Trabalho final da disciplina

Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciência da computação

Projeto e analise de algoritmos

Aluno: João Marcos Martins da Costa Cota

Modelagem, algoritmo exato e plano de experimentos.

Encontrar cliques em redes de colaboração, estudo de caso dos 30 anos do SBBD

---

1 Descrição do problema

O Simpósio Brasileiro de Bancos de Dados (SBBD) é o evento oficial de bancos de dados da Sociedade Brasileira de Computação (SBC). Segundo informações do site da SBC é o maior evento da América Latina para apresentação e discussão de resultados de pesquisas e aplicações na área de bancos de dados reunindo mais de 400 pessoas nos últimos anos dentre pesquisadores, alunos e profissionais da área.

Dada a relevância do evento, em 2016, ele completou 30 anos. Devido ao tempo de existência e participação desses pesquisadores, alunos e profissionais da área foi possível obter uma base de dados notável das participações e dos trabalhos ali apresentados. Então é possível analisar a colaboração entre esses autores criando uma representação na forma de um grafo.

1.1 Modelagem do problema em Grafos

Seja um grafo G = (V, E) não direcionado, V representa o conjunto de vértices e E o conjunto de arestas. Os vértices são os autores de vários trabalhos e as arestas conectam vários autores de um mesmo trabalho, formando assim a rede de colaboração. Autores que participaram de apenas um único trabalho em todo evento do SBBD, portanto não estarão relacionados a outros autores. A tabela abaixo mostra um exemplo:

Tabela 1 - Relacionamento entre autores e trabalhos

Trabalho	Autores participantes
1	2
1	3
2	4
2	5
2	6
3	1
4	5
4	11
5	11
7	8

Isso é representado pelo seguinte grafo:

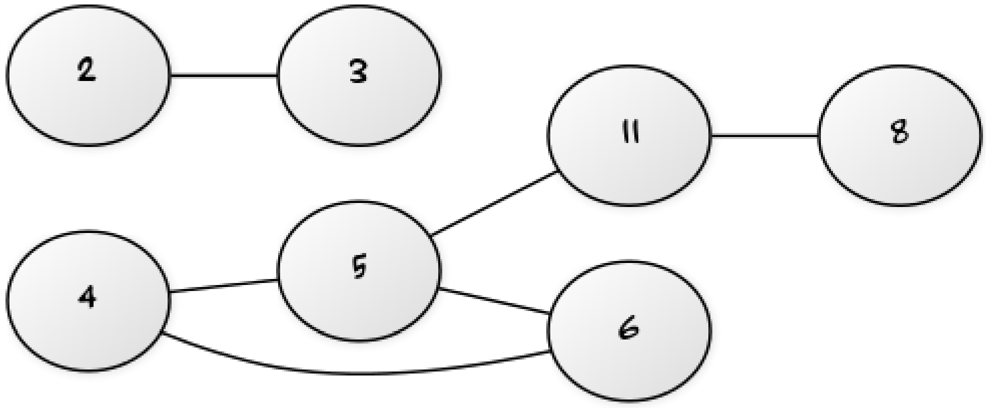

Figura 1 - Conjunto modelado em forma de Grafo

Dessa forma, pode-se observar pelo grafo, que o conjunto de vértices e arestas possuem um clique máximo, que é representado pelos vértices S = {4, 5, 6}. Esse clique máximo representa que esses autores trabalharam juntos várias vezes. Enquanto o conjunto dado por S' = {2, 3}; indica que os autores apenas participaram juntos e possivelmente uma única vez no evento.

2 Implementação do algoritmo exato (*baseline*)

Foram selecionados dois algoritmos exatos (PROSSER, 2012) para o problema do clique máximo. O primeiro algoritmo, MC (FAHLE, 2002), possui uma forma simplista, que computa todos cliques no grafo e ao final define qual o maior deles. O segundo algoritmo, MCQ (TOMITA; SEKI, 2003), apresenta uma forma melhorada do algoritmo anterior ao utilizar um princípio chamado de colaboração de vértices para então descobrir o limite superior do clique máximo (SUYUDI et al., 2014).

