Projet en recherche opérationnelle

Optimisation dans les réseaux optiques

2022

1 Déroulement du projet

Ce projet consiste en l'étude d'un problème via toutes les techniques de recherche opérationnelle que l'on vous a enseigné jusqu'à présent ou que vous connaissez. Il propose des étapes d'étude de l'aspect théorique ou appliqué. Il s'effectue en binôme. Vous devez rendre un code, un rapport et effectuer une soutenance. Le code et le rapport doivent être rendus en temps et la soutenance doit être effectuée sans quoi la note que le binôme se verra attribuée sera 0. En cas d'absence d'un des deux membres à la soutenance, la personne présente se verra le droit de soutenir seule. Si l'absence est justifiée, votre chargé de projet peut s'arranger pour faire soutenir l'autre personne un autre jour. Sinon elle se verra seule attribuée la note de 0.

2 Le rapport

Vous devez rendre le rapport dans une archive rapport.tar.gz. Le rapport en lui même doit être rédigé dans un fichier rapport.pdf. Si vous le souhaitez, vous pouvez ajouter des documents annexes dans un dossier Annexe, mais ces documents ne seront pas nécessairement relus par votre chargé de projet. Le fichier rapport.pdf doit avoir une dizaine de pages maximum (images comprises). Il n'y a pas de minimum de page autre que celui que votre conscience vous impose. Bref, le rapport ne doit être ni insuffisant ni du remplissage inutile. Petit rappel : un rapport comporte une introduction rappelant rapidement le sujet et présentant rapidement la contribution, une conclusion, une page de garde, un sommaire, ... Ce rapport se référera uniquement à la partie 2 (étude commune). Les annexes servent à mettre les preuves trop longues, les tableaux trop grands, ... Les annexes ne comptent pas dans le nombre de page du rapport. Dans tous les cas, le rapport doit se suffira lui même. Par exemple, si vous décidiez de placer une preuve en annexe, il peut être bon de laisser une idée de la preuve dans le rapport. Le rapport sera noté sur le fond et la forme. Le fond regroupe la justesse des preuves, la validité/pertinence des modèles de programmation linéaire et des algorithmes (pourquoi pensez vous que ce sont de bons modèles/algorithmes) et, enfin, la pertinence des évaluations numériques que vous effectuerez. La forme regroupe votre qualité à synthétiser, la qualité des images et des exemples, la clarté des propos, les fautes d'orthographe, ...

3 Le code

Vous allez devoir coder en langage Python. Ce langage devrait vous faciliter les choses et vous permettre d'utiliser les librairies bien utiles dans le cadre de ce projet. Pour homogénéiser les choses et permettre que les séances de TP se déroulent de manière confortable pour le chargé de TP, je vous propose d'utiliser l'IDE Pycharm. Celle-ci est orientée entreprise et est supposée être payante, mais une version académique (pour les enseignants et les étudiants) est disponible gratuitement. Vous pouvez postuler à cette version gratuite sur le lien suivant : https://www.jetbrains.com/fr-fr/community/education/#students Normalement, en renseignant votre adresse mail de l'école, vous allez pouvoir accéder sans délais à la version académique. Bien-entendu, il existe d'autres IDE mais Pycharm est très utilisé actuellement dans les entreprises et il serait utile pour vous de vous familiariser avec cet environnement. Nous allons également avoir besoin de l'éditeur de programme linaire Pulp. Celui-ci va nous permettre d'écrire les PL en python dans des fichiers .LP. Ceux-ci pourront, par la suite, être lus par des solveurs comme Cplex ou bien GLPK. Pour utiliser Pulp sous Pycharm il suffit d'aller dans **préférences** ensuite, au niveau de votre projet, il faut aller dans **Python interpreter** et faire une recherche sur Pulp et puis cliquer sur installer.

