DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Relatório Intermédio do Projeto de LAPR1

Subtítulo adequado ao âmbito/tema do trabalho.

1181628_André Novo
1180817 _ Beatriz Ribeiro
1181592 _ Guilherme Tavares
1170500 _ Hugo Frias
1180730 _ Vera Pinto

Docente(s)/Orientador(es)
Ana Barata (abt)
Luís Ferreira (llf)
Vítor Cardoso(vcc)

Cliente
Carlos Ferreira (cgf)

Unidade Curricular
Laboratório/Projeto I

Dezembro, 2018

isep Instituto Superior de Engenharia do Porto

ÍNDICE

1. Introdução

O presente relatório inclui todo o processo do desenvolvimento do projeto, no âmbito da Unidade Curricular de Laboratório/Projeto I, desenvolvido pela equipa *OKAPI*, e tem como objetivo criar uma aplicação na linguagem *JAVA* de modo a estudar a análise de redes sociais.

Com esta análise são disponibilizados vários resultados em relação a um determinado conjunto de dados previamente selecionado pelo utilizador, tais como: grau de nós, centralidade do vetor próprio, densidade, grau médio e potências da matriz de adjacência. Para tal, serão necessários estudos e aplicações relativos a Matemática e Álgebra Linear.

De um modo geral, neste documento vamos relatar a nossa metodologia de trabalho, ou seja, a maneira como foram distribuídas as tarefas e uma avaliação geral do processo. Vamos abordar também o tópico principal, a análise de redes sociais, os diferentes métodos utilizados para determinar cada uma das funcionalidades a nível teórico e prático (em linguagem *JAVA*), e uma conclusão.

Este projeto não só nos permite desenvolver outras capacidades de gestão e trabalho de equipas, como também obter experiência a nível profissional e académico, uma vez que vão ser aplicados conhecimentos relativos à área da Engenharia Informática.

2. Metodologia de trabalho

2.1 EduScrum

A metodologia *eduScrum* é uma estrutura de trabalho na qual nós, estudantes tentamos alcançar de forma produtiva e criativa as metas de aprendizagem. Este método de trabalho desafia-nos, também, na auto-organização e na qualidade do trabalho dentro de um determinado período de tempo com metas de aprendizagem claras (Beaumont, Delhij, & Gerrits, 2015).

Deste modo, a estrutura de *eduScrum* consiste numa equipa e as suas respetivas funções, como por exemplo, em cada equipa existe um *Scrum Master* que é o líder. Assim, o *Scrum Master* organiza a distribuição de tarefas entre os diferentes elementos, comunica com as diferentes entidades, tenta facilitar as diferentes tarefas e/ou remover impedimentos para o progresso da equipa. Para além disso, é o *Scrum Master* que define as tarefas a realizar em cada sprint (conjunto de tarefas para realizar num intervalo de tempo) e assegura-se de que estas são cumpridas (Beaumont et al., 2015).

Foi através desta metodologia que conseguimos definir métodos de trabalho, isto é, conseguimos organizar, distribuir as tarefas. Assim, podemos afirmar que ter uma autoridade em trabalho de equipa ajuda a melhorar o desempenho, a produtividade e a cooperação dos membros.

2.2 Planeamento e distribuição de tarefas (Iteração 1)

Inicialmente foi fornecido ao grupo o enunciado da primeira parte do Projeto Final da unidade curricular Laboratório/Projeto I. Para que fosse mais fácil no desenvolvimento do projeto, não só se dividiram os objetivos propostos em pequenos tópicos (tarefas), como também foi feita uma previsão da dificuldade e tempo necessário para cada um desses objetivos, de modo a controlar de uma forma mais eficaz o planeamento.

