Rapport du projet de graphe : problématique et enjeux.

Université de Bourgogne
UFR Sciences et techniques
Licence 3 - Informatique

Rapport Graphes

Clément GILI – Eddy DRUET

INTRODUCTION

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

Par la suite, nous devions implémenter un algorithme qui permet de calculer la dégénérescence d'un graphe afin de vérifier en quelles mesures les graphes réels sont denses. Il nous était demandé de réfléchir à la meilleure façon de réaliser cet algorithme, c'est-à-dire, si l'on devait marquer un sommet dès que celui-ci et été passé 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 était de choisir d'implémenter une fonctionnalité en plus à notre programme, nous avons choisi la troisième qui est d'afficher un joli dessin de notre dégénérescence calculée, 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 laissé le choix libre du langage à utiliser nous avons choisis le JavaScript (web), car il proposait plusieurs avantages qui allaient 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 utilisée pour l'option que nous avons choisie, car elle nous a permis de facilement afficher le graphe en lui-même ainsi que le joli dessin des numéros de centre.

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

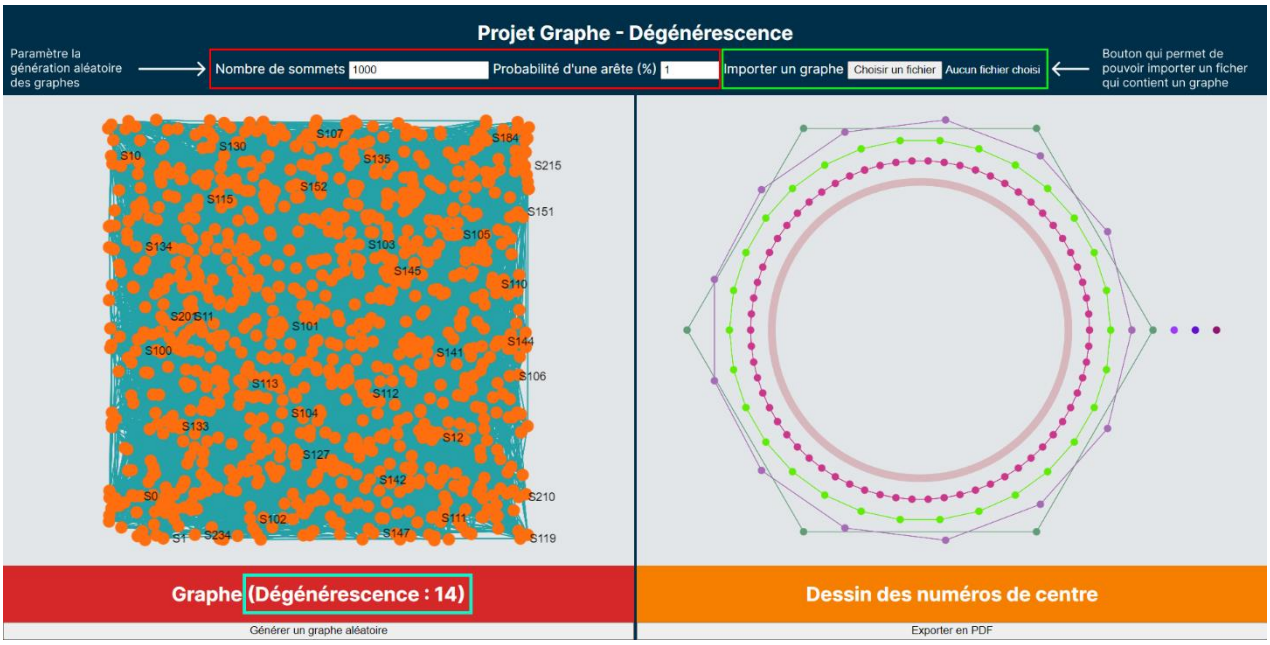

Figure 1 : Page web qui permet de générer un graphe aléatoirement

La deuxième chose à faire pour pouvoir importer les graphes fournis par les banques de graphes qui étaient répertoriés sur ces sites, était de pouvoir importer des graphes de ces fichiers et de les convertir et de les mettre dans une structure de données qui permettrait de les exploiter, à la fois par des calculs sur les dits graphes, mais également de pouvoir les afficher. Pour cela, nous avons fait des fonctions qui permettent de convertir des fichiers (CSV, texte) qui étaient sur le site, dans la structure de données que nous avons choisie.

Parlons de la structure de données, celle-ci est une liste d'incidence. Les graphes sont stockés dans un objet JavaScript qui *map* un numéro de sommet ➔ liste de ses voisins. Ce qui fait que l'on peut donc avoir dans un fichier importé, le couple (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 convertit un fichier texte en notre structure de données

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 entrée 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 DEGENERESCENCE

Nous avons implémenté l'algorithme de dégénérescence, et nous avons choisi l'option de marquer (en l'affectant par *null*) le sommet dans la liste d'incidence lorsque l'on est passé dessus. Voyons voir le fonctionnement de l'algorithme :

Dans un premier temps, nous avons mis en place un tableau « degenerer » qui permet de contenir une copie du graphe, mais cette fois-ci nous avons un tableau qui *map* le numéro de sommet ➔ son numéro K de dégénérescence, 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 de dégénérescence.

Également, nous avons définis des variables : le numéro « k » initialisé à 1, qui permet de savoir à quel niveau de dégénérescence nous en sommes ; et le nombre de sommets restants « nbSommetsRestants » qui est initialisé aux nombres de sommets du graphe.

