Felipe Ferreira Carvalho Silva
Lorena Kérollen Botelho Tavares
Rodrigo Pinto Herculano

RELATÓRIO DA SEGUNDA FASE DO TRABALHO DE ALGORITMOS EM GRAFOS - PROBLEMA EM GRAFOS

LAVRAS - MG
Julho de 2018

SUMÁRIO

INTRODUÇÃO

Motivação:

A montagem de grupos de estudo é um desafio para docentes, visto que a resolução dos problemas propostos aos estudantes requer habilidades distintas. Deste modo, é desejável que estes grupos de alunos sejam o mais heterogêneo possível, para que as características dos indivíduos envolvidos crie uma sinergia e torne o processo de aprendizado construtivo e eficaz.

Definição:

O problema proposto pode ser definido da seguinte maneira: cada estudante u possui uma nota p_u , que indica o grau de aptidão para a resolução do problema proposto. O total de grupos de estudantes a ser formado é igual a g. Os grupos possuem limites mínimos e máximos de aptidão dados por L e U , respectivamente. Cada estudante deve pertencer a apenas um grupo. A relação entre cada par de estudantes u e v é dado pelo valor $d_{uv} \in \mathrm{R}$, que quantifica as diferenças das características entre os dois estudantes relativas à resolução do problema proposto pelo docente. Nosso objetivo é criar g grupos de estudantes que maximize o somatório das diferenças entre os estudantes escolhidos a pertencer a cada grupo.

Contextos de aplicação:

-> Pacotes turísticos

Nesse contexto, cada pacote seria um grupo, e os vértices os pontos turísticos, a relação entre os pontos turísticos diria respeito à diferença de suas atrações.

O objetivo aqui seria montar pacotes o mais heterogêneo possível para que os turistas não tenham atrações repetidas durante a visitação.

-> Empresas

Nesse contexto, uma equipe da empresa seria um grupo, os vértices os empregados da empresa, e a relação entre eles o quão bem eles trabalham juntos de acordo com suas especificações.

Dessa forma, visando a maior produtividade da empresa, formaríamos grupos onde os empregados se relacionam melhor.

ALGORITMO PROPOSTO

```
1.  procedure Algoritmo(G) {     G = (V,E), |V| = n, |E| = m, S = Grupos, v = qtdGrupos
2.      maioresArestas ← v maiores arestas ∈ G, sem vértices repetidos
3.      S = [ v ];                                                    # vetor de
grupos(S)
4.      for each grupo ∈ S:
5.          grupo.arestas ← {maioresArestas[ i ]};          # recebendo as  maiores
arestas
6.
7.      for each grupo ∈ S:
8.          while grupo.somatorioAptidao <= grupo.limiteInferior  do:
9.              Encontre a aresta (j,k) ∈ E, j ∈ grupo, k ∈ grupo.adjacentes, d_jk maximo;
10.             if k.aptidao + grupo.somatorioAptidao < grupo.limiteSuperior:
11.                 grupo.somatorioAptidao += k.aptidao
12.                 grupo ← grupo U {k};
13.             else:
14.                 grupo.adjacentes ← grupo.adjacentes \ {k}
15.         while grupo.somatorioAptidao < limiteSuperior do:
16.             Encontre a aresta (j,k) ∈ E, j ∈ grupo, k ∈ grupo.adjacentes, djk maximo;
17.             if k.aptidao + grupo.somatorioAptidao < grupo.limiteSuperior:
18.                 grupo.somatorioAptidao += k.aptidao
19.                 grupo ← grupo U {k};
20.             else:
21.                 grupo.adjacentes ← grupo.adjacentes \ {k}
22. }
```

