordo com a lista de probabilidades
5      Chama a função Fase Construtiva (Algoritmo 1)
6      Chama a função Busca Local (Algoritmo 2)
7      Verifica se a partição é válida
8      Atualiza vetores para calculo de probabilidades
9      Atualiza as probabilidades a cada 10 iterações
10     gapTotal ← ∑ gapSubgrafo
11     gapMedio += gapTotal
12 end
13 return gapMedio/100
```

4 Experimentos computacionais

Nesta seção, serão descritos todos os detalhes relacionados aos experimentos computacionais realizados nesse relatório. Para isso, são definidas subseções conforme a seguir:

4.1 Descrição das instâncias

As instâncias utilizadas nos experimentos foram obtidas a partir do site Cordone Research [2], que disponibiliza um total de 225 instâncias. No entanto, foram selecionadas 19 instâncias para os experimentos, seguindo os critérios estabelecidos pela docente da disciplina.

As 19 instâncias foram testadas em três computadores diferentes, com cada computador executando três vezes cada instância. Para garantir a precisão dos resultados, foi calculada a média dos três tempos de execução para cada instância em cada computador. Os resultados médios são apresentados na Tabela 1.

Tabela 1: Descrição das instâncias utilizadas

Instance Name	**num_subgraphs**	**num_vertex**	**num_edges**
n100d03p1i2	5	100	3300
n100d03p2i2	10	100	3300
n100d03p3i2	22	100	3300
n100d06p1i3	5	100	6600
n100d06p2i3	10	100	6600
n100d06p3i3	22	100	6600
n100plap1i1	5	100	570
n100plap2i1	10	100	570
n100plap3i1	22	100	570
n200d03p1i5	5	200	13266
n200d03p2i5	14	200	13266
n200d03p3i5	38	200	13266
n200plap1i4	5	200	1140
n200plap2i4	14	200	1140
n200plap3i4	38	200	1140
n300d06p2i2	17	300	59800
n300plap1i1	6	300	1736
n300plap2i4	17	300	1704
n300plap3i5	53	300	1718

4.2 Ambiente computacional do experimento e conjunto de parâmetros

O ambiente computacional utilizado para a implementação e execução dos testes foi baseado na linguagem de programação C++. O compilador utilizado foi o `g++`, e o comando de compilação foi `g++ *.cpp -o execGrupoX`, executado em um sistema Linux. Adicionalmente, foi empregado o Windows Subsystem for Linux (WSL) para realizar testes em um ambiente Windows.

As máquinas utilizadas nos testes incluíram as seguintes configurações:

- **Intel i7-12700H**: 16 GB de RAM, 4800 MT/s
- **Intel i7-10750H**: 16 GB de RAM, 2933 MT/s
- **AMD Ryzen 5 5500U**: 8 GB de RAM, 3200 MT/s

Para a geração de números aleatórios, foi utilizada a biblioteca `rand` do C++.

Conjunto de Parâmetros Utilizados

Para o algoritmo guloso randomizado, foram realizadas 10 iterações com os seguintes valores de α: 0.05, 0.2, 0.3, 0.45 e 0.5. O tamanho do bloco de atualizações adotado foi de 10. Esses parâmetros foram selecionados para avaliar a eficácia do algoritmo sob diferentes condições, permitindo uma análise abrangente do desempenho.

4.3 Resultados quanto à qualidade e tempo

A tabela 2 possui colunas para identificar a instância, o menor valor teórico (LB), o melhor resultado obtido através dos nossos algorítimos (Best Gap), o resultado e o tempo dos algoritimos Guloso, Guloso Randomizado e Guloso Randomizado Reativo. Podemos perceber que o melhor resultado obtido pelos nossos algoritmos se aproxima do menor valor teórico em alguns casos, mas em outros ele fica distante, como podemos ver no n300plap3i5, que tem como menor gap teórico 152, mas o gap que foi obtido por nós é 1758. Percebemos também que o algoritmo se adaptou melhor quando o grafo é denso, visto que com mais arestas, é possível remover um vértice mais facilmente sem deixar o subgrafo desconexo, permitindo assim maior alteração nos subgrafos para reduzir o gap obtido pela Fase Construtiva através da Busca Local.