As tarefas propostas para este trabalho até ao momento foram distribuídas de forma equitativa para que nenhum membro da equipa ficasse prejudicado ou sobrecarregado. Numa primeira fase, o *Scrum Master* da primeira semana reuniu o grupo e questionou cada um sobre qual a sua tarefa preferencial, ou seja, a distribuição das tarefas foi feita de modo a facilitar o desenvolvimento do projeto a cada elemento. No entanto, caso duas ou mais pessoas pretendessem trabalhar na mesma tarefa, a distribuição terá sido feita aleatoriamente através de uma plataforma online, *Random*.

Com o objetivo de facilitar a gestão das tarefas foi usada a ferramenta *Trello* (Trello, 2018), proposta pelos docentes da cadeira. Através dela foi possível verificar o progresso de cada um, bem como comunicar entre equipa para esclarecermos as dúvidas em relação ao projeto, antes de recorrermos aos docentes/cliente.

Com esta plataforma, o nosso espírito de trabalho de equipa progrediu de forma a que todos nós atualizássemos toda a equipa sobre o que já tinha sido realizado da parte de cada um. Esta simples atitude permitiu que o trabalho de ninguém se sobrepusesse, isto é, ao fazer *updates* periódicos nos nossos *cards*, garantimos que somente um único elemento da equipa trabalho numa tarefa.

O *Trello* também possui um sistema de , *labels* por cores que nos foi demonstrado pela docente, para que nós o pudéssemos aplicar ao longo deste projeto final. Ora, estas *labels* possibilitaram a cada um sinalizar o estado de determinado projeto ou atividade de uma forma mais intuitiva (Trello, 2018).

Em suma, o planeamento e a distribuição de tarefas ao longo desta primeira parte do projeto final decorreu de uma forma mais eficaz do que nos projetos anteriores da Unidade Curricular. Fomos capazes de cumprir com os prazos do *Trello*, bem como de nos incentivar a atualizar a equipa em horários periódicos de modo a não prejudicar ninguém.

2.3 Autoavaliação

Desde do início do projeto que a nossa equipa teve bons resultados, isto é, em termos de dividir tarefas, realizá-las e a ajudarem-se mutuamente. Deste modo, podemos afirmar que não tivemos problemas em discutir as formas de resolver problemas, apesar de termos maneiras diferentes de os solucionar chegamos sempre a um consenso.

Para além disso, as maiores dificuldades que tivemos a desenvolver a aplicação foram as verificações dos ficheiros, nós e ramos e os métodos para a centralidade de vetores. Contudo conseguimos ultrapassar todas as adversidades que tivemos ao longo destas duas semanas e atingimos os objetivos apresentados.

3. Análise de Redes Sociais

Ao longo dos anos, as redes sociais têm obtido cada vez mais importância nas últimas gerações, uma vez que contribuem para a comunicação entre toda a sociedade. Porém, para a criação destas redes mencionadas é preciso realizar um processo muito importante: a análise das Redes Sociais.

3.1. Introdução à Análise das Redes Sociais

A análise de redes sociais é uma metodologia que tem vindo a ser desenvolvida devido a várias ciências desde a sociologia, antropologia, física, matemática, psicologia social entre outras, tornando a análise de redes sociais um campo científico bastante rico Através da Álgebra Linear e de cálculos matemáticos, a análise de redes sociais consiste em estudar as relações que existem entre várias entidades de uma determinada rede utilizando diversas métricas (Gama & Oliveira, 2012).

Uma rede social é uma estrutura social composta por diversas entidades como pessoas, grupos ou organizações que possuem algum tipo de padrão em termos de relacionamentos ou interações entre os mesmos. Podem ser representadas por um grafo matemática, em que todas as entidades são denominadas como nós ou vértices e as interações entre eles são os ramos ou arestas (Gama & Oliveira, 2012).