Il suffit maintenant d'écrire **import pulp as pl** dans votre fichier.py pour que Pulp fonctionne. Notons que pl est le nom que vous allez donner à votre programme linéaire. Le solveur de programmes linaires que nous allons privilégier est Cplex. Ce solveur, proposé par IBM, est la référence dans le domaine de l'optimisation combinatoire. Il est très largement utilisé par les entreprise malgré son caractère payant. Il existe là aussi, une licence académique gratuite. Pour l'obtenir il faut s'inscrire à l'academic initiative program de l'IBM via ce lien https://www.ibm.com/academic/home Une fois que vous avez téléchargé et installé la dernière version de Cplex, il faut installer l'API Python de Cplex. Ceci est faisable en utilisant la commande : python setup.py install –home yourPythonPackageshome/cplex Il est important de noter que Cplex ne reconnait que la version 2.8 (et inférieur) de python. Il faut donc bien veiller à utiliser cette version. Vous pouvez vérifier la version actuelle utilisée en tapant dans le terminal : python –version. Il est possible d'ajouter le package Cplex directement depuis Pycharm mais ce sera la version gratuite qui sera automatiquement ajoutée, chose qui ne nous intéresse pas ici. Il est cependant possible de faire cela pour GLPK, qui est un solveur gratuit. Il suffit d'aller dans **préférences** ensuite, au niveau de votre projet, il faut aller dans **Python interpreter** et faire une recherche sur glpk et puis cliquer sur installer.
Dans le cadre de ce projet, nous allons utiliser très largement la notion de graphe ainsi que les concepts qui sont sous-jacents. Pour modéliser et manipuler des graphes sous Python, je vous propose d'utiliser la librairie NetworkX est conçu pour fonctionner sur les grands graphes du monde réel, c'est-à-dire par exemple, des graphes de plus de 10 millions de nœuds et 100 millions d'arêtes. En raison de sa dépendance vis-à-vis de la structure de données en « dictionnaire de dictionnaire » (pur Python), NetworkX est raisonnablement efficace, très évolutif, faisant de lui un outil intéressant dans le cadre de l'analyse des réseaux. Tous les grands algorithmes qui résolvent des problèmes de graphes connus y sont disponibles (shortest path, connectivity, clustering, coloring, ...). Cette librairie vous facilitera la manipulation, l'analyse et la visualisations de vos graphes. Comme pour les packages précédents, il suffit d'aller dans **préférences** ensuite, au niveau de votre projet, il faut aller dans **Python interpreter** et faire une recherche sur networkx et puis cliquer sur installer.

4 Les instances

Les instances du problèmes à résoudre se présentent sous forme de graphes. Ce sont des graphes connexes de faible densité pour pouvoir avec des solutions intéressantes (autre que des chemins hamiltoniens). La taille des graphes varie de 20 à 1000 sommets. Pour une taille de graphe et un nombre d'arêtes donnés (densité), 5 graphes ont été générés aléatoirement en utilisant différentes valeurs de graine. Ces instances peuvent être récupérées ici lien. Chaque fichier contient sur la première ligne le nombre de sommets ainsi que le nombre d'arêtes. Chacune des autres lignes contient une arête qui est représentée par ses deux sommets extrémités. C'est instances que l'on peut qualifier de benchmark vont être utiliser pour évaluer vos algorithmes et les comparer aux résultats des autres algorithmes élaborés par les autres groupes.

5 Notation du code

Chacun de vos fichiers algorithmes sera relu puis testé et évalué sur un ensemble d'instances de petite taille. Ces instances ne sont pas exactement celles dont vous disposez dans l'archive instances, mais seront générés de la même façon. Ces tests permettront uniquement de vérifier si les algorithmes répondent correctement aux entrées par rapport à ce qui est indiqué dans le rapport et s'ils sont assez rapides pour résoudre les instances les plus simples. Vos algorithmes seront également testés sur les benchmarks présentés dans l'archive instances mais aussi sur d'autres graphes de même taille mais avec une densité et des particularités qui peuvent être différentes. Ceci me permettra de m'assurer que vous avez réalisé des algorithmes génériques et non spécifiques aux benchmark (en utilisant des algorithmes d'apprentissage par exemple). Vos méthodes seront également testées sur des graphes de plus grande taille (2000, 4000, 6000, ...) pour en détecter les limites. Le time_ limit par instance sera fixé à 10 minutes. Au delà, il est considéré que votre algorithme a échoué à trouver une solution.