O pseudocódigo do Algoritmo MC é apresentado logo a seguir. A solução do algoritmo é armazenada em *cliqueVertice* e o tamanho do clique máximo em *cliqueTam.* O conjunto *C* contém um clique em crescimento e o conjunto *P* contém os candidatos para a solução. Quando um vértice *v* é selecionado em P, sendo adicionado em C, cria então um novo conjunto de vértices candidatos, *novoP*, para qual os vértices serão adjacentes ao vertice *v*. Ao ser notado que o tamanho do conjunto *C* ficou maior do que a solução atual, *cliqueVertice* e *cliqueTam* são atualizados e ao final da execução tem o tamanho e os vértices do clique máximo.

O pseudocódigo do Algoritmo MCQ é apresentado logo a seguir. Como já mencionado, ele é baseado no primeiro algoritmo onde trata a busca pelo clique máximo de forma semelhante, com o diferencial de buscar por um limite superior. A função *EXPAND* é semelhante ao algoritmo MC e ele possui uma função extra, *NUMBERSORT* que determina a coloração dos vértices e armazena esses vértices num conjunto *colour*. A função *NUMBERSORT* é chamado no início da função *EXPAND* que recebe o conjunto *P* atualizado e entrega a saída o mesmo conjunto *P*, já com os vértices ordenados em ordem crescente de acordo com sua colocação.

Os algoritmos *baselines* foram testados utilizando de grafos da coleção DIMACS, muito comuns para análise de *benchmark* para clique máximo. Na Tabela 2 fica evidente a relação dos algoritmos MC e MCQ com os grafos utilizados, é importante salientar que os algoritmos encontraram o clique máximo do mesmo tamanho do que informado pela base coleção DIMACS, isso prova que os *baselines* são capazes de detectar o clique máximo.

- Algoritmo MC

```
int cliqueTam
List cliqueVertice

função expand(Graph g, List C, List P)
    para i ← |P|-1 até 0 faça
        se |C| + |P| ≤ cliqueTam então return
        fim se
        int v ← P(i)
        C ← v
        List novoP
        para u ← 0 até |P| faça
            se u ∈ adj(v) então
                novoP ← u
            fim se
        fim para
        se novoP = ∅ e |C| > cliqueTam então
            cliqueVertice ← C
            cliqueTam ← |C|
        fim se
        se novoP != ∅ então
            EXPAND(g, C, novoP)
        fim se
        C ← remove(v)
        P ← remove(v)
    fim para
fim função
```

Tabela 2 - Testes Preliminares

Grafo	Vértices	Arestas	Clique Máximo	Tempo MC	Tempo MCQ
Hamming8-4	256	20864	16	15601	28658
Mann-A9	45	918	16	1469	2719
Johnson8-4-4	70	1825	14	313	469

- Algoritmo MCQ

```
int cliqueSize
List cliqueVertex

função expand(Graph g, List C, List P)
    NUMBERSORT(g, P)
    para i ← |P|-1 até 0 faça
        se |C| + |P| ≤ cliqueSize então return
        fim se
        int v ← P(i)
        C ← v
        List newP
        para j ← 0 até i faça
            u ← P(j)
            se u ∈ adj(v) então
                newP ← u
            fim se
        fim para
        se newP = ∅ e |C| > cliqueSize então
            cliqueVertex ← C
            cliqueSize ← |C|
        fim se
        se newP != ∅ então
            EXPAND(g, C, newP)
        fim se
        C ← remove(|C|)
        P ← remove(i)
    fim para
fim função

função numberSort(Graph g, List P)
    int colours
    para i ← 0 até |P| faça
        v ← P(i)
        k ← 0
        enquanto adj(v ∈ colour(k)) faça
            k++
        fim enquanto
        colour(k) ← v
        colours ← max(colours, k+1)
    fim para
    P.clear()
    para k ← 0 até colours faça
        para j ← 0 até |colour(k)| faça
            v ← colour(k)(j)
            P ← v
        fim para
    fim para
fim função
```