Tabela 2: Resultados dos algoritmos de construção para cada instância.

Instance	**LB**	**Best Gap**	**Greedy**		**Randomized**		**Reactive**	
			Gap	**Time**	***Gap***	**Time**	***Gap***	**Time**
n100d03p1i2	82	82	**82**	**0.0202**	**82**	0.2769	**82**	3.2119
n100d03p2i2	67	70	72	0.0236	**70**	**0.8378**	**70**	7.2435
n100d03p3i2	48	59	75	0.0325	**59**	**1.0039**	60	10.0875
n100d06p1i3	84	85	**85**	**0.0377**	**85**	0.5344	**85**	5.2435
n100d06p2i3	73	74	**74**	**0.0512**	75	0.6578	**74**	24.5310
n100d06p3i3	49	52	57	0.0563	57	0.8751	**52**	**20.0195**
n100plap1i1	78	109	156	0.0055	**109**	**0.2151**	**109**	1.4433
n100plap2i1	65	140	222	0.0096	213	0.2021	**140**	**1.2951**
n100plap3i1	44	362	412	0.0060	396	1.7532	**362**	**122.7956**
n200d03p1i5	182	183	**183**	**0.2861**	**183**	10.9319	**183**	94.3627
n200d03p2i5	155	166	173	0.2846	**166**	**17.2390**	**166**	160.6514
n200d03p3i5	105	159	188	0.2965	166	21.0921	**159**	**218.7877**
n200plap1i4	178	219	289	0.0397	259	0.3617	**219**	**3.4426**
n200plap2i4	151	260	837	0.0339	437	1.3432	**260**	**10.1630**
n200plap3i4	102	752	962	0.0347	799	0.9089	**752**	**8.6119**
n300d06p2i2	236	256	240	2.9663	274	23.1880	**256**	**256.5307**
n300plap1i1	276	404	538	0.1017	**404**	**1.3823**	**404**	10.8422
n300plap2i4	238	702	1488	0.1126	736	1.8867	**702**	**18.9577**
n300plap3i5	152	1758	**1758**	**0.1443**	2570	2.0009	2476	22.7132

5 Conclusões e trabalhos futuros

Assim, com base na proposta do trabalho de implementar uma otimização para o Problema de Particionamento de Grafos de Diferença Mínima, foram elaborados com sucesso 3 algoritimos gulosos diferentes para otimizar o problema, um guloso simples, um guloso randomizado, e um guloso randomizado reativo.

O algoritimo principal consiste em 2 partes, uma fase construtiva (que gera os subgrafos baseado no α usado como parâmentro) e uma fase de busca local (que tem como objetivo reduzir o GAP obtido pela fase construtiva melhorando o subgrafo obtido).

Entre alguns dos desafios encontrados, tivemos que lidar com problemas de vazamento de memória, que resultaram em sobrecarga da memória RAM do sistema, e inconsistências no algoritmo ao resolver certos casos de teste. Ao final obtivemos resultados próximos às soluções reais, mostrando a boa eficiência dos algoritimos.

Referências

[1] Repository. https://github.com/Lucas-Henriquee/Trabalho-de-Grafos, 2024. Acesso em: 29 de setembro de 2024.

[2] Giovanni Cordone. Research projects. https://homes.di.unimi.it/cordone/research/research.html, 2024. Acesso em: 29 de setembro de 2024.

[3] Jens Uhlmann, Andreas Fink, and Cornelis W Duin. Solving the minimum weight dominating set problem on large-scale graphs by a local search hybridized with integer programming. *European Journal of Operational Research*, 291(2):490–501, 2021.