6 L'archive à renvoyer

Les archives rapport.tar.gz et code.tar.gz doivent être toutes les deux insérées dans une archive nommée opti2_binome_N.tar.gz ou N est votre numéro de binôme attribué à la première séance. Cette archive sera rendue seule sur exam.ensiie.fr.

7 Dates importantes

— Soutenance : vendredi 13 janvier. Toute absence non justifiée à votre soutenance entrainera la note de 0.

— Rapport et code à rendre au plus tard le mercredi 11 janvier à 23h59. Tout rapport/code non rendu entrainera la note de 0.

8 Projet

Nous évoluons dans un environnement urbanisé autour de réseaux divers, de plus en plus denses et inter-connectés. L'acheminement de manière fiable et à moindre coût de l'information au sein d'un réseau a toujours été une préoccupation permanente de la recherche scientifique; du développement technologique en général et de la recherche opérationnelle en particulier. L'acheminement de l'information se fait par une structure couvrant les terminaux et les liens du réseau utilisés à cet effet. Le projet que je vous propose s'articule autour des problèmes de recherche de structure de recouvrement d'un réseau modélisé par un graphe sous contrainte sur le degré des sommets.

Étant la structure connexe permettant de couvrir les sommets en utilisant un minimum de liens, l'arbre est la structure la plus utilisée dans le domaine des réseaux. L'introduction du broadcast/multicast par Deering dans les années 90 a considérablement augmenté son impact. Deering avait constaté que l'envoi d'un seul message de la source à chaque sommet consommait beaucoup de ressources (bande passante). Il a alors proposé l'envoi par la source d'un unique message, puis de le dupliquer au cours de son acheminement [Dee92]. Dans bon nombre de cas pratiques, borner le degré de chaque sommet de l'arbre de recouvrement de coût minimum est essentiel. La diffusion partielle ou totale dans les réseaux de communication classiques se fait grâce à la duplication de chaque message arrivant à un sommet par le nombre de ses successeurs. Afin de réaliser le routage en un temps acceptable ou d'augmenter la vitesse de celui-ci, il est souhaitable de limiter le nombre de duplications du message routé [ZOK93]. La robustesse du routage dépend également du degré des sommets. En effet, si un sommet de l'arbre a un degré élevé, alors en cas de panne, un grand nombre de sommets se retrouveraient isolés et il serait nécessaire de les relier rapidement à d'autres sommets de l'arbre [DH68].

Le routage broadcast/multicast dans les réseaux optiques constitue l'essentiel de notre préoccupation. La transmission des messages dans ce type de réseaux se fait grâce à la technologie du multiplexage en longueur d'onde communément appelée WDM (Wavelength Division Multiplexing). Cette technologie consiste en la division du spectre optique en plusieurs sous-canaux où chacun est associé à une longueur d'onde spécifique [Gre92]. Il est ainsi possible de faire transiter, dans une même fibre, plusieurs longueurs d'onde en parallèle ou encore ce que l'on peut considérer comme étant plusieurs couleurs en même temps.

Dans les réseaux tout optique, les fonctions de commutation et de routage sont fournies par les brasseurs optique OXC (optical cross-connect) et permettent la mise en œuvre de communications de bout en bout entre les nœuds d'accès. Grâce au démultiplexage du signal optique entrant, un brasseur optique OXC peut commuter chacune des longueurs d'onde d'un port d'entrée vers un port de sortie quelconque. Certains OXC particuliers, appelés OXC-MC, peuvent également diviser une longueur d'onde entrante vers plusieurs ports de sortie à la fois grâce à un coupleur optique, afin d'offrir un service multicast.