Em alternativa, é possível representar uma rede social através de uma matriz de Adjacências. Esta matriz, é uma matriz quadrada, ou seja, o número de colunas é igual ao número de linhas, em que cada conjunto de linha com coluna representam as diferentes entidades. Se existir conexão entre dois nós, o coeficiente da matriz toma o valor 1. Caso contrário, esta toma valor 0. Para representar as ligações de uma rede com elevado número de entidades, recorre-se a uma lista de ramos, ou seja, uma lista que contém um nó, o seu vizinho e por último o indicador 1 caso haja ligação entre eles (Mohammad, Abbasi, & Liu, n.d.).

Algumas das tarefas mais comuns do processo de análise de redes são, por exemplo, a descoberta de comunidades e a identificação das entidades mais influentes utilizando medidas estatísticas (Gama & Oliveira, 2012). Por exemplo, há também empresas que utilizam a análise de redes de modo a maximizar o poder de marketing dos seus produtos, ao fazê-los chegar aos clientes com maior influência numa determinada rede social. Há também casos em que a análise das redes pode ser aplicada a redes relacionadas com meios de

comunicação de modo a analisar a frequência e direção de e-mails informais ou formais para revelar padrões de comunicação entre funcionários.

3.2. Métodos de Análise de Redes

Com a ajuda da estatística é possível obter informações bastante importantes de uma determinada rede social sem necessitar da representação gráfica da mesma. Estas análises podem ser divididas de acordo com o seu nível de análise, indo desde a análise de unidades pequenas como os atores, até á análise de redes sociais inteiras (Gama & Oliveira, 2012).

3.2.1. Métodos ao nível dos atores utilizados

Um dos métodos utilizados foi o do $grau\ de\ um\ nó$[1] (*Node degree*). O grau de um nó v, que é geralmente identificado como k_v, é uma medida de adjacência imediata e do envolvimento do nó da rede e corresponde ao número de arestas ligadas a qualquer nó. Apesar da sua simplicidade, o grau do nó é uma medida eficaz para obter a importância de uma certa entidade (Mohammad et al., n.d.).

$$k_v = \sum_{j=1}^{n} a_{vj}\,, \qquad 0 < k_v < n \tag{1}$$

No caso das redes sociais, o grau representa o número de amigos que um determinado utilizador tem. Por exemplo, no Instagram o grau do nó corresponde ao número de seguidores de um utilizador. a_{vj} representa um coeficiente da matriz de adjacências A e n é o número de nós da rede (Mohammad et al., n.d.).

Outro método ao nível dos atores utilizado foi o da $centralidade\ do\ vetor\ próprio$[2] (*Eigenvector Centrality*). Com este método é possível generalizar a medida grau do nó, medindo quão bem está conectado uma entidade aos seus vizinhos, não pela quantidade de ligações, mas pela sua qualidade. Dito isto, a ideia básica por trás deste conceito é que o poder de um determinado nó é defenido pelo poder e estado dos nós conectados ao mesmo (Gama & Oliveira, 2012).

$$x_i = \frac{1}{\gamma} \sum_{j=1}^{n} a_{ij}\, x_j \tag{2}$$

Nesta fórmula x_i/x_j representa a centralidade do nó i, a_{ij} representa um coefeciente da matriz de ajacências A e γ é o maior vlaor próprio dessa mesma matriz.

Também é possível determinar a centralidade do vetor próprio através do Teorema de Perron-Frobenius, que mostra que ao determinar o maior valor próprio da matriz Adjacências, o seu correspondente vetor próprio indica as centralidades de todos os nós da matriz. O nó, ou seja, entidade com o maior valor do vetor próprio é a entidade mais central, isto é, a entidade com mais importância a nível qualitativo (Mohammad et al., n.d.).

3.2.2. Medidas ao nível da rede

A primeira das medidas a nível da rede utilizadas foi o $grau\ médio$[3] (*Average Degree*). Esta medida representa a média dos graus de todos os nós de uma rede, permitindo medir a conectividade global da mesma rede (Gama & Oliveira, 2012; Mohammad et al., n.d.).

$$\bar{k} = \frac{1}{n}\sum_{i=1}^{n} k_i \tag{3}$$

Nesta fórmula, k_i é o grau do nó i e n é o número de nós da rede.

