Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
Teoria dos Grafos

Algoritmos Construtivos para o Problema da Clusterização Capacitada

Grupo 45

Maria Eduarda Ribeiro Facio MAT 202065516B

Mariana Richa Ferreira - MAT 202065517B

Yuri Alexander Sudre Almeida Souza - MAT 202065512B

Professor: Stênio Sã Rosário F. Soares

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

Juiz de Fora

Agosto de 2022

1 Introdução

O presente Relatório Técnico tem como objetivo descrever o uso de algoritmos construtivos gulosos para o Problema da Clusterização Capacitada. Considerou-se neste trabalho as características do problema modelado sobre um grafo simples e não direcionado. Foram desenvolvidos três algoritmos, guloso, randomizado adaptativo e randomizado reativo. Os mesmos foram avaliados sobre um conjunto de instâncias e os resultados foram comparados com os apresentados na seção 4 deste relatório.

O restante do trabalho está assim estruturado: na Seção 2 o problema é descrito formalmente através de um modelo em grafos; a Seção 3 descreve as abordagens propostas para o problema, enquanto a Seção 4 apresenta os experimentos computacionais, onde se descreve o design dos experimentos, o conjunto de instâncias (*benchmarks*), bem como se apresenta a análise comparativa dos resultados dos algoritmos desenvolvidos e a literatura; por fim, a Seção 5 traz as conclusões do trabalho e propostas de trabalhos futuros.

2 Descrição do problema

O Problema da Clusterização Capacitada consiste em particionar os vértices de um grafo em k partições (*clusters*), de forma que as somas dos pesos dos vértices em cada *cluster* seja maior ou igual ao limite inferior L e menor ou igual ao limite superior U. O objetivo é maximizar a soma dos pesos das arestas entre vértices alocados em uma mesma *cluster*.

Se, ao final da execução do algoritmo que resolve o Problema da Clusterização Capacitada, alguma das *clusters* tiver a soma dos pesos dos vértices menor do que o limite inferior L ou se algum vértice do grafo não estiver em nenhuma partição, então considera-se que não foi possível resolver o Problema da Clusterização Capacitada.

As aplicações práticas desse problema encontram-se na formação de grupos de pessoas com traços em comum como forma de segmentação, na organização dos dados de uma empresa que vise facilitar o acesso por parte dos usuários dos softwares e sistemas, na clusterização de servidores, usada por empresas de cloud computing, SaaS, IaaS e PaaS para assegurar amplo acesso dos seus clientes aos respectivos sistemas, etc.

3 Abordagens gulosas para o problema

Algoritmos gulosos são soluções simples e de fácil implementação para problemas em que se espera que, ao escolher a opção mais promissora dada a iteração atual, a mesma acarrete em menores custos e evite problemas futuros em nível global, sendo que para cada tomada

de decisão,há uma tratativa diferente de acordo com o tipo de algoritmo guloso escolhido. Foram estudados e implementados 3 algoritmos gulosos: simples, randomizado adaptativo e randomizado reativo, os quais serão melhores explicados a seguir. Todos retornam a qualidade da solução, que é a soma dos pesos das arestas em todas as *clusters*.

3.1 Algoritmo Guloso

O Algoritmo Guloso simples constroi uma solução de forma iterativa, inserindo a cada passo um elemento no conjunto solução. As escolhas são definitivas e o algoritmo não sofre arrependimentos durante a sua execução.

Para o Problema da Clusterização Capacitada, a solução final será um conjunto (de tamanho k) de subgrafos aresta-induzidos. Para encontrar tais arestas, devemos encontrar, primeiro, o que chamamos de $solucao\_clusters$, um conjunto (de tamanho k) de conjuntos de arestas.

Inicializa-se uma lista de candidatos (*arestas*) com todas as arestas do grafo ordenadas em ordem crescente de pesos.

A cada iteração, deve-se escolher o melhor candidato da lista de candidatos, ou seja, a aresta com o maior peso, e inseri-la na solução parcial, na posição $i \in 1, ..., k$ que ela melhor se encaixar e, em seguida, removê-la da lista de candidatos, ou, no caso de ela não se encaixar em nenhuma *cluster*, apenas realizar a remoção.

Lembrando que, para que uma aresta seja inserida em uma *cluster*, nenhum dos dois nós ligados a ela podem pertencer a outra *cluster*. Além disso, para inserir a aresta e, consequentemente, os seus vértices (se eles já não estiverem na *cluster* atual), a soma dos pesos dos nós não poderá ultrapassar do limite superior U.