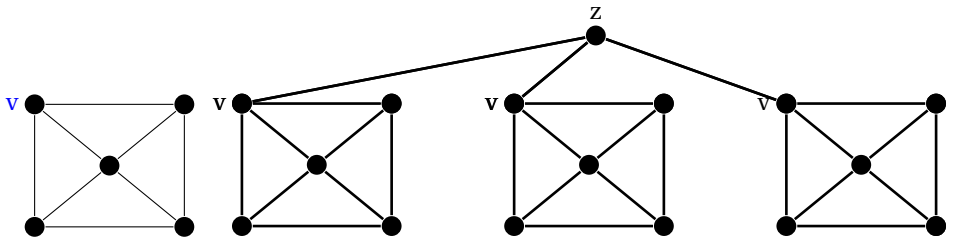
Nous rappelons que dans les réseaux optiques, il existe deux types de commutateurs. Les commutateurs munis de coupleur optique sont dits OXC-MC et ceux qui en sont démunis sont dits OXC-MI. Contrai-

rement aux seconds, les premiers ont une capacité de division du signal lumineux. Naturellement, les commutateurs OXC-MC ont un coût de fabrication supérieur aux autres. Il est donc préférable de minimiser leur nombre dans le réseau. De plus, afin de permettre un routage multicast, il ne suffit pas trouver un arbre de recouvrement du graphe modélisant le réseau ; il est également nécessaire que les sommets de degré strictement supérieurs à deux (sommet de branchement) correspondent à un nœud muni d'un commutateur OXC-MC. Enfin, minimiser le nombre de sommets de branchement peut contribuer à réduire la division du signal lumineux dans le réseau et par conséquent, à réduire la perte de puissance de ce dernier. Gagano et al. ont formellement défini dans [GHSV02] le problème de recherche d'arbre de recouvrement d'un graphe tel que le nombre de sommets de branchement $s(T)$ soit minimum :

Problème : Minimum Branch Vertices Spanning Tree - MBVST
Données : Graphe connexe non-orienté arêtes pondérées $G = (V_G, E_G),\ c(e) \forall e \in E_G$
Résultat : Arbre de recouvrement T du graphe G.
Objectif : Minimiser le nombre de sommets de branchement de T.

8.0.1 Complexité

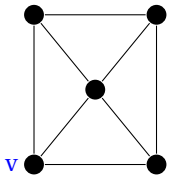
Décider qu'un graphe admet un arbre de recouvrement T tel que $s^*(T) = 0$ revient à décider que ce graphe est hamiltonien, ce qui est NP-complet [GJ79]. L'araignée de recouvrement est un arbre particulier qui contient au plus un sommet de branchement. Décider qu'un graphe admet une araignée de recouvrement est également NP-complet [GHSV02]. En effet, posons $G = (V_G, E_G)$ un graphe connexe et $v \in V_G$. On construit un nouveau graphe G' en dupliquant 3 fois le graphe G. On ajoute un nouveau sommet z, on choisit un sommet quelconque $v \in V_G$, et on relie le sommet v au sommet z par une arête dans les 3 duplications 1. Le graphe G' admet une araignée de recouvrement si et seulement si les 3 duplications du graphe G admettent un chemin hamiltonien ayant v pour extrémité. Or, décider qu'un graphe $G = (V_G, E_G)$ admet un chemin hamiltonien ayant $v \in V_G$ pour extrémité est NP-complet [Kar72].