3 Heurísticas

Pelo emprego de técnicas de *Branch and Bound*, pretende-se determinar os limites superiores e inferiores para tamanho de um clique máximo (JOHNSON, 1973). Desse modo a determinação do limite superior será dado pela coloração dos vértices e o limite inferior pela busca gulosa do clique máximo.

Determinado os valores inferiores e superiores não será necessário buscar por cliques maiores e verificar vértices com grau menor, visto que se o clique máximo será maior ou igual ao limite inferior e não haverá clique maior que o limite superior.

A coloração é fazer com que vértices adjacentes sejam coloridos de formas diferentes, tal que o número cromático é o menor número de cores necessário para colorir todos vértices de um grafo (CORMEN et al., 2001). Assim o número cromático pode ser usado como limite superior.

A busca gulosa parte pelo princípio que o clique máximo pode ser encontrado pelo vértice que possui maior grau. Então a busca se inicia por esse vértice, inserindo-o no conjunto solução e fazendo isso para os próximos vértices que possuem maior grau e não estão no conjunto solução. A cada passo os vértices não adjacentes ao conjunto solução são removidos da busca (SUYUDI et al., 2014).

4 Algoritmo proposto

Para esse trabalho foram desenvolvidos três algoritmos. Eles permitiram efetuar a busca pelo clique máximo, utilizar as heurísticas apresentadas. Permitindo então a comparação dos resultados das heurísticas e da *baseline*.

Os algoritmos propostos foram: *Lower bound Maximum Clique* (LMC), *Upper bound Maximum Clique* (UMC) e *Lower and upper bound Maximum Clique* (LUMC), são algoritmos de funcionamento semelhante, o que os diferenciam é a forma como é calculado o clique. LMC o cálculo leva em consideração o limite inferior. UMC o cálculo leva em consideração apenas o limite superior. LUMC o cálculo leva em consideração ambos limites, superior e inferior.

Os limites são importantes para que não façam comparações desnecessárias, dessa forma, não procurar cliques em vértices com grau menor que o limite inferior ou procurar por cliques que sejam maiores do que o grau superior.

Ambos algoritmos fazem uso da função EXPAND apresentada nos algoritmos MC e MCQ. A análise assintótica para os algoritmos revela complexidade de tempo $O(2^n)$ e complexidade de memória $O(n)$. Com o uso dos limites o melhor caso pode ser $\Omega(n)$.

No algoritmo LMC, o limite superior é encontrado através de uma busca gulosa, de forma que o vértice de maior grau pode ser considerado como parte da solução. Após achado o vértice de maior grau, é construído sua lista de adjacências. Então é realizado a expansão sobre esse vértice, e assim é retornado o tamanho do clique. Possui complexidade $O(n)$ uma vez que só um vértice é expandido.

No algoritmo UMC, o limite superior é determinado por uma busca gulosa do número cromático do grafo. Os vértices são ordenados pelo seu grau, de forma que os que possuem maior grau são coloridos primeiro. Nesse algoritmo é usado *Merge-Sort* que foi omitido no pseudocódigo e as cores são armazenadas na lista *colour*. Desconsiderando a complexidade da ordenação, o algoritmo apresenta complexidade $O(n)$.

Em seguida é apresentado os pseudocódigos dos algoritmos propostos.

