INSTITUTO DE CIÊNCIA E TECNOLOGIA - ICT

BACHARELADO EM CIÊNCIA E TECNOLOGIA

ALGORITMOS E ESTRUTURAS DE DADOS II
TRABALHO 3 – MEDIDAS DE CENTRALIDADE EM GRAFOS

Aluno: Pedro Spoljaric Gomes – R.A.: 112344

Orientadora: Profª Drª Lilian Berton

São José dos Campos

21 de junho de 2017

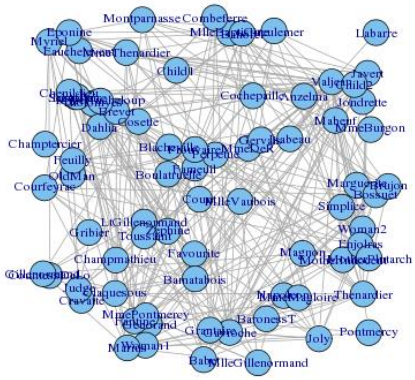
1. INTRODUÇÃO

O trabalho consiste em analisar dois arquivos de estrutura de grafos, considerando grau e medidas de centralidade. Além disso, o objetivo é ranquear os vértices de acordo com tais informações, do maior par ao menor valor.

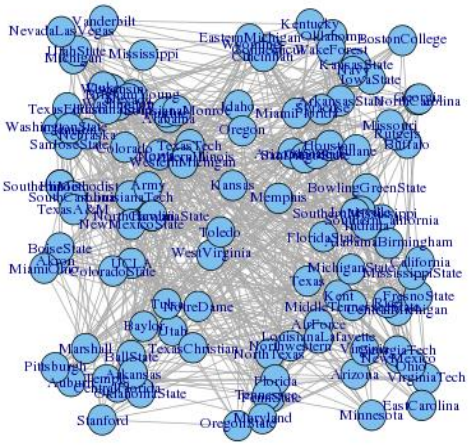
2. MATERIAIS E MÉTODOS

Escolhi dois conjuntos de dados para analisar:

- Um grafo que representa as aparições de personagens simultaneamente no livro de Victor Hugo “Les Miserables”. Os vértices representam os personagens e as arestas entre dois vértices representam uma aparição simultânea entre eles durante o livro. A seguinte imagem demonstra visualmente essa estrutura:



- O segundo conjunto de dados que escolhi representa os jogos de um campeonato de futebol americano do ano 2000. Os vértices representam os times que participaram do campeonato e as arestas os jogos entre eles.



Utilizei as seguintes structs para manipular os dados dos grafos:

```c
typedef struct Vertice
{
    int id;
    int grau;
    double closeness;
    double betweenness;
    double clustering;
    Lista *listaAdj;

    //BFS
    int cor;
    int dist;

    //Lista, fila
    struct Vertice *proximo;
}Vertice;
```

```c
struct Lista
{
    Vertice *inicio, *fim;
};

typedef struct Fila
{
    Vertice *inicio, *fim;
}Fila;

typedef struct Grafo
{
    Vertice *vert;
    int nVerts;
}Grafo;
```

2.1 MÉTODOS

I. Grau

Para calcular o grau dos vértices, sabendo que estou trabalhando com grafos não dirigidos, sempre que uma nova aresta é criada entre 2 nós, os graus deles são incrementados.

```c
void InserirAresta(Grafo *g, int v1, int v2)
{
    g->vert[v1].grau++;
    g->vert[v2].grau++;
    InserirLista(g->vert[v1].listaAdj, g->vert[v2]);
    InserirLista(g->vert[v2].listaAdj, g->vert[v1]);
}
```

II. Closeness

Para calcular essa medida me baseei na seguinte fórmula, que calcula a média das distâncias mínimas entre um vértice e todos os outros vértices do grafo.