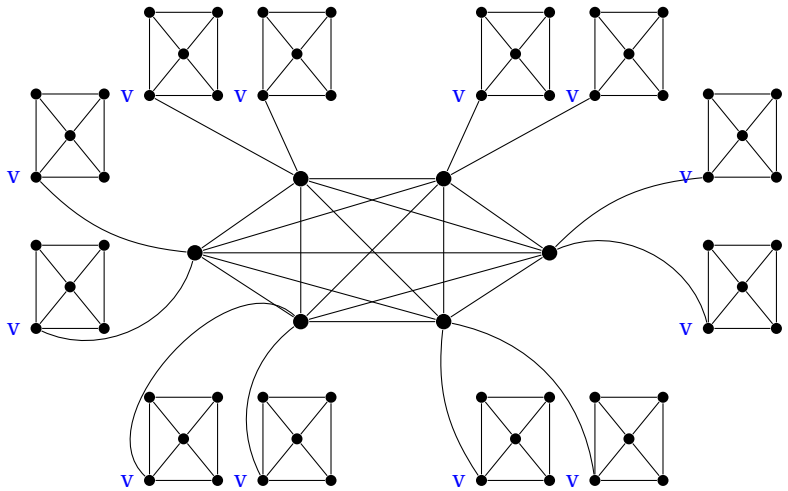
(a) **Le graphe** G (b) **Graphe** G'

FIGURE 1 – Il existe un chemin hamiltonien dans les 4 duplications du graphe G. Le graphe (arbre) G' contient un seul noeud de branchement (le sommet z).

Plus généralement, soit k un entier positif. Il est NP-complet de décider qu'un graphe $G = (V_G, E_G)$ admet un arbre de recouvrement T tel que $s(T) \leqslant k$. Ceci se démontre facilement en généralisant la preuve précédente. Les cas $k = 0$ et $k = 1$ étant démontrés par les résultats précédents, nous pouvons poser $k \geqslant 2$. Étant donné un graphe $G = (V_G, E_G)$, on construit un nouveau graphe G' en dupliquant $2 \cdot k$ fois le graphe G et en ajoutant un graphe complet K_k. On choisit un sommet quelconque v de V_G, et on relie chaque sommet du graphe complet à deux duplications distinctes de G à partir de leur sommet v (figure 2). Comme dans tout arbre de recouvrement de G', les k sommets du graphe complet sont nécessairement des sommets de branchements, il est trivial de constater que le graphe G' admet un arbre de recouvrement ayant k sommets de branchement si et seulement si le graphe G admet un chemin hamiltonien ayant v pour sommet initial.



(a) **Le graphe** G



(b) **Le graphe** G'

FIGURE 2 – Construction du graphe G' pour $k = 6$.

Dans la classe des graphes cubiques, le problème MBVST reste NP-complet [GHSV02]. Cette affirmation résulte de la relation entre le problème MBVST et le problème LONGEST PATH qui consiste à trouver un chemin de longueur maximum d'un graphe. Notons $\ell(G)$ la longueur d'un tel chemin. Les deux problèmes sont liés par l'équation la suivante :

$$\ell(G) \geqslant \frac{|V|}{s^*(G)+1} \tag{1}$$

Étant donné un graphe $G = (V_G, E_G)$ et un entier positif k, Bazgan et al. démontrent dans [BST99], qu'il n'existe pas d'algorithme polynomial qui permette de trouver un chemin de longueur maximum de G tel que $\ell(G) \geqslant |V|/k$ même si G est cubique. Par conséquent, la valeur de $s^*(G)$ dans l'inégalité 1 ne peut être trouvée en temps polynomial.

