Pontifícia Universidade Católica de Minas Gerais
Engenharia de Software

Trabalho Final da Disciplina Algoritmos Computacionais em Grafos

Bruno Armanelli
Douglas Domingues
Henrique Freire
Luiz Antunes

Belo Horizonte
2020

INTRODUÇÃO

Como avaliação final na disciplina de *Algoritmos Computacionais em Grafos* o seguinte problema foi proposto:

> "Em uma universidade existe um departamento de computação com cursos de graduação, mestrado e doutorado. Ao todo, existem n alunos e k professores nos três níveis de ensino. Todos os alunos do departamento estão envolvidos com trabalhos de pesquisa em alguma área da computação. O grau de relacionamento entre os alunos é medido pela proximidade entre os temas de seus trabalhos de pesquisa. Dessa forma, deseja-se alocar os k professores do departamento de computação para orientar os trabalhos de pesquisa dos alunos, sendo que cada professor deve ser alocado para orientar um grupo de alunos com trabalhos semelhantes (ou o mais semelhante possível). O número n de alunos da instituição é maior do que o número k de professores, ou seja, $k \leq n$. Assim, um professor irá orientar um ou mais alunos."

Considere o grafo cujos vértices representam os alunos. As arestas entre eles tem pesos correspondentes às distâncias (baseadas na semelhança entre suas áreas de pesquisa). Como conhecemos as distâncias entre todas as áreas do conhecimento, e as áreas de pesquisa de todos os alunos, esse é um grafo completo.

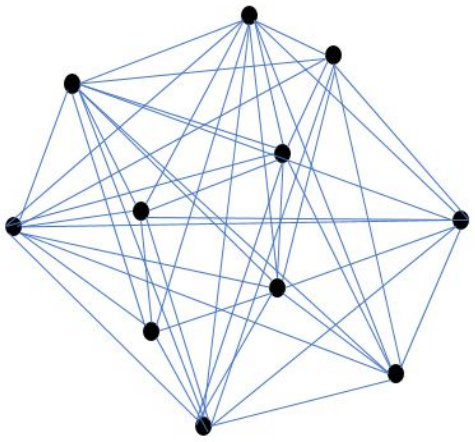


Figura 1 - Grafo Completo de Alunos

Uma rápida observação nos leva à conclusão que a tarefa de dividir os n alunos entre os k professores se trata de um problema de particionamento de grafos. Mais especificamente, de **particionamento em subgrafos conexos e balanceados com pesos**. O método pensado para resolver foi baseado nos algoritmos de *Clustering,* especialmente de tipo hierárquico.

Primeiramente, então, precisamos de algumas definições:

DEFINIÇÕES INICIAIS

Definição 1.1: A **distância entre dois vértices**, ou o valor do peso da aresta entre dois vértices, é dada pelo grau de dissimilaridade entre as áreas de pesquisas dos respectivos alunos, fornecida como entrada do algoritmo.

A distância entre um aluno *a* a ele mesmo d(a,a) = 0.

A distância é simétrica: d(a, b) = d(b, a).

Definição 1.2: A **distância entre dois subgrafos** é dada pela distância média entre os vértices de subgrafos opostos. Ou seja, dados os subgrafos S_1 e S_2, e os vértices v_{i1} e v_{j2}, a distância $d(S_1, S_2)$ é dada pela média dos pesos das arestas (v_{i1}, v_{j2}).

No caso particular em que o subgrafo é um vértice único, vale a definição acima.

Definição 1.3: O **grau de diferença** de um subgrafo é a soma dos pesos das suas arestas.

Definição 1.4: Um **casamento,** ou **matching**, em um grafo *G* é um conjunto M de arestas não adjacentes duas a duas.

SOLUÇÃO

A ideia geral do algoritmo consiste em encontrar os casamentos maximais entre subgrafos do grafo da figura 1. Então, os subgrafos são unidos pelas arestas de casamento e assim se reduzem o número de subgrafos (desconexos entre si) até que esse número iguale o número de professores. Queremos fazer isso de modo que esses subconjuntos sejam balanceados - ou seja, o número de alunos é distribuído o mais igualmente possível entre os professores - e que o grau de diferença de cada grupo seja o menor possível, significando que esses alunos pesquisem áreas do conhecimento relacionadas.