Ao final de todas as iterações, quando a lista de candidatos estiver vazia, deve-se verificar se foi ou não possível resolver o Problema da Clusterização Capacitada, seguindo os critérios já citados na seção 2.

A seguir, tem-se o pseudocódigo do algoritmo:

Algorithm 1: Algoritmo Guloso Simples

Output: *Qualidade Solução*

```
1  solucao ← ∅;
2  arestas ← E;
3  ordenar arestas em ordem crescente de pesos
4  while ! arestas.empty() do
5      best_candidato ← arestas.back();
6      if best_candidato se encaixa em alguma cluster then
7          if (soma dos pesos dos vértices já inseridos + pesos dos vértices que serão
             inseridos) ⩽ U then
8              solucao ← solucao ∪ arestas[best_candidato];
9              remove arestas[best_candidato];
10         else
11             if best_candidato não se encaixa em mais nenhuma cluster then
12                 remove arestas[best_candidato];
13             end
14         end
15     else
16         remove arestas[best_candidato];
17     end
18 end
19 for i ← 0 to k do
20     if soma dos pesos dos nós na cluster i < L then
21         não foi possível resolver o problema
22         return -1;
23     end
24 end
25 if algum vértice do grafo não está presente em nenhuma cluster then
26     não foi possível resolver o problema
27     return -1;
28 end
29 qualidade ← soma dos pesos das arestas em todas as clusters;
30 return qualidade;
```

3.2 Algoritmo Guloso Randomizado Adaptativo

Sobre o algoritmo guloso simples, ele é eficiente em questão de tempo, porém, sempre retorna a mesma solução, que nem sempre é de boa qualidade/eficácia. Por isso, deve-se

tirar proveito da sua eficiência para explorar melhor o espaço de busca, utilizando, assim, uma abordagem gulosa e estocástica simultaneamente.

O Algoritmo Guloso Randomizado Adaptativo parametriza o quanto o algoritmo será guloso. Para isso, é usado um laço externo ao algoritmo já apresentado em que, a cada passo, uma solução é construída de forma gulosa-randomizada, segundo um parâmetro α no intervalo [0, 1]. Ou seja, o elemento escolhido dentro da lista de candidatos não será mais o que consideramos o melhor candidato, mas sim um valor aleatório, que tem sua posição na lista de candidatos definida por:

$$K = randomRange(0, \alpha * |L| - 1)$$

A seguir, tem-se o pseudocódigo para esse tipo de abordagem:

Algorithm 2: Algoritmo Guloso Randomizado Adaptativo

Input: $\alpha, numIter$
Output: *Qualidade Solução*

```
1  BESTsolucao ← ∅;
2  for (i = 0; i < numIter; i++) do
3      solucao ← ∅;
4      arestas ← E;
5      ordenar arestas em ordem crescente de pesos
6      while ! arestas.empty() do
7          K = randomRange(0, α*arestas.size()-1);
8          best_candidato ← arestas[K];
9          Realiza o mesmo algoritmo de antes, com a diferença do melhor candidato.
10     end
11     if solucao atual melhor do que BESTsolucao then
12         ou seja, se a solução atual tiver a soma do peso das arestas em todas as
             clusters maior do que essa soma da melhor solução encontrada:
13         BESTsolucao = solucao;
14     end
15 end
16 Considerando agora o conjunto BESTsolucao, realiza a mesma lógica de antes
     para saber se o problema foi resolvido ou não e encontrar o valor da qualidade.
17 return qualidade;
```

3.3 Algoritmo Guloso Randomizado Reativo

Sobre o algoritmo guloso randomizado adaptativo, ele explora mais o espaço de busca, porém, dependendo da instância, o ajuste do parâmetro α pode ser um problema. Para resolver isso, deve-se atribuir ao algoritmo a capacidade de reagir às características da instância.

O Algoritmo Guloso Randomizado Reativo procura ampliar a capacidade do anterior de explorar diferentes regiões do espaço de busca. O algoritmo requer o monitoramento dos resultados obtidos ao longo do processo de busca e o ajuste automático do parâmetro α de aleatoriedade.