Explicações do algoritmo

1. Na linha 2 é onde pegamos as v maiores arestas do grafo, sendo v é a quantidade de grupos exigidas pelo problema. No código fonte, utilizamos o método de ordenação quick-sort para ordenarmos as arestas do grafo de forma decrescente de modo que sempre pegaríamos as maiores arestas possíveis (sem repetir vértices entre as arestas).
2. Nas linhas 4 e 5 iniciamos os v grupos com cada um contendo uma das v maiores arestas. No código fonte foi criada, além da classe “Grafo” da etapa anterior, uma classe “Grupo” para armazenar os grupos, contendo os seguintes atributos:
    - limInferior -> limite inferior do somatório das aptidões dos vértices;
    - limSuperior -> limite superior do somatório das aptidões dos vértices;
    - somaAptidao -> somatório das aptidões dos vértices;
    - somaArestas -> somatório dos valores das arestas;
    - qtdVertices -> quantidade de vértices pertencente ao grupo;
    - qtdArestas -> quantidade de arestas pertencente ao grupo;

- arestas -> vetor de arestas (i, j, d_{ij});
- vertices -> vetor de vértices pertencentes ao grupo.

3. Na linha 7 iniciamos o preenchimento dos grupos. Da linha 8 a 14 adicionamos os vértices adjacentes aos vértices que já estão no grupo, maximizando o somatório dos pesos das arestas, tendo como limitante o limite inferior do grupo (limInferior). E da linha 15 a 21, executamos o mesmo algoritmo da linha 8 a 14 tendo como limitante o limite superior (limSuperior). Optamos por executar o mesmo algoritmo duas vezes para que houvesse uma melhor distribuição dos vértices para os grupos, sem que nenhum grupo seja prejudicado e o somatório das aptidões (somaAptidao) não fique abaixo do limite inferior, nem ultrapasse o limite superior.
4. No código fonte, no arquivo “grupo.py”, temos 6 métodos que fazem o preenchimento dos grupos, sendo que dois são para a matriz de adjacência, dois para matriz de incidência e dois para a lista de adjacência, seguindo o raciocínio do tópico 3.
5. Informações referentes ao grupos também estão armazenadas na classe “Grafo” da seguinte forma:
    - qtdVertices -> quantidade de vértices do grafo;
    - qtdArestas -> quantidade de arestas do grafo;
    - aptidao -> vetor que contém as aptidões dos vértices, a posição da aptidão indica a qual vértice ela pertence;
    - inseridos -> vetor booleano com qtdVertices posições, iniciado com “False”, que indica se determinado vértice foi inserido ou não em algum grupo, se sim ele recebe “True” na posição do vértice;
    - maioresArestas -> vetor que contém as maiores arestas do grafo, sem vértices repetidos;
    - limites -> vetor que contém os limites de cada grupo, cada posição contém o limite inferior e superior do grupo daquela posição;
    - qtdGrupos -> quantidade de grupos indicada pelo arquivo de entrada do usuário.

Explicação do algoritmo principal

Após a escolha das maiores arestas do grafo, aplicamos um algoritmo que se assemelha ao algoritmo de Prim, onde para cada grupo, percorremos todos os vértices adjacentes possíveis (a posição dele no vetor inseridos de “Grafo” é “False”) ao grupo buscando a maior aresta que os liga. Dessa forma, verificando qual vértice fará parte do grupo em questão.

Após encontrarmos a maior aresta (i, j), adicionamos o vértice j ao grupo e todas as arestas de j que se ligam aos vértices já pertencentes ao grupo. Repetimos o algoritmo enquanto o somatório das aptidões for menor ou igual ao limite inferior (limInferior).

O algoritmo é repetido utilizando agora como condição de parada o limite superior (limSuperior).

