Clément GILI – Eddy DRUET

Rapport du projet de graphe : problématique et enjeux

**Université de Bourgogne**

**UFR Sciences et techniques**

**Licence 3 - Informatique**

Rapport Graphes

# Introduction

Tout d’abord quelle était le cahier des charges du projet, dans le sujet il nous était demander d’effectuer dans un premier temps la prise en charge de différents types de fichier qui contenait des graphes afin de pouvoir par suite faire des opérations sur les graphes contenus dans ces fichiers étant disponible sur les sites (SNAP, KONECT).

Par la suite nous devions implémenter un algorithme qui permettent de calculer la dégénérescence d’un graphe afin de vérifier en quelles mesures les graphes réels sont denses. Il nous était demander de réfléchir à la meilleure façon de réaliser cet algorithme c’est-à-dire si l’on devait marquer un sommet dé que celui-ci et été passer ou si a contrario il fallait effacer le sommet de la structure du graphe. Nous allons répondre à cette problématique dans la section de l’implémentation de l’algorithme.

Une option en plus étaient de choisir d’implémenter une fonctionnalité en plus a notre programme, nous avons choisis la troisième qui est d’afficher un joli dessin de notre dégénérescence calculé, ce qui permet de pouvoir avoir la densité d’un côté plus visuelle.

# structure du projet

Afin de pouvoir réaliser le projet, il nous était laisser le choix libre du langage à utiliser nous avons choisis le web, car il proposait plusieurs avantages qui allais nous permettre de réaliser plus facilement certaines fonctionnalités.

Dans un premier temps, afin de pouvoir mieux appréhender les graphes nous avons directement voulu les visualiser graphiquement c’est pour cela que nous avons cherché à nous procurer une librairie permettant d’afficher les graphes sur un site web. Cette librairie a été très utiliser pour l’options que nous avons choisis car celle-ci recommandais de pouvoir afficher les graphes ou du moins graphes des graphes spécifiques et plus petite par rapport a celui que l’on voulait analyser.

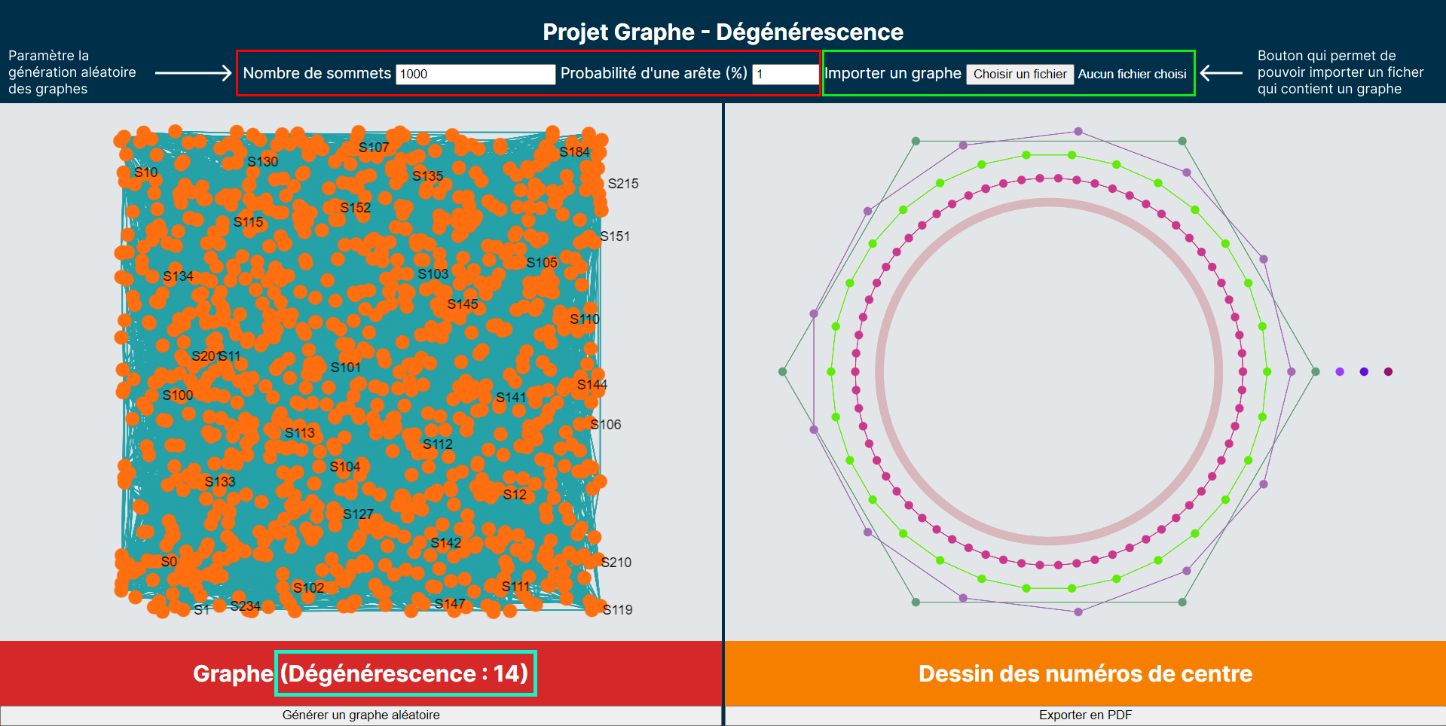
Voyons voir un visuel de l’application qui permettra de comprendre les explications suivantes.