Outra medida usada foi a $densidade$[4] (Density). Esta medida serve essencialmente para explicar o nível geral de uma rede. O valor da densidade representa o número de ramos da rede em relação ao número máximo possível de ramos. A densidade é uma quantidade que varia entre 0 até 1. A partir da análise deste valor podemos definir o tipo de rede com o qual estamos a lidar, sendo ela uma rede densa ou escassa quando a densidade é baixa ou elevada, respetivamente (Gama & Oliveira, 2012; Mohammad et al., n.d.).

$$\rho = \frac{m}{m_{max}}, \qquad 0 < \rho < 1 \tag{4}$$

Nesta fórmula, ρ representa a densidade, m é o número de ramos da rede e m_{max} é o número de ramos caso consideremos que existe um ramo entre cada um dos pares de nós da rede em questão.

Por fim, segue-se o método das $Potências\ da\ Matriz\ de\ Adjacências$[5] (Powers of the adjacency matrix).

$$A^k = \prod_{i=1}^{k} A \tag{5}$$

Nesta fórmula, A^k é a k-enésima potência da matriz de adjacências A.

A matriz de adjacência diz-nos quantos caminhos de comprimento um existem entre cada par de nós. A matriz de adjacência de ordem *k* permite obter o número de caminhos de comprimento *k* entre qualquer para de nós.

4. Desenvolvimento e Implementação da Aplicação

Neste capítulo vão ser apresentados todos os métodos necessários para o desenvolvimento e implementação da aplicação. Alguns termos utilizados anteriormente no capítulo 3 vão ser aqui também referidos de modo a que se obtenha uma linguagem mais adequada e organizada para a compreensão do próprio relatório. Para analisar a rede, implementámos vários métodos requisitados pelo projeto: grau do nó, centralidade do vetor próprio, grau médio, densidade e potência da matriz de Adjacências.

4.1. Linha de comandos, leitura de ficheiros

A aplicação inicialmente apresenta duas alternativas: obter o output na consola através de um menu com várias opções ou enviar todos os resultados de todas as opções em conjunto para um novo ficheiro de teste. Para tal, através da linha de comandos se o primeiro argumento fosse *-n* o programa seguia para a primeira alternativa, caso os dois primeiros argumentos fossem *-t* e *-k*, o programa seguia para a outra alternativa.

Também como argumentos entram os ficheiros cujo nome deve estar apresentado da seguinte maneira: *rs_media_nos.csv* e *rs_media_ramos.csv*, que contêm toda a informação das entidades e toda a informação relativamente ao estado de conexão entre eles (lista de ramos), respetivamente.

Em relação ao conteúdo dos ficheiros, este foi analisado de modo a que os resultados fossem válidos e corretos. Por exemplo, ambos os ficheiros não podem ter linhas repetidas, portanto, é proposto ao utilizador a opção de continuar o programa ou sair deste. Caso o utilizador pretenda continuar, a linha repetida é eliminada. Também foi verificado se no ficheiro dos ramos alguma entidade tinha ligação com ela mesma.

4.2. Grau do nó

No método apresentado é determinado o grau de cada nó e são recebidos como parâmetros a matriz de Adjacências (*branchesMatrix*), um *array* vazio para guardar os graus de cada nó (*nodeDegreeArray*) e o número de elementos (*nElem*).

```
public static int nodeDegree(double[][] branchesMatrix, double[]
nodeDegreeArray, int nElem) {
       int indexArray = 0;

       for (int node = 0; node < nElem; node++) {
           int nodeDegree = 0;

           for (int neighbour = 0; neighbour<branchesMatrix[0].length;
neighbour++) {
               if (branchesMatrix[node][neighbour] != 0) {
                   nodeDegree++;
```

```
                }
            }
            nodeDegreeArray[indexArray] = nodeDegree;
            indexArray++;

        }

        return nElem;
    }
```

O grau do nó pode ser calculado através da contagem de dígitos iguais a 1 para cada nó, sendo este valor necessariamente maior que "0" e inteiro, uma vez que pelo menos um nó deve ter ligação com um nó vizinho. À medida que se determina o grau para cada nó, este é adicionado a uma matriz de comprimento igual ao número de entidades. Por fim, este método retorna o número de nós.