Décider de l'existence d'une araignée de recouvrement devient facile sous certaines conditions. Il est démontré dans [GHH$^+$04] que si $\delta(G) \geqslant (n-1)/3$ (plus généralement, $\delta_3(G) \geqslant (n-1)$), alors G contient une araignée de recouvrement et il existe un algorithme polynomial ($O(|V_G|^3)$) qui la trouve. Dans [Dir52], Dirac prouve que si la somme des degrés de n'importe quelle paire de sommets indépendants de G est au moins égale à $|V_G|-1$, alors G contient un chemin hamiltonien. Dans [GH03], Gargano et al. se réfèrent aux travaux de Dirac pour prouver que si la somme des degrés de n'importe quel triplet de sommets est au plus égale à $|V_G|-1$, alors G contient une araignée de recouvrement et il existe un algorithme polynomial ($O(|V_G|^3)$) qui la trouve. Étant donné un graphe connexe $G = (V_G, E_G)$ et un sommet $u \in V_G$, Flandrin et al. prouvent dans [FKK$^+$08] que si $\delta_2^u(G) \geqslant |V_G|-1$, alors G contient une araignée de recouvrement centrée sur u. Dans [GHSV02], une conjecture généralisant le théorème de Dirac est

proposée : si chaque sommet d'un graphe $G = (V_G, E_G)$ est de degré au moins $\frac{|V_G|-1}{k+2}$ (plus généralement, si la somme des degrés de chaque ensemble de $k+1$ sommets indépendants de G est d'au plus $|V_G|-1$), alors $s^*(G) \leqslant k$. Cette conjecture est prouvée pour $k = 0$ et $k = 1$, mais reste cependant ouverte pour $k \geqslant 2$.

Il est démontré que la borne sur le degré suffisante pour l'existence d'un chemin hamiltonien peut être améliorée pour les graphes bipartis [Ber85]. Il en est de même pour l'existence de l'araignée de recouvrement [GH03]. En effet, soit $G(V1_G, V2_G, E_G)$ un graphe biparti connexe tel que $|V1_G| \leqslant |V2_G|$, si pour tout $u \in V1_G$ et $v \in V2_G$, $d_G(u) + d_G(v) \geqslant |V2_G|$ alors, pour tout $u \in V1_G$, le graphe G contient une araignée de recouvrement centrée sur u.

Un graphe $K_{1,3}$ est appelé $claw$. Un graphe est dit $claw\text{-}free$ s'il ne contient pas $K_{1,3}$ comme sous-graphe induit. Il est prouvé dans [MOY14] que si dans un graphe $claw\text{-}free$ de taille n, la somme des degrés de tout ensemble de cinq sommets indépendants est supérieure ou égale à $n-2$, alors il existe un arbre de recouvrement de ce graphe avec au plus un sommet de branchement. Cette condition de densité d'un graphe $claw\text{-}free$, qui permet de répondre par l'affirmative à l'existence d'une araignée de recouvrement de celui-ci, est la meilleure possible.

9 Résolution exacte et approchée

La résolution exacte du problème MBVST est réalisée en utilisant la programmation linéaire en nombre entier. La résolution approchée est réalisée en affectant des relaxations des programmes linéaires ou bien à l'aide d'algorithmes d'approximations classiques. Ce problème n'appartient pas à la classe d'approximation APX. Il est de ce fait non-approximable à facteur constant près [BGPea97]. Les schémas d'approximations proposés dans la littérature sont donc sans garantie de performance.

9.1 Utilisation de la programmation linéaire

Plusieurs formulations mathématiques du problème MBVST ont été proposées. Une formulation en programme linéaire en nombres entiers avec un nombre exponentiel de contraintes d'une part ; une deuxième formulation en programme linéaire en nombres entiers basée sur le concept de flot et enfin une formulation en programme linéaire en nombres entiers basée sur le concept de flots multiples.