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

Como optei por trabalhar com grafos não ponderados, utilizei um algoritmo de busca em largura (BFS) para calcular os caminhos mínimos entre os vértices (para grafos ponderados, Dijkstra seria um substituto). Poderia ser utilizado o algoritmo de Floyd-Warshall, mas optei por calcular uma distância mínima por vez, como a quantidade de vértices não era tão grande.

```c
int BFS(Grafo *g, int vInicial)
{
    int i;
    int dists[g->nVerts];
    int cores[g->nVerts];
    int total = 0;

    for (i = 0; i < g->nVerts; i++)
    {
        cores[i] = BRANCO;
        dists[i] = INFINITO;
    }

    Fila f;
    f.inicio = NULL;
    f.fim = NULL;

    cores[vInicial] = CINZA;
    dists[vInicial] = 0;

    Vertice v;
    v.id = vInicial;
    v.cor = CINZA;
    v.dist = 0;

    Enfileirar(&f, v);

    while(!FilaVazia(&f))
    {
        Vertice v2 = Desenfileirar(&f);
        cores[v2.id] = PRETO;
        total += v2.dist;

        Vertice *aux = g->vert[v2.id].listaAdj->inicio;

        while (aux != NULL)
        {
            if (cores[aux->id] == BRANCO)
            {
                cores[aux->id] = CINZA;
                dists[aux->id] = dists[v2.id]+1;

                v.cor = CINZA;
                v.dist = dists[v2.id]+1;
                Enfileirar(&f, v);
            }
            aux = aux->proximo;
        }
    }

    return total;
}
```

III. Betweenness

Foi a medida mais complicada de ser calculada. Me baseei na seguinte fórmula mas acabei utilizando somente o conceito da medida (*"Betweenness: quantifica o número de vezes que um vértice age como uma ponte ao longo do caminho mais curto entre dois outros vértices. "*):

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Para começar, gerei uma matriz de distâncias mínimas com o algoritmo Floyd-Warshall.

```
void FloydWarshall(Grafo *g, int *M[])
{
    int i, j, k;
    for (i = 0; i < g->nVerts; i++)
    {
        Vertice *aux = g->vert[i].listaAdj->inicio;
        while (aux != NULL)
        {
            M[i][aux->id] = 1;

            aux = aux->proximo;
        }
    }

    for (i = 0; i < g->nVerts; i++)
        for (j = 0; j < g->nVerts; j++)
            for (k = 0; k < g->nVerts; k++)
                if (M[j][i] + M[i][k] < M[j][k])
                    M[j][k] = M[j][i] + M[i][k];
}
```

Em seguida, utilizei uma estrutura de 3 dimensões [i][j][k] ("array de 3 dimensões") para guardar informações sobre quantas vezes o vértice 'i' está em um dos caminhos mínimos entre 'j' e 'k'.

```
int ***passa = (int***) malloc(g->nVerts * sizeof(int**));
printf("%d\n", g->nVerts);
for (i = 0; i < g->nVerts; i++)
{
    passa[i] = (int**) malloc(g->nVerts * sizeof(int*));
    for (j = 0; j < g->nVerts; j++)
    {
        passa[i][j] = (int*) malloc(g->nVerts * sizeof(int));
        for (k = 0; k < g->nVerts; k++)
            passa[i][j][k] = 0;
    }
}
```

Ao analisar todos os caminhos entre dois vértices j e k, verifiquei se a distância total daquele caminho era igual à que constava na matriz de Floyd-Warshall, assim foi possível filtrar apenas os caminhos mínimos entre j e k. Para cada vértice i diferente de j e k e entre eles nesses caminhos já filtrados, atribuo à estrutura tridimensional na posição [i][j][k] o valor 1. Além disso, qualquer vértice L que estiver no caminho j-i ou i-k (ou seja, [L][j][i] > 0 ou [L][i][k] > 0) recebe um incremento na estrutura tridimensional na posição [L][j][k]. (nMin [j][k] armazena o número total de menores caminhos entre j e k)

```
for (i = 0; i < g->nVerts; i++)
{
    for (j = 0; j < g->nVerts; j++)
    {
        for (k = 0; k < g->nVerts; k++)
        {
            if (M[j][i] + M[i][k] == M[j][k] && j != k && M[j][k] != 1)
            {
                if (i != k && i != j)
                {
                    passa[i][j][k] = 1;

                    for (l = 0; l < g->nVerts; l++)
                        if (l != k && l != j && l != i)
                            if (passa[l][j][i] || passa[l][i][k])
                                passa[l][j][k]++;
                    nMin[j][k]++;
                }
                if (M[j][k] == 1) nMin[j][k] = 1;
            }
        }
    }
}
```