- Algoritmo LUMC

```
int upperBound
função search(Graph G)
    List C
    List P
    P ← G.vertex
    int upperBound ← UPPERBOUND(G, C, P)
    int upperBound ← LOWERBOUND(G, C, P)
    para i ← 0 até |P| faça
        v ← P_i
        se G.vertex_{v-1}.degree < lowerBound-1 então
            P ← remove(v)
            i ← i+1
        fim se
    fim para
    EXPAND(G, C, P)
fim função
função EXPAND(Graph G, List C, List P)
    para i ← |P|-1 até 0 faça
        se |C| + |P| ≤ cliqueSize então return
        fim se
        int v ← P(i)
        C ← v
        List newP
        para i ← 0 até |P| faça
            u ← P_i
        se u ∈ adj(v) então
            newP ← u
        fim se
    fim para
    se newP = ∅ AND |C| > cliqueSize então
        cliqueVertex ← C
        cliqueSize ← |C|
    fim se
    se newP /= ∅ AND |C| < upperBound então
        EXPAND(G, C, newP)
    fim se
    C ← remove(v)
    P ← remove(v)
    fim para
fim função
```

- Algoritmo UMC

```
int upperBound
função calcUpperBound(Graph G, List C, List P)
    List[] colour
    para i ← 0 até |G| faça
        colour_i ← 0
    fim para
    SORT(G, P, 0, |P|)
    para i ← |P|-1 até 0 faça
        int v ← P_i
```

```
            int k ← 0
            enquanto colour_k[] ∈ v faça
                k ← k+1
            fim enquanto
            colour_k ← v
            upperBound ← MAX(upperBound, k+1)
        fim para
fim função
```

- Algoritmo LMC

```
int lowerBound
função calcLowerBound(Graph G, List C, List P)
        int maxDegree, vertexMax
        para i ← 0 até |G| faça
            se G_i.degree > maxDegree então
                vertexMax ← G_i
                maxDegree ← G_i.degree
            fim se
        fim para
        List newP
        para i ← 0 até |P| faça
            int u ← P_i
            se u ∈ adj(v) então
                newP ← u
            fim se
        fim para
        newP ← vertexMax
        EXPAND-L(G, C, newP, vertexMax)
fim função
função Expand(Graph G, List C, List P, int v)
        se |C| + |P| ≤ lowerBound então return
        fim se
        C ← v
        List newP
        para i ← 0 até |P| faça
            u ← P_i
            se u ∈ adj(v) então
                newP ← u
            fim se
        fim para
        se newP = ∅ AND |C| > lowerBound então
            lowerBound ← |C|
        fim se
        se newP /= ∅ então
            EXPAND(G, C, newP, newP_size-1)
        fim se
        C ← remove(v)
        P ← remove(v)
fim função
```

5 Experimentos

Os algoritmos implementados foram testados em conjuntos de *benchmark* da coleção de grafos DIMACS e então aplicados sobre a base de dados coletados do SBBD. A tabela 3 mostra informações a respeito do conjunto de grafos DIMACS que são utilizados para validação dos algoritmos implementados.

Tabela 3 - Conjuntos DIMACS analisados

Grafo	Vértices	Arestas	Limite Inferior	Limite Superior	Clique Máximo
C-FAT200-1	200	1534	12	14	12
C-FAT200-2	200	3235	24	24	24
C-FAT200-5	200	8473	58	58	58
C-FAT500-1	500	4459	14	14	14
C-FAT500-2	500	9139	26	26	26
C-FAT500-5	500	23191	64	64	64
HAMMING6-2	64	1824	32	32	32
HAMMING6-4	64	704	2	8	4
JOHNSON8-2-4	28	210	4	6	4
JOHNSON8-4-4	70	1855	14	14	13

Em relação a base do SBBD, essa possui 1039 vértices, 1480 arestas, limite superior de 14 e inferior de 8 e clique máximo de 8.

Em seguida a tabela 4 com os tempos para o clique das bases. Em negrito a base do SBBD.