1. **Formulation en programme linéaire en nombres entiers (PLNE) avec un nombre exponentiel de contraintes [CCGG13] :** Soit un graphe connexe $G = (V_G, E_G)$. L'ensemble des variables est le suivant : une variable binaire x_e pour chaque arête $e \in E_G$, qui vaut 1 si l'arête e est sélectionnée, 0 sinon. Une autre variable binaire y_v pour chaque sommet $v \in V_G$ qui est égale à 1 si le sommets v est un sommet de branchement, 0 sinon. Le programme se présente comme suit :

$$min \sum_{v \in V_G} y_v \tag{2}$$

$$\sum_{e \in E_G} x_e = |V_G| - 1 \tag{3}$$

$$\sum_{e \in E(S)} x_e \leqslant |S| - 1 \qquad \forall S \subseteq V_G \tag{4}$$

$$\sum_{e \in \lambda_G(v)} x_e - 2 \leqslant d_G(v) y_v \qquad \forall v \in V_G \tag{5}$$

$$y_v \in \{0,1\} \qquad \forall v \in V_G \tag{6}$$

$$x_e \in \{0,1\} \qquad \forall e \in E_G \tag{7}$$

Pour chaque sous ensemble $S \subseteq V_G$, on note $E(S)$ l'ensemble des arêtes ayant leurs deux extrémités dans S. La fonction objective (2) vise à minimiser le nombre total de sommets de branchement. La contrainte (3) garantit la présence de exactement $|V| - 1$ arêtes dans l'arbre de recouvrement. La contrainte (4) garantit que l'arbre de recouvrement est sans cycle. La contrainte (5) assure que la variable y_v est égale à 1 ssi le degré du sommet v est strictement supérieur à 2 dans l'arbre de

recouvrement.

2. **Formulation en programme linéaire en nombres entiers basée sur le concept de flot(PLNE-CP) [CGI09] :** Pour la formulation basée sur le flot, on considère la version orientée symétrique du graphe où deux arcs (u,v) et (v,u) sont associés à chaque arête $\{u,v\} \in E_G$. On note par E' ce nouvel ensemble d'arcs. On note $A^+(v)$ et $A^-(v)$ l'ensemble des arcs de E' sortant et entrant dans v, respectivement. Des variables de flot sont également introduites. On note par $f_{u,v}$ la quantité de flot transitant dans l'arc $(u,v) \in E'$. Une source est choisie de manière arbitraire parmi tous les sommets du graphe et une unité de flot est envoyée à partir de cette source à destination de chaque sommet $v \in V_G \setminus \{s\}$. Les arcs faisant transiter une quantité de flot non nulle correspondent aux arêtes de l'arbre de recouvrement. Ceci permet de garantir la connexité de l'arbre de recouvrement construit.

$$min \sum_{v \in V_G} y_v \tag{8}$$

$$\sum_{(u,v) \in A^-(v)} x_{u,v} = 1 \tag{9}$$

$$\sum_{(s,v) \in A^+(s)} f_{s,v} - \sum_{(v,s) \in A^-(s)} f_{v,s} = |V_G| - 1 \tag{10}$$

$$\sum_{(v,u) \in A^+(v)} f_{v,u} - \sum_{(u,v) \in A^-(v)} f_{u,v} = -1 \qquad \forall v \in V_G \setminus \{s\} \tag{11}$$

$$x_{u,v} \leqslant f_{u,v} \leqslant (|V_G|) \cdot x_{u,v} \qquad \forall (u,v) \in E'_G \tag{12}$$

$$\sum_{(v,u) \in A^+(v)} x_{v,u} + \sum_{(u,v) \in A^-(v)} x_{u,v} - 2 \leqslant d_G(v) y_v \qquad \forall v \in V_G \tag{13}$$

$$y_v \in \{0,1\} \qquad \forall v \in V_G \tag{14}$$

$$x_e \in \{0,1\} \qquad \forall (u,v) \in E'_G \tag{15}$$

$$f_{u,v} \geqslant 0 \qquad \forall (u,v) \in E'_G \tag{16}$$

La fonction objectif (8) vise toujours à minimiser le nombre de sommets de branchement de l'arbre de recouvrement. La contrainte (9) assure que chaque sommet de l'arbre de recouvrement a exactement un arc entrant. Les contraintes (10) et (11) assurent qu'une unité de flot soit envoyée depuis la source à chaque sommet de l'arbre de recouvrement, garantissant ainsi la connexité de toute solution réalisable. La contrainte (12) relie les variables de flot f aux variables binaires x. La contrainte (13) garantit que la variable y_v est égale à 1 si et seulement si le degré du sommet v est strictement supérieur à 2 dans l'arbre de recouvrement.