4.3. Centralidade do vetor próprio

Em relação a esta modalidade, para ser calculada a centralidade do vetor próprio, são recebidos como parâmetros a matriz das Adjacências (*branchesMatrix*), o número de elementos (*nElem*) e um array vazio para guardar todos as centralidades de cada nó.

```
public static void
determinarCentralidadeVetorProprio(double[][]branchesMatrix, int nElem,
double[] eigenValueCentrality) {

        Matrix matrixA = new Basic2DMatrix(branchesMatrix);

        int colunaMaiorValorProprio;

        EigenDecompositor decompositor = new EigenDecompositor(matrixA);
        Matrix[] eigen;
        eigen = decompositor.decompose();

        double[][] valoresProprios = valoresProprios =
eigen[INDICE_VALORES_PROPRIOS].toDenseMatrix().toArray();

         double[][] vetoresProprios =
eigen[INDICE_VETORES_PROPRIOS].toDenseMatrix().toArray();

         double[] arrayVetorProprio = new double[nElem]; // Criar vetor só com
os valores do vetor correspondente ao maior valor próprio

         colunaMaiorValorProprio=obterColunaMaiorValorProprio(valoresProprios);

        for (int i = 0; i < nElem; i++) {
          arrayVetorProprio[i] = vetoresProprios[i][colunaMaiorValorProprio];
        }

        for (int i = 0; i < nElem; i++) {
            double soma = 0;
            for (int j = 0; j < nElem; j++) {
                soma = soma + (arrayVetorProprio[j] * branchesMatrix[i][j]);
            }

         eigenValueCentrality[i] = (double) Math.round(1 /
valoresProprios[colunaMaiorValorProprio][colunaMaiorValorProprio] * soma *
1000) / 1000;
        }
```

```
    }
```

Para tal, com o auxílio da biblioteca *la4j- linear algebra for Java* (http://la4j.org/apidocs/), foi possível determinar os valores e vetores próprios da matriz de adjacências, utilizando os métodos:

- org.la4j.Matrix;
- org.la4j.decompositionEigenDecomposition;
- org.la4j.matrix.dense.Basic2DMatrix.

Os valores próprios e vetores próprios são guardados em duas matrizes diferentes (*valoresProprios* e *vetoresProprios*, respetivamente). Pela matriz *valoresProprios* determina-se o maior valor próprio, cujo índice da sua coluna corresponderá ao índice da coluna da matriz *vetoresProprios* que contém o respetivo vetor próprio. Cada valor desse vetor será aplicado na fórmula referida já anteriormente.

Foi possível concluir também que, pelo Teorema de Perron-Frobenius, a centralidade de cada nó corresponde ao vetor próprio do maior valor próprio.

4.4. Grau médio

Este *método* recebe como parâmetros um *array* de inteiros onde estão todos os nós de uma rede (*nodeDegreeArray*) e retorna o valor médio dos valores desse *array* numa variável (*averageDegree*).

```
public static double calculateAverageDegree(double[] nodeDegreeArray) {
        int sum = 0;

        double averageDegree;

        for (int i = 0; i < nodeDegreeArray.length; i++) {
            sum += nodeDegreeArray[i];
        }

        averageDegree = (double) sum / nodeDegreeArray.length;

        return averageDegree;
    }
```

