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* [3]. O *Algoritmo de Girvan Newman* é um método proposto para detecção de comunidades em redes (grafos). Uma comunidade consiste em um subconjunto de nodos ligados por uma densidade grande de arestas. O método usado é baseado no conceito de centralidade (*betweenness*). Para uma aresta i, a *centralidade* é definida como o número de caminhos mínimos entre pares de vértices que passam por i. A *centralidade* de uma aresta é uma métrica que indica sua influência na estrutura da rede. Com base nesta ideia, o algoritmo de *Givan Newman* propõe uma abordagem iterativa para detectar comunidades. Primeiramente, calcula-se a *centralidade* de cada aresta. Em seguida, a aresta com maior valor da métrica calculada é removida. Após a remoção, verifica-se o número de comunidades existentes na nova configuração do grafo. O processo é repetido sucessivamente, recalculando os valores de *centralidade* e removendo arestas, até que o grafo resultante contenha o número de comunidades desejado.

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
```

O grafo foi representado e implementado usando a estrutura de dados lista de adjascência, onde cada vértice da lista contém uma lista associada com os identificadores dos vértices com os quais são formadas arestas.

Foram feitas duas implementações do algoritmo de *Girvan Newman*. As implementações seguiram o algoritmo 1, variando apenas o método para cálculo dos caminhos mínimos (necessário para determinar a centralidade das arestas). Foram adotados os algoritmos para obtenção de caminhos mínimos *Breadth First Search* e *Floyd Warshall*.

2.1 Solução por *Breadth First Search*

Breadth First Search (ou *busca em largura*), em teoria de grafos, é um algoritmo para execução de buscas em árvores [1]. O método expande e examina sistematicamente todos os nós da árvore. Todos os nós filhos obtidos pela expansão de um nó são adicionados a uma fila (FIFO). Os nós examinados são retirados da fila e marcados. O algoritmo é aplicado para obtenção do caminho mínimo entre pares de vértices, como detalhado no algoritmo 2.

Algorithm 2 Algoritmo *Breadth First Search*

```
escolha uma raiz s de G
marque s
insira s na fila F
while F não está vazia faça do
    seja v o primeiro vértice de F
    for w ∈ listaDeAdjacência de v do
        if w não está marcado então then
            visite aresta entre v e w
            marque w
            insira w em F
        else
            if w ∈ F then
                visite aresta entre v e w
            end if
        end if
    end for
    retira v de F
end while
```

2.2 Solução por Floyd Warshall

O algoritmo de *Floyd Warshall* é um método usado no cálculo do caminho mais curto entre todos os pares de vértices em um grafo orientado (com direção) e valorado (com peso) [2]. O algoritmo pode ser adaptado para grafos não orientados e não valorados (replicando as arestas direcionadas e atribuindo pesos iguais para as arestas).

O algoritmo recebe como entrada uma matriz de adjacência que representa o grafo. No caso do trabalho prático, uma matriz é construida, atribuindo-se o valor 1 para posições representando arestas existentes e *infinito* para as demais. O valor de um caminho entre dois vértices é a soma dos valores de todas as arestas ao longo desse caminho. O método calcula, para cada par de vértices, o menor de todos os caminhos entre os vértices.

O algoritmo de *Floyd Warshall* é baseado no conceito de *Programação Dinâmica*. O cálculo de um caminho mínimo entre dois vértices é executado uma única vez. Sempre que um caminho já calculado faz parte de um caminho para o qual o cálculo está sendo executado, o valor já obtido é reaproveitado.

O algoritmo 3 resume o que foi explicado.

Algorithm 3 Algoritmo de *Floyd Warshall*

```
1: inicializa a matriz (nxn) path com 1 para arestas existentes e INF caso contrário
2: for k = 0 to n do
3:     for i = 0 to n do
4:         for j = 0 to n do
5:             sum = path[i][k] + path[k][j]
6:             path[i][j] = min(path[i][j], sum)
7:         end for
8:     end for
9: end for
```

2.3 Complexidade

Considerando um grafo G com $v = |V|$ vértices e $n = |E|$ arestas. A solução proposta envolve um ciclo de iterações em que arestas são removidas até que o grafo possa ser dividido em k comunidades. A operação de selecionar e remover a aresta de maior centralidade, tem custo, no máximo $O(n)$ (considerando que os valores de centralidade já são conhecidos).