Figure 1 : Code qui permet de générer un graphe aléatoirement

La deuxième chose à faire pour pouvoir importer les graphes fournis par les banques de graphes qui était répertorié sur les sites, était de pouvoir importer des fichiers ces fichiers et de les convertir et de les mettre dans une structure informatique qui permettrait de les exploiter, à la fois par des calcules sur les dit graphes mais également de pouvoir les afficher. Pour cela nous avons fait des fonctions qui permettent de convertir des fichiers (csv, txt) qui était sur le site, dans la structure que nous avons choisie.

Parlons de la structure, les graphes sont stockés dans un objet javascript qui contient lui-même des tableaux à l’index indiqué par un sommet ce qui fait que l’on peut donc avoir dans un fichier (1 2) et que le 1 sera donc le sommet et le 2 sera donc le voisin également je précise que pour que tout fonctionne correctement il faut donc également que le sommet 2 est comme voisin le sommet 1, c’est donc comme ça que la fonction d’importation des graphes va fonctionner.

function parseTXT(contenu) {

  let graphe = {};

  let lines = contenu.split("\n");

  // Retirer les lignes qui ne sont pas un couple (nombre, nombre).

  lines = lines.filter((line) => (line.match(/^\d+[\t ]\d+/) ? true : false));

  lines.forEach((line) => {

    let colonnes = line.split(/[\t ]/);

    let source = colonnes**[**0**]** - 1;

    let voisin = colonnes**[**1**]** - 1;

    if (source === voisin) return; // Dans certains fichiers, des sommets sont voisin avec eux-mêmes.

    if (!graphe**[**source**]**) graphe**[**source**]** = **[]**;

    if (!graphe**[**voisin**]**) graphe**[**voisin**]** = **[]**;

    if (graphe**[**source**]**.indexOf(voisin) === -1) {

      graphe**[**source**]**.push(voisin);

    }

    if (graphe**[**voisin**]**.indexOf(source) === -1) {

      graphe**[**voisin**]**.push(source);

    }

  });

  const event = new **CustomEvent**("graph-load", { detail: graphe });

  document.dispatchEvent(event);

}

Figure 2 : Code qui converti un fichier txt en graphe

Je vous ai parlé de l’importation des fichiers qui contiennent des graphes mais il y a une fonctionnalité qui permet également de générer des graphes de façon aléatoire en fonction du nombre de sommets en entrés et de la probabilité des arêtes entre les sommets.

function genererGraphe(nbSommets, probabilite) {

  const graphe = **[]**;

  for (let i = 0; i < nbSommets; i++) {

    graphe**[**i**]** = **[]**;

  }

  for (let i = 0; i < nbSommets; i++) {

    for (let j = 0; j < nbSommets; j++) {

      const num = **Math**.random() \* 101;

      if (num < probabilite && j != i && graphe**[**i**]**.indexOf(j) === -1) {

        graphe**[**i**]**.push(j);

        graphe**[**j**]**.push(i);