Par la 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'aura pas été traité. 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 :

1) Insérer dans le tableau de dégénérescence, le K du sommet ;
2) Retirer ce sommet de la liste des voisins avec qui il est voisin, et le marquer lui-même comme traité (avec *null*) ;
3) On décrémente le nombre de sommets restants et on met « trouver » a TRUE pour signaler que nous avons trouvé un sommet et qu'il ne faut pas changer de K, à l'inverse s'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

OPTION : JOLI DESSIN DES NUMEROS DE CENTRE

Afin de pouvoir effectuer l'option que nous avons choisie, 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, nous utilisons deux librairies : une qui permet de pouvoir prendre une photo de la page ; l'autre qui permet de générer des PDF, le combo de ces deux librairies nous a permis de faire la sortie d'une image.

Mais avant de pouvoir sortir une image, nous avons dans un premier temps dû afficher le graphe des numéros de centre, pour cela, nous avons écrit une fonction qui permet de prendre le tableau de dégénérescence qui a été généré 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` applicable à l'affichage.
  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 canvas = renderers.rendererCores.graph;
  canvas.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;

      canvas.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];
      }

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

    diametre--;
  });

  // Afficher la dégénérescence.
  document.getElementById("label-result").innerText = centres.length;
}
```

Figure 5 : Code qui affiche le joli dessin des numéros de centre à l'écran

Par la suite, nous avons donc juste à faire un screenshot de ce joli dessin et mettre cette image dans un 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 SUPPLEMENTAIRE ET EXEMPLE

Pour ce qui est de l'utilisation de l'application, vous avez juste à lancer le « index.html » dans « src », il y a des champs de formulaires pour paramétrer les graphes générés 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 « graphes » où 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 dessin des numéros de centre.

L'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 texte, on peut voir la dégénérescence est de 3 ici.

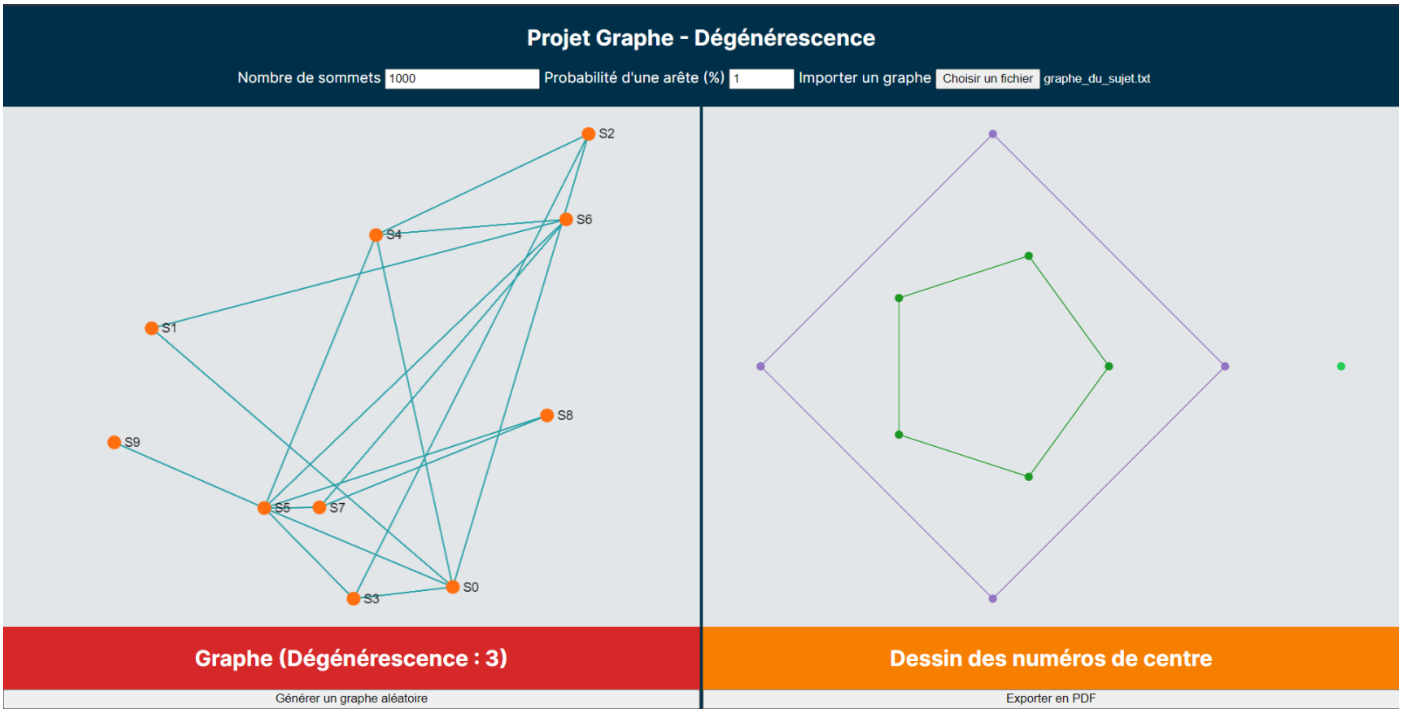

Figure 7 : Image de l'application qui affiche le graphe du sujet

Pour conclure, notre application permet de générer des graphes aléatoires, d'importer des graphes depuis des fichiers (CSV, texte), de calculer la dégénérescence de graphes, d'afficher le graphe avant application de l'algorithme, d'afficher une visuelle des numéros de centre, et 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 suggéré dans les options, ou encore l'option qui permet de pouvoir comparer des graphes en fonction de leur dégénérescence et d'autres paramètres (nombres de chromatique, coloration d'un graphe) …

Lien YouTube de la démonstration :

https://www.youtube.com/watch?v=mjWiRz99-f8