Após isso, a medida de centralidade Betwenness é calculada usando a fórmula acima.

```
for (i = 0; i < g->nVerts; i++)
{
    for (j = 0; j < g->nVerts; j++)
    {
        for (k = 0; k < g->nVerts; k++)
        {
            if (nMin[j][k] != 0)
                min[i] += (float)passa[i][j][k] / (float)nMin[j][k];
        }
    }
    g->vert[i].betweenness = min[i];
}
```

IV. Coeficiente de Clustering

$$Cc_i = \frac{2t_i}{g_i(g_i - 1)}$$

Essa medida calcula o número de triângulos formados a partir de um vértice i, e divide pelo número total de triângulos que existiriam caso todos os vértices adjacentes à i fossem conectados. Para calcular o número de triângulos partindo de i, verifiquei se os vértices j adjacentes a i, possuíam adjacentes k que eram conectados a i. Como cada triângulo era contabilizado duas vezes (partindo de i para j e de i para k), dividi o número de triângulos por 2 antes de aplicar na fórmula.

```
for (i = 0; i < g->nVerts; i++)
{
    nTriang[i] = 0;

    Vertice *aux = g->vert[i].listaAdj->inicio;

    while (aux != NULL)
    {
        Vertice *aux2 = g->vert[aux->id].listaAdj->inicio;

        while (aux2 != NULL)
        {
            Vertice *aux3 = g->vert[aux2->id].listaAdj->inicio;

            while (aux3 != NULL)
            {
                if (aux3->id == i)
                    nTriang[i]++;

                aux3 = aux3->proximo;
            }

            aux2 = aux2->proximo;
        }

        aux = aux->proximo;
    }

    nTriang[i] /= 2;

    if (g->vert[i].grau > 1)
        g->vert[i].clustering = (float)(2*nTriang[i])/(float)(g->vert[i].grau*(g->vert[i].grau-1));

    else
        g->vert[i].clustering = 0;
}
```

V. Ranking

Criei e armazenei em arquivos rankings de cada medida utilizando o método de ordenação Quicksort.

3. RESULTADOS

Ao analisar os dados gerados a partir do grafo sobre “Les Miserables”, pude verificar que a personagem que aparece simultaneamente com mais personagens diferentes é Valjean, que é a personagem principal do livro, aparecendo em conjunto com exatas 36 personagens diferentes.

A personagem com maior proximidade média com relação a todas as outras personagens é Jondrette. Jondrette é a fase adolescente da personagem Eponine, que também possui uma proximidade relativamente alta, o que tornaria sua proximidade ainda mais alta caso as duas fossem consideradas a mesma personagem. Essa medida representa quão próxima ela está de todas as outras personagens do romance, ou seja, ela se relaciona com pessoas de várias comunidades diferentes, o que a torna próxima de todas as comunidades.

A personagem com maior betwenness é também Valjean. Como ele conhece quase todas as personagens do livro, ele serve de “ponte” entre muitas delas.

Muitas personagens possuem coeficiente de clustering igual a 1, ou seja, todas as personagens que estas conhecem, também se relacionam. Geralmente esse valor é agregado a personagens de papel secundário, com um grau de relacionamentos pequeno.

Sobre o grafo do campeonato de turebol, o time com maior grau foi KansasState, que foi o campeão e disputou com o maior número de times diferentes.

O time com maior closeness foi OregonState, o que nos leva a crer que ele disputou com os times que chegaram nas finais.

O que possui maior betweenness é BrighamYoung, o que indica que foi o time que disputou com o maior número de times de regiões diferentes.

O maior coeficiente de clustering ficou com o time BrighamYoung, que podemos inferir ser um time que não se destacou nos jogos, como muitos dos times com os quais ele disputou, também disputaram entre si.

4. CONCLUSÃO

O desenvolvimento desse trabalho permitiu que eu aprendesse a fundo os conceitos de Grafos, utilizando medidas de análise que dizem muito sobre as características de sua estrutura e como elas se relacionam com a análise real dos dados. A maior dificuldade que encontrei foi na implementação do método que calcula a medida de centralidade Betweenness, pois tive que desenvolver uma lógica do zero, me baseando apenas nos conceitos da medida.

5. **REFERÊNCIAS**

BERTON, L., slides de aula. Primeiro semestre de 2017.

D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley, Reading, MA (1993).

M. Girvan and M. E. J. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821-7826 (2002).