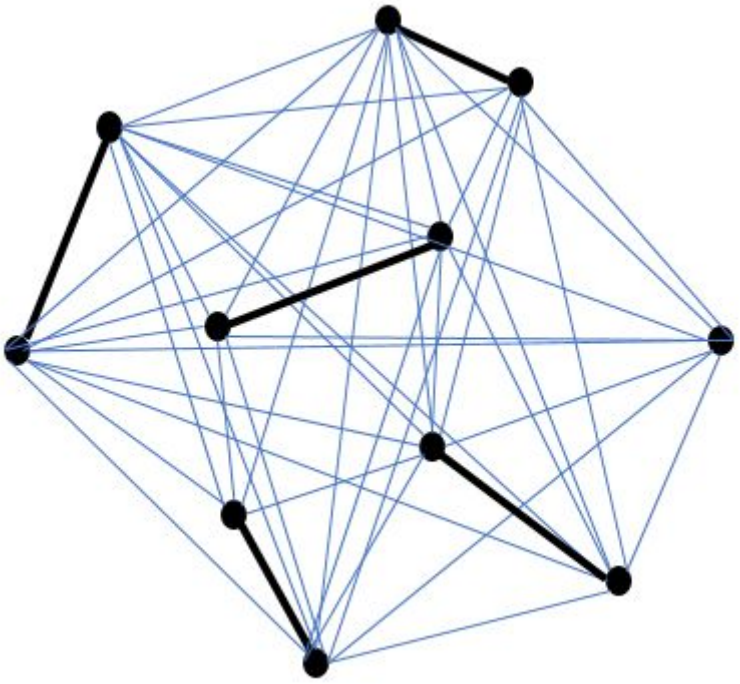
Tomaremos como ponto de partida o grafo da figura 1. No entanto, é preciso esclarecer que esse grafo representa apenas as **possíveis ligações** entre os alunos, e aqui serão considerados à princípio **apenas os seus vértices**. Dado que pode ocasionar um custo computacional significativamente grande dependendo do número de alunos, ele não será computado com suas arestas.

Supomos que queremos dividir esses 11 alunos entre 2 professores.

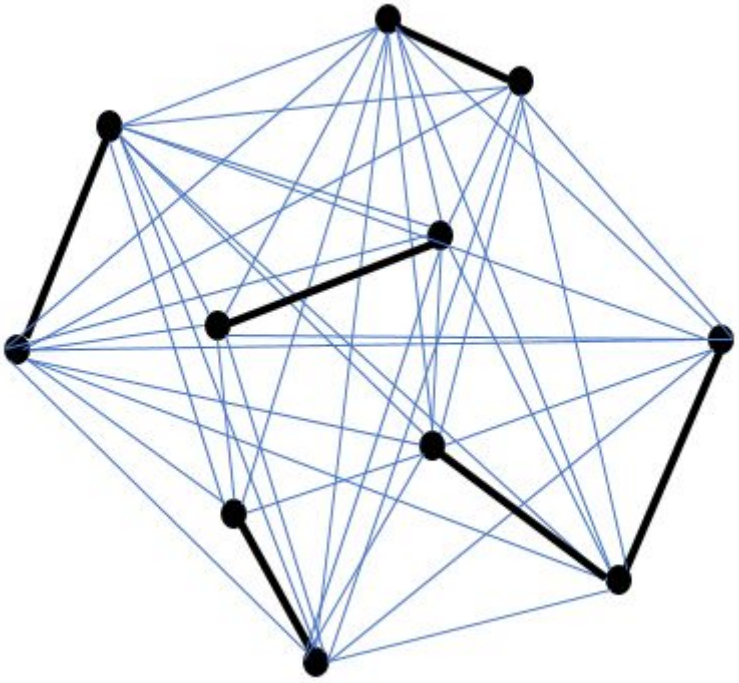
n = 11; #número inicial de subgrupos

k = 2. #número de professores (número final de subgrupos)

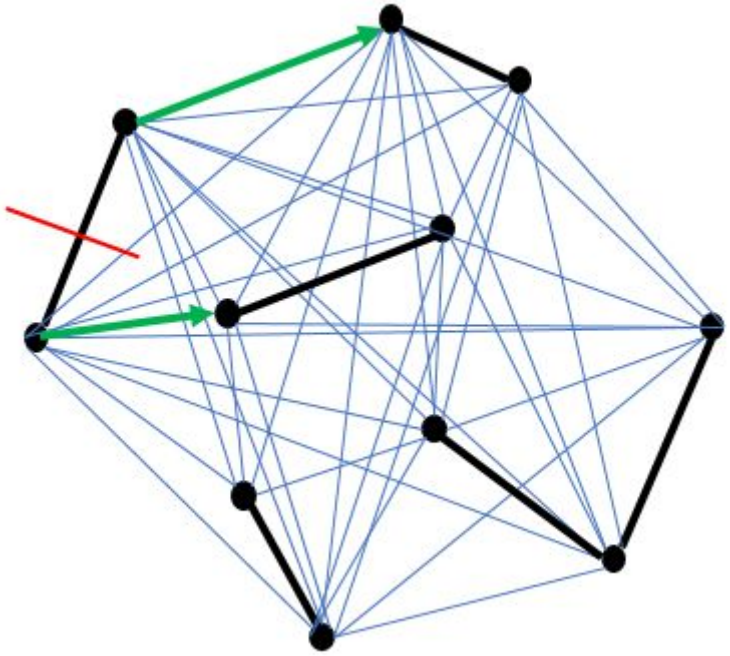
Passo 1: Realizar o matching no grafo, na ordem crescente de arestas por pesos. Ou seja, formam-se duplas entre os alunos com os temas mais próximos. A cada passo, verifica-se a condição necessária para dar sequência ao algoritmo: $k < n$.