4.5. Densidade

O método *calculateDensity* recebe como parâmetros as variáveis *density* (densidade) iniciada a zero, *nElem* (número de nós), *actualConnections* (número de ramos existentes) e *potencialConnections* (máximo número de ramos) e retorna o valor da densidade.

```
public static double calculateDensity(double density, int nElem, double
actualConnections) {
        double potencialConnections = 0;

        potencialConnections = calcPotencialConnections(nElem,
potencialConnections);

        density = calcDensity(actualConnections, potencialConnections);

        return density;
    }
```

O método *calcDensity* calcula o quociente do número de ramos que existem (*actual connections*) e o número de ramos que podiam existir entre aqueles nós (*potencial connections*). Deste modo, o método *calcPotencialConnections* calcula o número de ramos que pode existir através do número de nós, isto é, ao multiplicar o número de nós por ele próprio e subtrair-lhe um, obtemos o número de conexões que existem, mas ao dividirmos por dois retira os ramos duplicados. Também para calcular o número de nós temos o método *calcActualConnections* que percorre a matriz que contém as ligações entre cada nó e sempre que encontrar uma ligação incrementa 1 às *actualConnections* e, no final, divide por dois para retirar ligações duplicadas.

4.6. Potência da matriz Adjacências

O método criado calcula todas as potências de uma determinada matriz de adjacência até um *k* (*exponent*) fornecido pelo utilizador. Este método recebe como parâmetros a própria matriz de adjacência (*adjacencyMatrix*) e o expoente em questão (*exponent*) e, além de mostrar todas as potências desta matriz de adjacência até k, retorna a matriz-resultado de maior grau (*powerAdjacencyMatrixResult*).

```
public static double[][] powerAdjacencyMatrix(double[][] branchesMatrix, int
exponent, int nElem) {
        double[][] powerAdjacencyMatrixResult = branchesMatrix.clone();

        showHeader(1);
        listMatrixDouble(powerAdjacencyMatrixResult, nElem);

        out.format("%n");

        for (int i = 0; i < exponent - 1; i++) {
            out.format("%n");
            showHeader(i + 2);
            powerAdjacencyMatrixResult =
multiplyMatrices(powerAdjacencyMatrixResult, branchesMatrix, nElem);
            listMatrixDouble(powerAdjacencyMatrixResult, nElem);
            out.format("%n");
        }

        return powerAdjacencyMatrixResult;
    }
```

O valor de k é validado nas duas alternativas do programa, ou seja, no menu quando pedido ao utilizador e na leitura dos argumentos quando se pretende enviar tudo para um ficheiro.

5. Conclusão

Concluindo, a análise de redes sociais, desenvolvida com a ajuda de várias ciências, consiste na análise das entidades e ligações existentes dentro de uma rede e é importante para, por exemplo, marketing de empresas ou obter dados que podem ser uteis, tais como: frequência de certas pesquisas ou popularidade.

Em suma, a análise das redes sociais é um processo bastante importante, e com o decorrer dos anos, com a influência de algumas redes como o *Facebook* ou o *Instagram*, a tendência para o seu estudo e análise está cada vez maior.

O desenvolvimento da aplicação não só permitiu que pudéssemos aplicar todo o conhecimento adquirido até ao momento, relativo à área da Engenharia Informática como também evoluir a nível da programação. Para além disso, foi possível compreender a importância da álgebra linear na análise de redes sociais e ganhar experiência profissional.

Por fim, é possível reconhecer que, apesar de a equipa ter apresentado algumas dificuldades no início do procedimento do trabalho, foram alcançados bons resultados. Conseguimos cumprir com os requisitos do cliente e todos puderam beneficiar deste projeto, uma vez que abordámos e aprendemos novos temas. O trabalho desenvolvido foi organizado e produtivo, no entanto podemos melhorar em relação à implementação do programa.