A operação de cálculo da centralidade de todas as arestas do grafo depende do algoritmo implementado para obtenção dos caminhos mínimos.

Para a **Solução por Breadth First Search**, a complexidade do cálculo do caminho mínimo é $O(|E| + |V|)$, já que todos os vértices e arestas são explorados, no pior caso. Dependendo da densidade do grafo, esta complexidade pode variar de $O(|V|)$ para $O(|V|^2)$. Este procedimento é executado variando o vértice inicial, para cada vértice do grafo. Portanto, a complexidade de se calcular a centralidade por esta solução é, no pior caso, $O(|V|^3)$.

Para a **Solução por Floyd Warshall**, todos os v^2 caminhos mínimos (para todo i e j) são obtidos por operações constantes executadas em três *loops* aninhados, que variam de 1 a n (ver algoritmo 3). Portanto, a complexidade do cálculo é cúbica $O(|V|^3)$.

Por fim, resta a operação de verificar quantas comunidades existem e de identificá-las. Para isso, é feita uma busca em largura no grafo. Quando um vértice é visitado, este é marcado. Se ao fim da busca, algum vértice não recebeu a marcação, significa que este faz parte de outra comunidade. O processo é repetido para o vértice desmarcado, e assim, sucessivamente, até que todos os vértices estejam marcado e as comunidades identificadas. O custo de efetuar este procedimento é $O(|E| + |V|)$, já que todos os vértices e arestas são visitados.

A complexidade do algoritmo todo corresponde ao custo (que prevalece) de se computar os valores de centralidade para todas as arestas, que é $O(|V|^3)$.

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*):

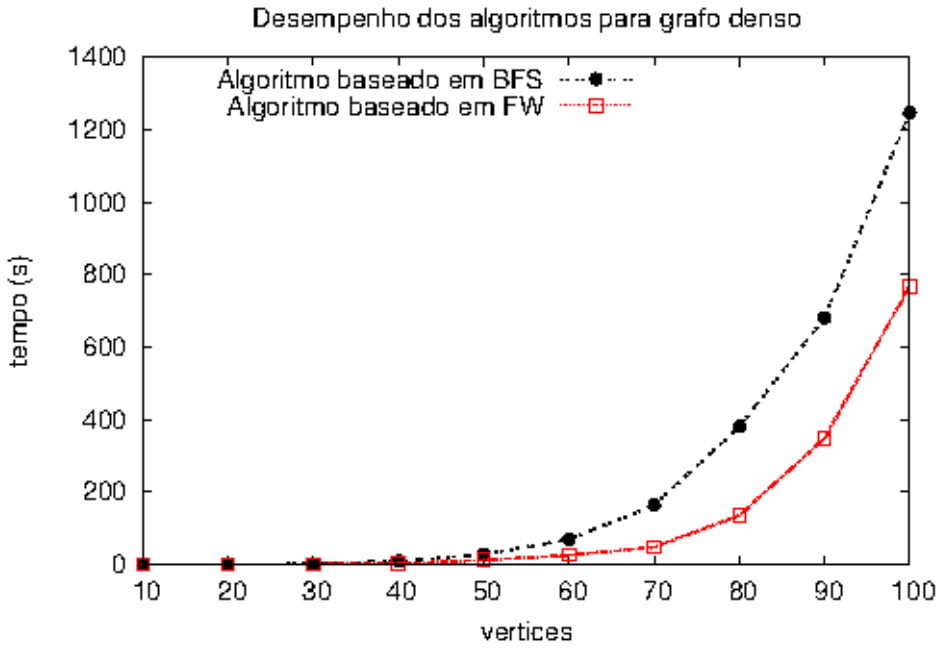
```
java Main <arquivo_arestas> <num_comunidades>
  <num_vertices> <algoritmo: b|f>
```

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

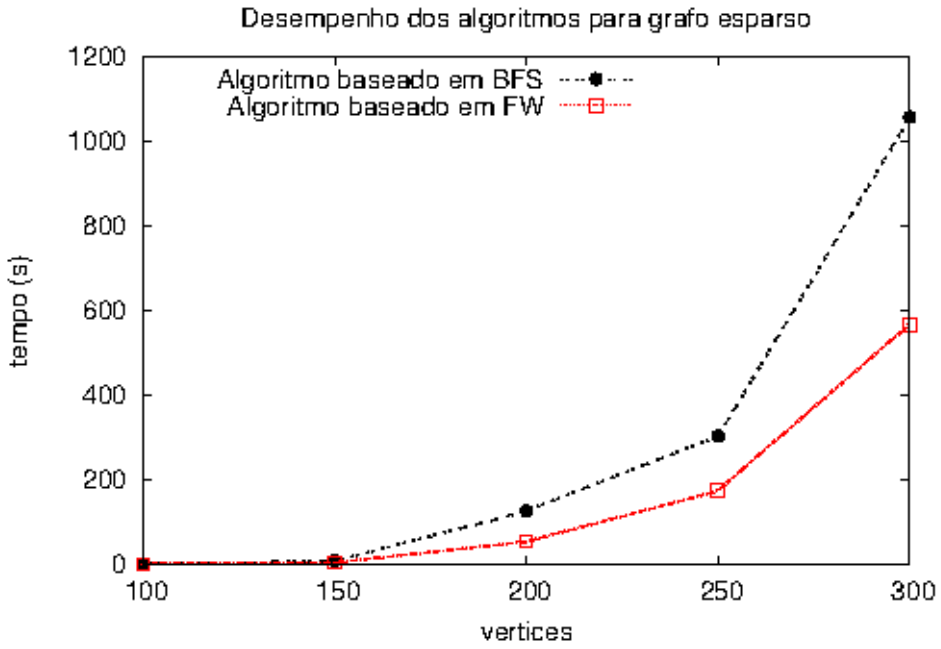
4 Análise Experimental

