Universidade Federal de Minas Gerais
Ciência da Computação

TP2: Comunidades em Grafos

Projeto e Análise de Algoritmos

Lucas Cunha de Oliveira Miranda
lucmir@dcc.ufmg.br

1 Proposta

O Trabalho Prático propõe a implementação e análise de algoritmos para busca de comunidades em grafos. Uma comunidade é um subconjunto de vértices que apresentam ligações mais densas. Os algoritmos implementados devem ser baseados no conceito de *centralidade* de uma aresta. A centralidade de uma aresta correponde ao número de caminhos mínimos que passam por ela. Os algoritmos para obtenção de caminhos mínimos *BFS* e *Floyd-Warshall* devem ser implementados como parte da solução.

2 Solução

O problema de encontrar comunidades em grafos foi solucionado pelo uso de um método inspirado no trabalho de *Girvan Newman* [1]. O *Algoritmo de Girvan Newman* é um método proposto para detecção de comunidades... baseado no conceito de centralidade (*betweenness*) pseudocódigo

O algoritmo 1 resume a solução detalhada.

Algorithm 1 Método baseado no Algoritmo de *Girvan Newman*

1: calcule centralidade para todas as arestas do grafo
2: **while** número de comunidades no grafo é igual ao desejado **do**
3: remova aresta com maior centralidade
4: recalcule a centralidade para arestas afetadas
5: **end while**
6: **return** comunidades do grafo

2.1 Solução por Breadth First Search

```
1  procedure BFS(G,v):
2      create a queue Q
3      enqueue v onto Q
4      mark v
5      while Q is not empty:
6          t ? Q.dequeue()
7          if t is what we are looking for:
8              return t
```

```
9           for all edges e in G.incidentEdges(t) do
10              o ? G.opposite(t,e)
11              if o is not marked:
12                   mark o
13                   enqueue o onto Q
```

2.2 Solução por Floyd Warshall

```
 1 /* Assume a function edgeCost(i,j) which returns the cost of the edge from i t
 2    (infinity if there is none).
 3    Also assume that n is the number of vertices and edgeCost(i,i) = 0
 4 */
 5
 6 int path[][];
 7 /* A 2-dimensional matrix. At each step in the algorithm, path[i][j] is the sh
 8    from i to j using intermediate vertices (1..k?1).  Each path[i][j] is init
 9    edgeCost(i,j).
10 */
11
12 procedure FloydWarshall ()
13    for k := 1 to n
14       for i := 1 to n
15          for j := 1 to n
16             path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );
```

2.3 Complexidade

Considerando um grafo G com $v = |V|$ vértices e $n = |A|$ arestas.
Para a **Solução por Breadth First Search**...
Para a **Solução por Floyd Warshall**...

3 Implementação

A solução para o problema proposto foi implementada na linguagem *Java* e plataforma *Linux*. Os arquivos do diretório *src* contêm o código da implementação.

Antes de executar os código, é necessário compilá-los, pelo comando:

```
javac -d ./bin *.java
```

Alternativamente, pode ser usado o programa *Makefile*. O seguinte comando executa a compilação:

```
make
```

Os arquivos compilados (com extensão *.class* serão gerados no diretório *bin*).

Para executar o programa, é necessário passar os seguintes parâmetros:

- arquivo contendo as arestas do grafo
- número de *comunidades* que serão obtidas
- número de vértices

- algoritmo usado:
  - b: Breadth First Search
  - f: Floyd Warshall

A execução é feita pelo comando (no diretório *bin*):

```
java Main <arquivo_arestas> <num_comunidades>
  <num_vertices> <algoritmo: b|f>
```

Em execuções com arquivos de entrada muito grandes, talvez seja necessário aumentar o tamanho em memória disponível para a Máquina Virtual Java. Os parâmetros *Xmx* e *Xms* são adicionados, neste caso. Um exemplo de execução usando estes parâmetros é:

```
java Main -Xmx1024m -Xms1024m <arquivo_arestas> <num_comunidades>
                              <num_vertices> <algoritmo: b|f>
```

O código que implementa a solução foi compilado e executado corretamente na máquina *jaguar*, do departamento de computação.

4 Análise Experimental

Para verificar o comportamento do algoritmo, foram realizados alguns experimentos. Foi usado um *script* gerador de entradas (grafos e categorias) aleatórias segundo o padrão definido na especificação. Foram geradas entradas variando o número de vértices (e, consequentemente, o número de arestas) e deixando o número de *clusters* fixo (em 100). Posteriormente, foi analisada a influência do parâmetro número de *clusters* no desempenho do algoritmo. O número de vértices foi variado de 1000 a 4500. O gráfico seguinte apresenta os resultados obtidos:

Referências

[1] Girvan newman algorithm. http://en.wikipedia.org/wiki/Girvan