Para isso, o algoritmo toma como entrada não apenas um valor de α, mas um conjunto de valores a serem testados. A escolha do valor do parâmetro α em cada iteração leva em consideração o histórico da qualidade obtida em iterações anteriores com o mesmo valor de α. Tem-se os seguintes parâmetros:

- $\alpha_i \in \{\alpha_1, \alpha_2, ..., \alpha_m\}$: conjunto de valores de α
- $p_i \in \{p_1, p_2, ...p_m\}$: probabilidade de cada α_i ser escolhido para uma iteração
- M_i: média de qualidade das soluções obtidas quando se utilizou α_i na construção
- $F(S*)$: qualidade da melhor solução obtida

$$q_i = (\frac{F(S*)}{M_i})^{\delta}$$

$$p_i = \frac{q_i}{\sum_{j=1}^{j=m} q_j}$$

As probabilidades são calculadas a cada bloco de tamanho N iterações. A seguir, tem-se o pseudocódigo deste algoritmo:

Algorithm 3: Algoritmo Guloso Randomizado Reativo

Input: $\alpha, numIter, bloco$

Output: *Qualidade Solução*

```
1  BESTsolucao ← ∅;
2  inicializaVetores(P, M, m);
3  for (i = 0; i < numIter; i++) do
4      if i % bloco == 0 then
5          atualizaProbabilidades(P, M, α, BESTsolucao);
6      end
7      solucao ← ∅;
8      arestas ← E;
9      ordenar arestas em ordem crescente de pesos
10     α = escolheAlfa(P);
11     while ! arestas.empty() do
12         K = randomRange(0, α*arestas.size()-1);
13         best_candidato ← arestas[K];
14         Realiza o mesmo algoritmo de antes, com a diferença do melhor candidato.
15     end
16     atualizaMedidas(M, solucao, α);
17     if solucao atual melhor do que BESTsolucao then
18         ou seja, se a solução atual tiver a soma do peso das arestas em todas as
            clusters maior do que essa soma da melhor solução encontrada:
19         BESTsolucao = solucao;
20     end
21 end
22 Considerando agora o conjunto BESTsolucao, realiza a mesma lógica de antes
     para saber se o problema foi resolvido ou não e encontrar o valor da qualidade.
23 return qualidade;
```

4 Experimentos computacionais

Nesta seção estão descritos os experimentos computacionais realizados, especificando as instâncias e as configurações da máquina utilizada para realizar o experimento.

4.1 Descrição das instâncias

As instâncias utilizadas e suas informações foram obtidas através do material disponibilizado pelo Professor Stênio Sã na plataforma virtual Classroom. Tais instâncias estão

listadas na Tabela 1.

Tipo de Instância	Nome da Instância	#vértices	#grupos	Best Literatura
Handover	20_5_270001	20	5	
Handover	20_10_270001	20	10	2148.00
Handover	30_5_270003	30	5	920.00
RanReal240	RanReal240_01.txt	240	12	225003.70
RanReal240	RanReal240_04.txt	240	12	225683.17
RanReal240	RanReal240_07.txt	240	12	209305.70
RanReal480	RanReal480_01.txt	480	20	556639.68
RanReal480	RanReal480_04.txt	480	20	522790.22
RanReal960	RanReal960_01.30.txt	960	30	1340369.47
Sparce82	Sparce-82_02.txt	82	8	1306.64

Tabela 1: Descrição das instâncias usadas no experimento

4.2 Ambiente computacional do experimento e conjunto de parâmetros

O código foi implementado em $C++$ e todos os testes foram realizados em ambiente Linux (Ubuntu), utilizando o terminal do Ubuntu via linha de comando. Os comandos utilizados foram: $g++ \ *.c* \ -o \ execGrupo45\_$ e $./execGrupo45\_ \ < arquivo\_entrada > \ < arquivo\_saida > \ < Tipo\_Instancia >$, sendo que $< Tipo\_Instancia >$ deve ser um dentre os tipos listados na Tabela 1.

As configurações da máquina estão listadas na Tabela 2.

Máquina	Notebook Samsung Book X55
Sistema Operacional	Ubuntu 20.04.4 LTS / Windows 11
Processador	Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
Memória RAM	16,0 GB
Versão do compilador GCC	(Ubuntu 9.4.0-1ubuntu1 20.04.1) 9.4.0

Tabela 2: Configurações da máquina

Para o Algoritmo Guloso Randomizado Adaptativo, foram utilizadas 500 iterações e os seguintes valores de α: $\alpha = \{0.05, 0.10, 0.15, 0.30, 0.50\}$. E, para o Algoritmo Guloso Randomizado Reativo, foram utilizadas 2500 iterações, um bloco de atualização das probabilidades de tamanho 100 e o seguinte vetor α: $\alpha = [0.05, 0.10, 0.15, 0.30, 0.50]$.