Para verificar o comportamento dos algoritmos, foram realizados alguns experimentos. Foi usado um *script* gerador de entradas aleatórias segundo o padrão definido na especificação (o *script* foi disponibilizado pelos monitores da disciplina). Foram geradas entradas variando o número de vértices (e, consequentemente, o número de arestas) e fixando o número de *comunidades* geradas (em 10).

Primeiramente, gerou-se um grafo denso, a partir da função de distribuição de arestas $|E| = \frac{|V|^2}{1.5}$). O número de vértices foi variado de 10 à 100. O gráfico seguinte apresenta os resultados obtidos para as duas soluções:



Outro experimento executado foi a geração de um grafo esparso, a partir de uma função de distribuição seguindo uma *power law*. O número de vértices foi variado de 100 à 300. O gráfico seguinte apresenta os resultados obtidos para as duas soluções:



Pela observação dos dois gráficos, é fácil constatar que o algoritmo de *Floyd Warshall* foi mais eficiente para todos os testes. Uma justificativa para isso é que o algoritmo BFS não possui a vantagem do reaproveitamento de cálculos, concedida pela programação dinâmica.

Apesar das diferenças de tempo de execução, o comportamento das curvas dos dois algoritmos foi semelhante. Nota-se que as curvas são semelhantes a graficos originados de funções cúbicas. O que foi sugerido pelo cálculo de complexidade das soluções.

Outra observação importante é que a densidade do gráfico tem muita influência no tempo de execução dos algoritmos. Os métodos gastaram, em média, 1000 segundos para executarem sobre um gráfico denso com 100 vértices. Porém, para um grafo esparso com 100 vértices, a execução foi quase que imediata.

Referências

[1] Breadth first search algorithm. http://en.wikipedia.org/wiki/Breadth-first_search.

[2] Floyd warshall algorithm. http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm.

[3] Girvan newman algorithm. http://en.wikipedia.org/wiki/Girvan-Newman_algorithm.