Tabela 4 – Tempo para clique

Grafo	MC	MCQ	LMC	UMC	LUMC
C-FAT200-1	175	272	327	286	227
C-FAT200-2	156	309	277	169	207
C-FAT200-5	1365	191807	2072	55385	52261
C-FAT500-1	319	423	456	557	365
C-FAT500-2	474	524	524	568	592
C-FAT500-5	3480	27144	4149	267223	277486
HAMMING6-2	2336	4589	1732	1731	1659
HAMMING6-4	221	92	101	162	103
JOHNSON8-2-4	66	65	71	76	103
JOHNSON8-4-4	1054	1347	1138	992	1178
SBBD	**279**	**455**	**414**	**54**	**500**

A tabela 5 mostra a quantidade de acessos aos vértices das bases. Em negrito a base do SBBD.

Tabela 5 - Quantidade de acessos

Grafo	MC	MCQ	LMC	UMC	LUMC
C-FAT200-1	809	2568	405	427	427
C-FAT200-2	1915	1867	958	1590	1590
C-FAT200-5	140787	10763211	70394	5703022	5703022
C-FAT500-1	2149	2027	1075	1119	1119
C-FAT500-2	8363	8231	4182	6011	6011
C-FAT500-5	571777	571635	285889	24266630	24266630
HAMMING6-2	508979	433471	254490	254490	254490
HAMMING6-4	1629	1413	1045	1045	1045
JOHNSON8-2-4	628	397	358	358	358
JOHNSON8-4-4	271999	160879	136000	136000	136000
SBBD	**921**	**793**	**1844**	**91**	**68576**

Em seguida os gráficos com as variações do tempo de acesso, figura 2, e da quantidade de cliques, figura 3, esse em escala logarítmica para facilitar a visualização da variação.