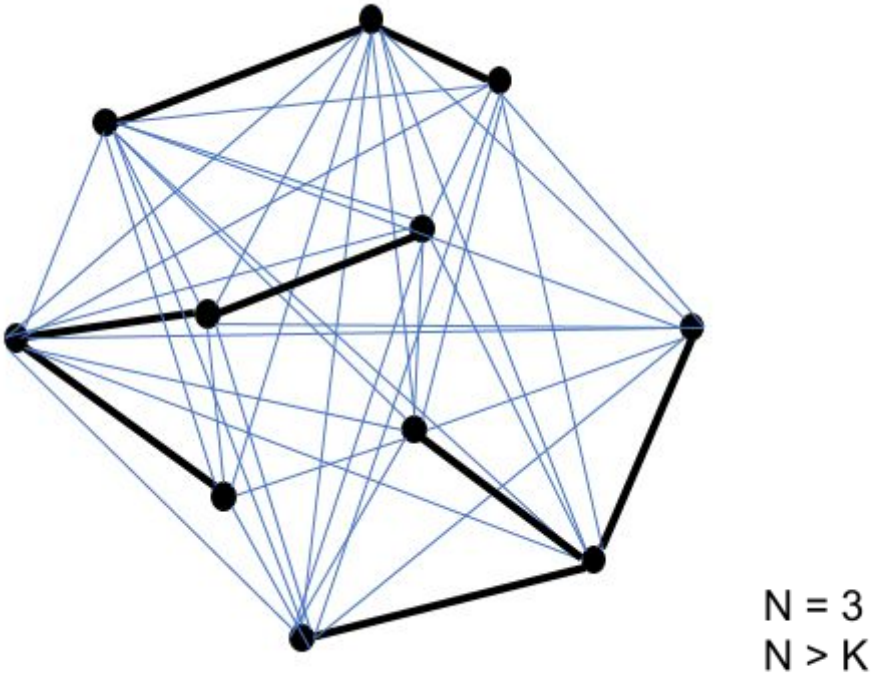
Observe o vértice restante, que por motivo de paridade ficou desconexo. Esse vértice será ligado ao vértice mais próximo do subgrafo mais próximo, formando o primeiro trio.



Passo 2: Deve-se desmembrar o subgrafo de menor grau (ou seja, neste caso: 2) de maior grau de diferença, e distribuir seus vértices nos subgrafos de menor grau mais próximos.

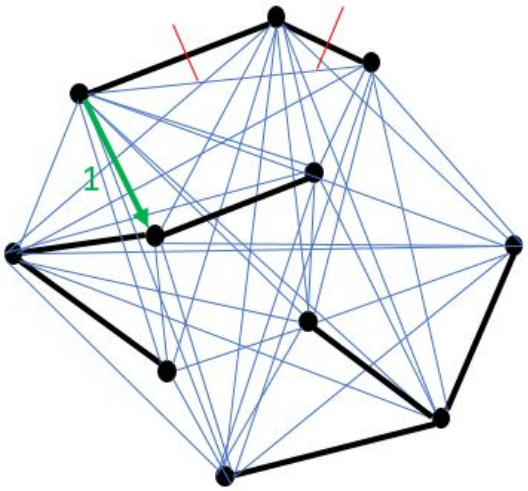


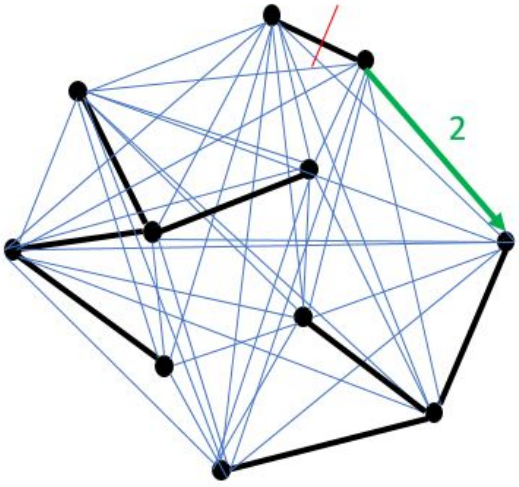
Fazendo este passo reduzimos o número de subgrupos de 5 para 4. E se continuarmos até extinguirmos as duplas (sempre verificando $k < n$) obtemos:

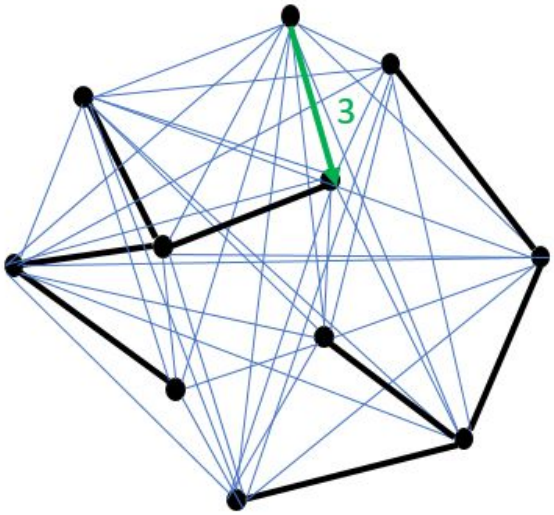


Até então, obtivemos 3 subgrupos, que seria a solução para o caso k = 3. No entanto, k = 2, e k < n = 3. Seguimos para o próximo passo.

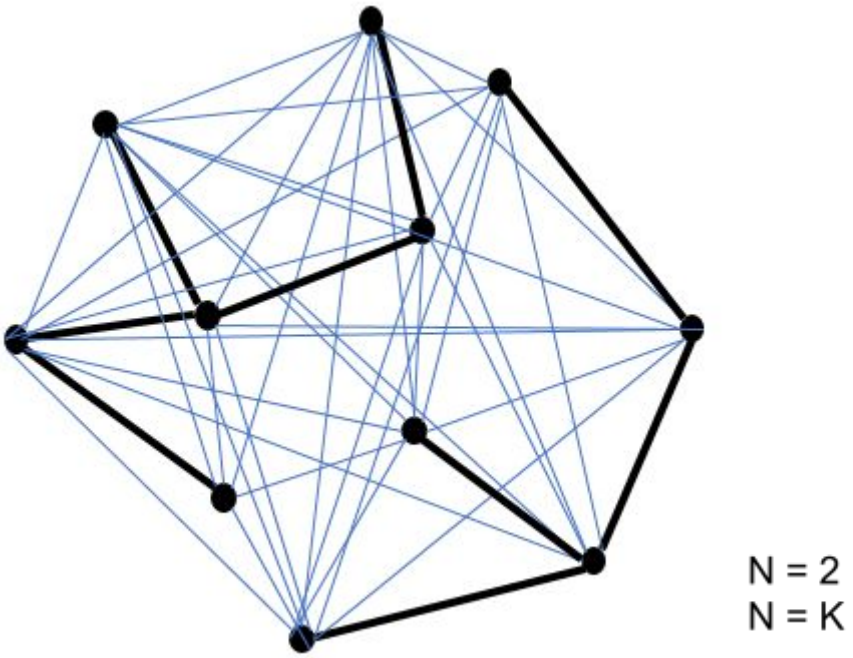
Passo 3: Desmembramos o subgrafo de menor grau (3) e distribuímos os seus vértices componentes entre os subgrupos restantes, na ordem crescente de grau do subgrafo, e crescente de grau de diferença.







Finalmente, k = n, e o algoritmo retorna o modelo final: 2 grupos com 5 e 6 integrantes divididos entre 2 professores.



ALGORITMO

```
funcao Partição (int k, Lista<Aluno> alunos):
        subgrafos = Lista<Grafo>;
        para aluno em alunos:                //cada aluno é um subgrafo inicialmente
                subgrafo = Grafo([aluno], [ ]);  //dupla de (vértices, arestas)
                subgrafos.add(subgrafo);
        fim para;
        enquanto (tamanho(subgrafos) > k) faça:
                ordenar(subgrafos, tamanho(subgrafos) crescente,
                                        diferença(subgrafos) decrescente); //diferenca medir
                                métricas aluno subgrupo
                s = subgrafos.getPrimeiro();      //remove e retorna o primeiro da lista
                para aluno em s.getVertices():   //vértices do primeiro subgrafo
                        sub = maisProximo(subgrafos, aluno)
                        sub.add(aluno)       //adiciona aluno ao grupo mais próximo
                fim para
        retorna subgrafos
```