4.3 Resultados quanto à qualidade e tempo

Após o desenvolvimento do problema, são demonstrados nas Tabelas 3 e 4 os resultados com relação à qualidade de cada instância.

Nos resultados apresentados na Tabela 3, a primeira coluna indica o nome da instância utilizada, a segunda coluna mostra o valor da melhor média das soluções encontradas dentre todas as execuções realizadas para os diferentes algoritmos apresentados e a terceira coluna representa os resultados contidos na literatura.

As demais colunas da tabela fornecem detalhadamente o desvio relativo da média das soluções do algoritmo com relação à melhor média para a instância (coluna 3), e o tempo médio de execução, dado em segundos.

As heurísticas de construção foram executadas para cada instância em três diferentes formas: gulosas, gulosas randomizadas adaptativas e gulosas randomizadas reativas. A Tabela 3 exibe os resultados gerados pelos algoritmos de construção guloso (colunas 4 e 5), guloso randomizado adaptativo (colunas 6, 7, 8, 9, 10, 11, 12, 13, 14 e 15) e guloso randomizado reativo (colunas 16 e 17) em comparação com as soluções produzidas pela literatura (coluna 3) para todas as instâncias. Os valores destacados em negrito indicam os melhores resultados.

Com o objetivo de realizar os testes comparativos, foi utilizado um conjunto de 10 instâncias, sendo as 3 primeiras consideradas pequenas (todas do tipo Handover), 4 instâncias consideradas médias (dos tipos Sparce82 e RanReal240), e 3 consideradas grandes (dos tipos RanReal460 e RanReal960).

A princípio, a diferença percentual média (RDI, do inglês *Relative Deviation Index*) das soluções pode ser calculada conforme a Equação 1 descrita a seguir:

$$RDI(a) = \frac{Media(a) - b}{c - b} \times 100 \tag{1}$$

onde $Media(a)$ é o valor médio das soluções obtidas pelo algoritmo a, b é o valor da melhor média obtida dentre todos os algoritmos comparados, c é o valor da pior solução dentre os algoritmos utilizados, e nesse caso específico, a pior média encontrada. Note que $RDI(a) \in [0, 100]$ e quanto menor o valor resultante, maior é a qualidade da solução do algoritmo a.

Qualidade Média dos Algoritmos									
Instância	Best	Literatura	Guloso	Randomizado					Reativo
				α = 0.05	α = 0.10	α = 0.15	α = 0.30	α = 0.50	
20_5_270001	**1786**		**1786**	**1786**	**1786**	1733.8	1759.5	1491.5	1451.2
20_10_270001	**2148**	**2148**	1023	1023	1023	1023	1017.1	841.6	948.4
30_5_270003	**3118**	920	**3118**	**3118**	2856.9	2980.7	2634.7	2743.5	2736.1
RanReal240_01.txt	**225003.7**	**225003.7**	130462	123654	122610	123452	121972	122153	121554
RanReal240_04.txt	**225683.17**	**225683.17**	137543	122722	123930	122870	122062	122435	122645
RanReal240_07.txt	**209305.7**	**209305.7**	131939	122469	124344	122711	122970	122512	120985
RanReal480_01.txt	**556639.68**	**556639.68**	335002	295671	301152	289985	292480	285267	287508
RanReal480_04.txt	**522790.22**	**522790.22**	316337	296624	294236	289847	294430	291857	285234
RanReal960_01.30.txt		1340369.47	833474						
Sparce-82_02.txt	**1306.64**	**1306.64**	849	842.9	794.1	771.8	692.9	620.5	734.7

Tabela 3: Qualidade Média dos Algoritmos para cada Instância

Analisando as Tabelas 3 e 4, é possível destacar:

- O Algoritmo Guloso possui, em 100% dos casos, o melhor tempo de execução. Esse fato se dá por causa da sua abordagem direta sobre cada uma das instâncias analisadas, uma vez que ele toma decisões de acordo com o príncipio da aresta de maior peso, optando sempre pelo candidato, a princípio, mais promissor. O algoritmo não considera outras opções que, inicialmente, não se apresentam como as melhores, porém, poderiam representar melhoras significativas, resultando em uma solução mais adequada ao problema.