      }

    }

  }

  const event = new **CustomEvent**("graph-load", { detail: graphe });

  document.dispatchEvent(event);

}

Figure 3 : Code qui permet de générer un graphe aléatoirement

# Algorithme de dégénerescence

Nous avons implémenté l’algorithme de dégénérescence, et nous avons choisis l’option d’effacer le sommet dans le tableau lorsque l’on est passé dessus. Voyons voir le fonctionnement de l’algorithme :

Dans un premier temps, nous avons mis en place un tableau qui permet de contenir l’identique du graphe mais cette fois-ci nous avons à la place des voisins pour le sommet le numéro K de dégénérescence du sommet qui sera inscrit au fur et à mesure, ce tableau sera retourné à la fin de la fonction. On peut noter qu’il faut donc faire une petite boucle pour initialiser ce tableau.

Également nous avons définis des variables, le numéro K initialisé à 1, qui permet de savoir à quel niveau de dégénérescence nous en somme et le nombre de sommets restants qui est initialisé aux nombres de sommets du graphe.

Par suite nous allons rentrer dans le cœur de l’algorithme dans une While qui va fonctionner tant qu’il restera des sommets dans le graphe qui n’aurons pas été traiter. Nous mettons une variable Trouver à False afin de savoir si durant le tour de la boucle nous avons trouver un sommet à supprimer ou s’il faut incrémenter le nombre K.

La boucle for dans la While permet de parcourir le graphe et de regarder si des sommets possèdent un nombre de voisin inférieur ou égale à K, pour par la suite mettre dans le tableau de dégénérescence K du sommet, supprimer dans les différents voisins de ce sommets sont numéro et le supprimer lui-même du tableau, on décrémente le nombre de sommets restants et on met Trouver a True pour signaler que nous avons trouver un sommet et qu’il ne faut pas changer de K, à l’inverse si il n’y a pas de sommet il faut incrémenter le nombre K.

function degenererGraphe(graphe) {

  const nbSommets = getObjectLength(graphe);

  graphe = **JSON**.parse(**JSON**.stringify(graphe)); // Dupliquer le graphe.

  const degenerer = {};

  let k = 1;

  let nbSommetsRestant = nbSommets;

  // Initialisation du tableau de dégénérescence.

  for (const numSommet in graphe) {

    degenerer**[**numSommet**]** = **[**+numSommet**]**;

  }

  while (nbSommetsRestant > 0) {

    let trouver = false;

    for (const numSommet in graphe) {

      const voisins = graphe**[**numSommet**]**;

      if (voisins && voisins.length <= k) {

        degenerer**[**numSommet**]**.push(k); // Affecter le numéro de centre au sommet.

        // Supprimer le sommet de tous les voisins.

        voisins.forEach((voisin) => {

          graphe**[**voisin**]**.splice(graphe**[**voisin**]**.indexOf(+numSommet), 1);

        });

        graphe**[**numSommet**]** = null; // Marquer le sommet comme traité.

        nbSommetsRestant--;

        trouver = true;

        break;

      }

    }

    // Si aucun sommet n'a été supprimé dans cette itération, alors on passe au centre suivant.

    if (!trouver) {

      k++;

    }

    console.log("K :", k);

  }

  return degenerer;

}

Figure 4 : Code de l’algorithme de dégénérescence

# Options : dégenerescence joli dessin

Afin de pouvoir effectuer l’options que nous avons choisis, nous avons dû utiliser la librairie qui nous a permis de pouvoir afficher les graphes mais également le but était de prendre un screenshot de ce joli dessin que nous effectue la dégénérescence d’un graphe pour cela le javascript a une solution deux librairies une qui permet de pouvoir prendre une photo de la page et l’autre qui permet de générer des PDF, le combo de toutes les librairies nous a permis de faire la sortie d’une image.