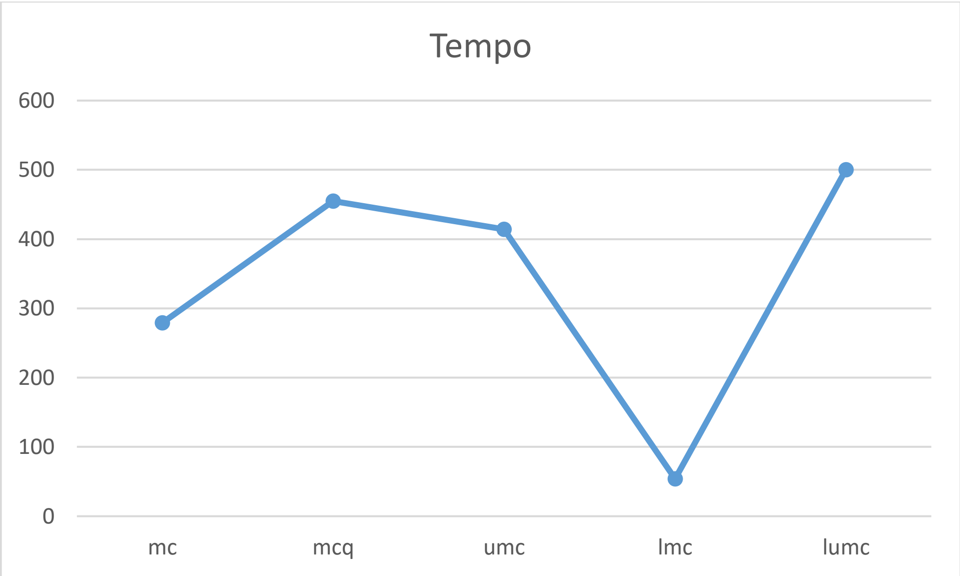

Figura 2 - Tempo para clique

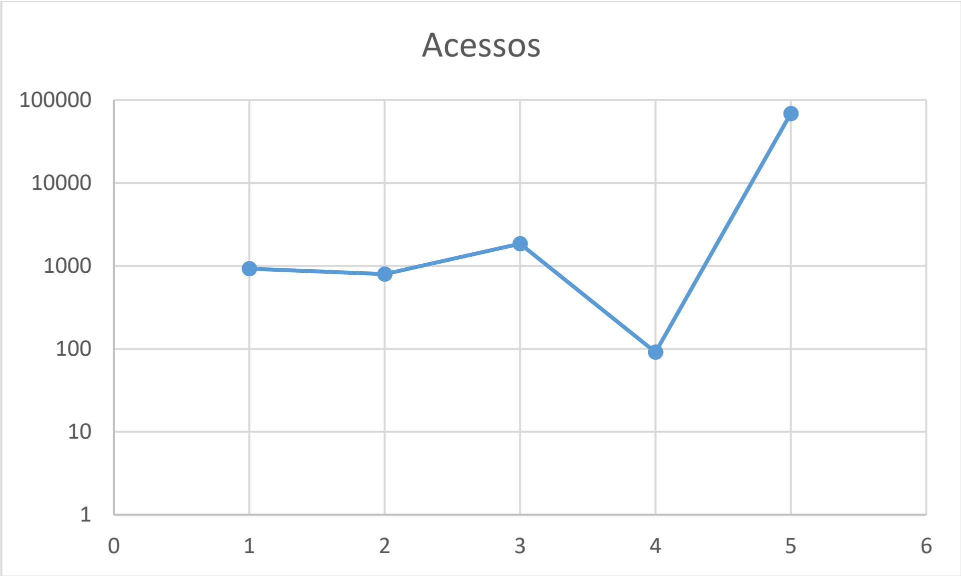

Figura 3 - Quantidade de acessos

A partir da execução dos algoritmos, nota-se que a quantidade média de cliques da rede de colaboração do SBBD é de 3.7, de forma que podemos analisar esse valor como a média de autores que participam juntos de um trabalho.

6 Conclusão

As heurísticas apresentadas apresentam melhoria de modo geral. A aplicação do limite inferior apresenta bons resultados nos casos analisados, resultando em um bom custo benefício dado a complexidade do cálculo guloso do clique máximo. Já a heurística de limite superior em alguns casos apresentou resultados piores que os demais, que pode ser justificado devido a ordenação dos vértices. Os grafos analisados são grafos de tamanho relativamente pequenos, dessa forma, se faz necessário observações em grafos maiores e em vértices ordenados para verificar a possível existência de variação dos resultados.

Outra analise interessante seria buscar pelos outros cliques, não ficando somente no clique máximo. De forma a tentar abranger mais áreas do grafo e realizar mais analises do conteúdo dessa e outras bases que poderão ser analisadas.

De modo geral, os algoritmos implementados para esse trabalho apresentaram melhores resultados que os *baselines* utilizados. Como proposta de trabalho futuro, é a utilização desses algoritmos em outras bases de redes de colaboração e analise em grafos maiores.

Referências

CORMEN, T. H. et al. *Introduction to algorithms*. [S.l.]: MIT press Cambridge, 2001. v. 2. page.66

DIMACS, DIMACS Challenge, Disponível em: < http://dimacs.rutgers.edu/Challenges/ >

FAHLE, T. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: *In 10th Annual European Symposium on Algorithms (ESA 2002), volume 2461 of LNCS*. [S.l.]: Springer Verlag, 2002. p. 485–498. page.33

JOHNSON, D. S. Approximation algorithms for combinatorial problems. In: ACM. *Proceedings of the fifth annual ACM symposium on Theory of computing*. [S.l.], 1973. p. 38–49. page.33

PROSSER, P. Exact algorithms for maximum clique a computational study tr-2012-33. page.33

SUYUDI, M. et al. Solution of maximum clique problem by using branch and bound method. *Applied Mathematical Sciences*, v. 8, n. 2, p. 81–90, 2014. page.33, page.66

TOMITA, E.; SEKI, T. An efficient branch-and-bound algorithm for finding a maximum clique. In: *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science*. Berlin, Heidelberg: Springer-Verlag, 2003. (DMTCS'03), p. 278–289. ISBN 3-540-40505-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1783712.1783736>. page.33