- Pode-se perceber que, na maioria dos casos, os resultados da literatura são os melhores. Isso significa que a heurística escolhida pode não ter sido a melhor, podendo existir outras não analisadas pela equipe que tenham melhor desempenho, como na literatura.

- Para a instância do tipo RanReal960, foi inviável rodar os algoritmos gulosos randomizados até que eles fossem concluidos efetivamente. Um teste foi feito, no algoritmo randomizado adaptativo, com apenas 1 iteração, ao invés de 500, e o tempo demandado foi de 4167.38 segundos para $\alpha = 0.05$, ou seja, uma execução do algoritmo com 500 iterações demoraria mais de 2083690 segundos, isto é, mais do que 3 semanas. Isso para apenas 1 execução, o objetivo seria obter o resultado médio de 10 execuções para cada um dos 5 alfas.

Instância	Best	Literatura	Guloso		Randomizado										Reativo	
					α = 0.05		α = 0.10		α = 0.15		α = 0.30		α = 0.50			
			RDI	Tempo	*RDI*	Tempo	*RDI*	Tempo	*RDI*	Tempo	*RDI*	Tempo	*RDI*	Tempo	*RDI*	Tempo
20_5_270001	**1786**		**0.0**	**0 s**	**0.0**	0.045 s	**0.0**	0.0398 s	15.59	0.0508 s	7.92	0.0412 s	87.96	0.0379 s	100	0.1884 s
20_10_270001	**2148**	**0.0**	86.11	**0 s**	86.11	0.0471 s	86.11	0.0418 s	86.11	0.0439 s	86.57	0.0443 s	100	0.0432 s	91.82	0.1953 s
30_5_270003	**3118**	100	**0.0**	**0 s**	**0.0**	0.071 s	11.88	0.0645 s	6.25	0.0642 s	21.99	0.0635 s	17.04	0.0649 s	17.37	0.293 s
RanReal240_01.txt	**225003.7**	**0.0**	91.39	**12.707 s**	97.97	7369.2 s	98.98	7217.84 s	98.17	7152.68 s	99.60	7355.39 s	99.42	7147.08 s	100	37018.9 s
RanReal240_04.txt	**225683.17**	**0.0**	85.06	**13.051 s**	99.36	7291.95 s	98.20	7297.3 s	99.22	7713.25 s	100	7211.5 s	99.64	7113.75 s	99.44	36670 s
RanReal240_07.txt	**209305.7**	**0.0**	87.60	**13.255 s**	98.32	7362.2 s	96.20	7289.95 s	98.05	7684.7 s	97.75	7118.25 s	98.27	7329.4 s	100	36525.25 s
RanReal480_01.txt	**556639.68**	**0.0**	81.67	**224.615 s**	96.17	125226 s	94.15	118454.5 s	98.26	117207.5 s	97.34	119587 s	100	118530 s	99.17	592850 s
RanReal480_04.txt	**522790.22**	**0.0**	86.91	**227.559 s**	95.21	119314 s	96.21	119387 s	98.06	117095.5 s	96.13	117964.5 s	97.21	117667.5 s	100	593192.5 s
RanReal960_01.30.txt				**4388.85 s**												
Sparce-82_02.txt	**1306.64**	**0.0**	66.70	**0.01 s**	67.59	2.9974 s	74.70	3.0842 s	77.95	3.272 s	89.45	3.1925 s	100	3.1547 s	83.36	15.9353 s

Tabela 4: RDI e Tempo de Execução dos Algoritmos para cada Instância

5 Conclusões e trabalhos futuros

O trabalho proposto consistiu em resolver de modo eficiente o Problema da Clusterização Capacitada. Para resolvê-lo, foi utilizada a modelagem de um grafo simples e não direcionado e foram desenvolvidos três algoritmos gulosos, um simples, um randomizado e um reativo.

Foi concluido que, na maioria dos casos analisados, os resultados da literatura são melhores do que os encontrados pelos algoritmos aqui implementados, além de que o tempo de processamento de instâncias grandes são extremamente longos e, às vezes, inviáveis. Por isso, em busca de melhorar os resultados, podem ser estudadas outras heurísticas não tratadas pela equipe, caso existam.

Referências

[1] I. Adler, N. K. Karmarkar, M. G. C. Resende, and G. Veiga. An implementation of Karmarkar's algorithms for linear programming. *Mathematical Programming*, 44:297–335, 1989.