Mais avant de pouvoir sortir une image, nous avons dans un premier temps du afficher le graphe de la dégénérescence, pour cela nous avons écrit une fonction qui permet de prendre le tableau de dégénérescence qui a été générer via la fonction qui applique l’algorithme et de pouvoir l’afficher comme un assemblage de petit graphe qui forme plusieurs cercles. On peut noter que pour que les sommets se mettent sous la forme d’un cercle nous avons utilisé la formule du périmètre d’un cercle.

function afficherNumerosCentre(centres) {

  // Transformer structure de données `centres`.

  const newCentres = **[]**;

  for (const arr of **Object**.values(centres)) {

    const sommet = arr**[**0**]**;

    const centre = arr**[**1**]** - 1;

    if (!newCentres**[**centre**]**) newCentres**[**centre**]** = **[]**;

    newCentres**[**centre**]**.push(sommet);

  }

  centres = newCentres;

  const graph = renderers.rendererCores.graph;

  graph.clear();

  let diametre = centres.length;

  // Pour chaque numéro de centre...

  centres.forEach((sommets) => {

    const anglePas = 360 / sommets.length;

    let angle = 0;

    const color = `rgb(${randInt(0, 255)}, ${randInt(0, 255)}, ${randInt(0, 255)})`;

    // ... plaçage des sommets autour d'un cercle.

    sommets.forEach((sommet) => {

      const x = **Math**.cos((angle / 180) \* **Math**.**PI**) \* (diametre / 2);

      const y = **Math**.sin((angle / 180) \* **Math**.**PI**) \* (diametre / 2);

      angle = (angle + anglePas) % 360;

      graph.addNode(sommet, {

        x: x,

        y: y,

        size: 5,

        color: color,

      });

    });

    // ... reliage des sommets pour former le périmètre du cercle.

    sommets.forEach((sommet, index) => {

      let voisin;

      if (index === sommets.length - 1) {

        voisin = sommets**[**0**]**;

      } else {

        voisin = sommets**[**index + 1**]**;

      }

      graph.addEdge(sommet, voisin, { color: color });

    });

    diametre--;

  });

  // Afficher la dégénérescence.

  document.getElementById("label-result").innerText = centres.length;

}

Figure 5 : Code qui affiche la dégénérescence à l’écran

Par la suite nous avons donc juste à faire un screenshot et mettre cette image dans une PDF.

function exportPDF() {

  const jpeg = new **Image**();

  jpeg.onload = () => {

    const pdf = new jspdf.jsPDF({ orientation: "landscape", unit: "px", format: **[**jpeg.height / 2, jpeg.width / 2**]** });

    pdf.addImage(jpeg, "JPEG", 0, 0, jpeg.height / 2, jpeg.width / 2);

    pdf.save("joli\_dessin.pdf");

  };

  const url = sigmaScreenshot(renderers.rendererCores, 768, 768);

  jpeg.src = url;

}

Figure 6 : Code pour prendre un screenshot et le mettre dans un PDF

# INformations supplémentaire et exemple

Pour ce qui est de l’utilisation de l’application, vous avez juste à lancer le index.html, il y a des champs de formulaires pour paramétrer les graphes générer de façon aléatoire, un bouton est proposé en bas pour regénérer un nouveau graphe. Quand vous importer un fichier de graphe, l’application va automatique l’afficher, à noter qu’il y a un dossier ou nous avons placer des graphes que vous pouvez tester. Afin de pouvoir tester la fonction permet d’exporter en PDF vous avez un bouton en dessous du joli dessin.

Une image contenant texte, ciel, capture d’écran

Description générée automatiquementL’exemple que je vais mettre ici est celui du graphe que vous avez mis dans votre sujet de projet nous avons mis les sommets dans un fichier txt.

Figure 7 : Image de l’application qui affiche le graphe du sujet

Pour conclure, notre application permet de générer des graphes aléatoires, d’importer des graphes depuis des fichiers (txt, csv), de la calculer la dégénérescence de graphes, d’afficher le graphe avant application de l’algorithme, et d’afficher une visuelle de la dégénérescence, d’exporter l’image dans un PDF.

Les améliorations possibles de l’application seraient de travailler sur l’implémentation de l’algorithme plus optimisé que vous avez suggérer dans les options, ou encore l’options qui permet de pouvoir comparer des graphes en fonction de leur dégénérescence d’autres paramètres (nombres de chromatique, coloration d’un graphe).