Referências

Beaumont, S., Delhij, A., & Gerrits, T. (2015). *The eduScrum Guide Written by Arno Delhij, Rini van Solingen and Willy Wijnands*. Retrieved from http://eduscrum.nl/en/file/CKFiles/The_eduScrum_Guide_EN_1.2.pdf

Gama, J., & Oliveira, M. (2012). An Overview of Social Network Analysis. Retrieved from https://moodle.isep.ipp.pt/pluginfile.php/248729/mod_resource/content/1/artigo_JoaoGama.pdf

Mads Haahr. (2018). RANDOM.ORG - True Random Number Service. Retrieved December 22, 2018, from https://www.random.org/

Mohammad, R. Z., Abbasi, A., & Liu, H. (n.d.). *Social Media Mining: An Introduction*. Retrieved from http://dmml.asu.edu/smm,

Trello. (2018). About | What is Trello? Retrieved November 23, 2018, from https://trello.com/about

Weisstein, E. W. (n.d.). Eigen Decomposition. Retrieved from http://mathworld.wolfram.com/EigenDecomposition.html

ANEXOS

ANEXO A _ Testes Unitários

```
public class Testes {

     //Teste para grau do nó
    public static boolean[] test_nodeDegree(double[] expectedMatrix,
double[][] branchesMatrix, double[] nodeDegreeResult) {
        int nElem = nodeDegree(branchesMatrix, nodeDegreeResult,
branchesMatrix.length);
        boolean test[] = new boolean[nElem];
        for (int i = 0; i < nElem; i++) {
            if (expectedMatrix[i] == nodeDegreeResult[i]) {
                test[i] = true;
            }
        }
        return test;
    }
    //Teste para centralidade do vetor próprio
    public static boolean[] test_EigenValueCentrality(double[][]
branchesMatrix, int nElem, double[] eigenValueCentrality, double[]
centralityValuesExpected) {
        determinarCentralidadeVetorProprio(BRANCHESMATRIX, nElem,
EIGENVALUECENTRALITY);
        boolean test[] = new boolean[nElem];
        for (int i = 0; i < nElem; i++) {
            if (centralityValuesExpected[i] == eigenValueCentrality[i]) {
                test[i] = true;
            }
        }
        return test;
    }
    // Teste para grau médio
    public static boolean test_calculateAverageDegree(double[] degreesGroup,
double averageDegreeExpected) {
        double averageDegree = calculateAverageDegree(degreesGroup);
        if (averageDegree == averageDegreeExpected) {
            return true;
        }
        return false;
    }
    //Teste para densidade
    public static boolean test_Density(double expectedValue, int nElem, double
actualConnections, double density) {
        boolean test = false;
        density = calculateDensity(density, nElem, actualConnections);
        String densityValue = df.format(density);
        double x = Double.parseDouble(densityValue.replace(",", "."));
        if (x == expectedValue) {
            test = true;
        }
        return test;
    }
    //Teste para potência de matriz de Adjacências
    public static boolean[][] test_powerAdjacencyMatrix(double[][]
adjacencyMatrix, int exponent, double[][] powerAdjacencyMatrix, double[][]
powerAdjacencyMatrixExpected, int nElem) {
        boolean[][] test = new boolean[nElem][nElem];

        for (int i = 0; i < adjacencyMatrix.length; i++) {
            for (int j = 0; j < adjacencyMatrix[0].length; j++) {
                if (powerAdjacencyMatrix[i][j] ==
powerAdjacencyMatrixExpected[i][j]) {
                    test[i][j] = true;
                }
            }
         return test
         }
    }
```

ANEXO B_Biblioteca la4j- linear algebra for Java

Kankava, George (2016) la4j.

https://github.com/vkostyukov/la4j/blob/master/src/main/java/org/la4j/LinearAlgebra.java

ANEXO C_Enunciado

LAPR1. (2018). *Enunciado*. Retrieved from https://moodle.isep.ipp.pt/pluginfile.php/248730/mod_resource/content/5/EnunciadoLAPR1_SNA_V2.pdf