TESTES COMPUTACIONAIS

➢ Matriz de Adjacência:

Instância	*bkv* (melhor valor conhecido)	*Valor obtido*	*Tempo de execução* (em segundos)	*GAP* (%)
P1	224964.8	200137.7	0.809318065643	-11.0
P2	204624.4	197072.8	0.756078004837	-3.6
P3	198937.2	192738.3	0.796390056610	-3.1
P4	225683.2	206005.2	0.727775096893	-8.7
P5	195521.0	186712.8	0.873157978058	-4.5
P6	555993.1	498811.3	5.638782024380	-10.2
P7	511107.9	489510.3	5.778824090960	-4.2
P8	497652.2	493486.1	5.866787195210	-0.8
P9	522604.8	505715.0	5.550273895260	-3.2
P10	484331.0	473240.5	6.292380094530	-2.2

GAP - percentual do valor obtido em relação a melhor solução conhecida (bkv)

➢ Matriz de Incidência:

Instância	*bkv* (melhor valor conhecido)	*Valor obtido*	*Tempo de execução* (em segundos)	*GAP* (%)
P1	224964.8	200137.7	421.186425924	-11.0
P2	204624.4	197072.8	424.022807121	-3.6
P3	198937.2	192738.3	477.187173843	-3.1
P4	225683.2	206005.2	468.668622016	-8.7
P5	195521.0	186712.8	466.542357206	-4.5
P6	555993.1	498811.3	7334.60764790	-10.2

P7	511107.9	489510.3	7659.57942485	-4.2
P8	497652.2	493486.1	7817.28427505	-0.8
P9	522604.8	505715.0	7146.29093694	-3.2
P10	484331.0	473240.5	7954.84224009	-2.2

➢ Lista de Adjacência:

Instância	*bkv* (melhor valor conhecido)	*Valor obtido*	*Tempo de execução* (em segundos)	*GAP* (%)
P1	224964.8	200159.6	1.53096318245	-11.0
P2	204624.4	197893.8	0.75734519958	-3.2
P3	198937.2	193513.8	1.20289707184	-2.7
P4	225683.2	206469.2	1.24819111824	-8.5
P5	195521.0	188160.0	1.42553997040	-3.7
P6	555993.1	500038.5	13.3423810005	-10.0
P7	511107.9	489402.0	18.6219871044	-4.2
P8	497652.2	495640.4	17.8198559284	-0.4
P9	522604.8	506434.4	17.1172769070	-3.0
P10	484331.0	475047.8	14.2721450329	-1.9

CONCLUSÃO

Em relação aos testes computacionais, podemos constatar que as estruturas mais indicadas para a resolução do problema proposto são matriz de adjacência e lista de adjacência, por conta de suas formas de armazenamento, pois diferente da matriz de incidência, onde a quantidade de linhas é a quantidade de arestas do grafo, o percorrimento é eficiente, exigindo um menor poder computacional, uma vez que na matriz de incidência, a cada busca da maior aresta possível, é necessário que todas as linhas da matriz sejam percorridas.

Nossa dificuldade maior foi chegar a um consenso sobre como faríamos as divisões dos grupos exigidos respeitando os limites inferior e superior de cada um, e ainda maximizando os pesos das arestas de cada grupo. Um outro problema que encontramos, após desenvolvermos a ideia inicial do algoritmo, foi adaptá-lo para as estruturas de dados exigidas, uma vez que a forma de armazenamento mudaria as verificações necessárias para o atendimento das restrições.

Podemos concluir com a realização deste trabalho que quando conseguimos modelar problemas utilizando teoria em grafos, conseguimos obter uma grande fonte de algoritmos que nos auxiliam na resolução do problema inicial, mesmo que eles não sejam exatamente iguais. Dessa forma, após o curso de algoritmos em grafos, nos sentiremos mais preparados quando nos depararmos com problemas que aparentemente não tem solução conhecida.

REFERÊNCIAS BIBLIOGRÁFICAS:

- Goldbarg, M., Goldbarg, E. (2012). Grafos: conceitos, algoritmos e aplicações. 1a ed., Elsevier: Rio de Janeiro, 622 p., ISBN 9788535257168.
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2012). Algoritmos: teoria e prática. 3a ed., Elsevier: Rio de Janeiro, 944 p., ISBN 9788535236996.