3. **Formulation en programme linéaire en nombres entiers basée sur le concept de flots multiples (PLNE-CPM) [CCGG13] :** Le problème MBVST peut également être formulé en utilisant le concept de flots multiples [BMMN95]. Pour ce faire, en introduisant le même ensemble d'arc E', nous introduisons, pour chaque arc $(u,v) \in E'$ un ensemble de variables de flot $f^k_{u,v}$, où $k \in V_G$ représente le flot passant par l'arc (u,v) relatif au chemin allant de la source arbitraire s au sommet k.

$$min \sum_{v \in V_G} y_v \tag{17}$$

$$\sum_{(u,v) \in A^-(v)} x_{u,v} = 1 \qquad \forall v \in V_G \setminus \{s\} \tag{18}$$

$$\sum_{(v,u) \in A^+(v)} f^k_{v,u} - \sum_{(u,v) \in A^-(v)} f^k_{u,v} = 0 \qquad \forall v \in V_G,\ v \in V_G \setminus \{s\},\ v \neq k \tag{19}$$

$$\sum_{(s,v) \in A^+(s)} f^k_{s,v} - \sum_{(v,s) \in A^-(s)} f^k_{v,s} = 1 \qquad \forall v \in V_G \setminus \{s\} \tag{20}$$

$$\sum_{(k,u) \in A^+(k)} f^k_{k,u} - \sum_{(i,k) \in A^-(k)} f^k_{i,k} = -1 \qquad \forall k \in V_G \setminus \{s\} \tag{21}$$

$$f^k_{u,v} \leqslant x_{u,v} \qquad \forall (u,v) \in E'_G,\ k \in V_G \tag{22}$$

$$\sum_{(v,u) \in A^+(v)} x_{v,u} + \sum_{(u,v) \in A^-(v)} x_{u,v} - 2 \leqslant d_G(v) y_v \qquad \forall v \in V_G \tag{23}$$

$$y_v \in \{0,1\} \qquad \forall v \in V_G \tag{24}$$

$$x_{u,v} \in \{0,1\} \qquad \forall (u,v) \in E'_G \tag{25}$$

$$f^k_{u,v} \geqslant 0 \qquad \forall (u,v) \in E_G,\ k \in V_G \tag{26}$$

L'objectif est identique à celui des modèles précédents. La contrainte (18) assure que chaque sommet de l'arbre de recouvrement a exactement un arc entrant. Les contraintes de conservation de flot (19)-(21) garantissent que la source s est connectée à tous les sommets du graphe. La contrainte (22) relie les variables de flot aux variables binaires d'arcs. La contrainte (23) est la même que dans le modèle précédent.

Formulation de Martin Étant donné un graphe $G = (V, E)$, on note par $\gamma(v)$ le degré du sommet v dans G. La variable x_{ij} pour chaque $(i, j) \in E$ vaut 1 si l'arête (i, j) est dans la solution, elle vaut 0 sinon. La variable $y^k_{i,j}$ pour chaque $(i, j) \in E$ et $k \in V$ vaut 1 si l'arête (i, j) est dans la solution et le sommet k est dans la composante connexe contenant j après la surpression de (i, j), elle vaut 0 si l'arête (i, j) n'est pas dans la solution ou bien si l'arête (i, j) est dans la solution mais le sommet k n'est pas dans la composante connexe contenant j. Les contraintes (1a), (1b), (1c) constituent la formulation de Martin [Mar91]. La contrainte (1d) est ajoutée pour garantir que la variable z_v, pour chaque $v \in V$, vaut 1 quand le sommet v a un degré strictement supérieur à k+2, elle vaut 0 sinon.

$$\min \sum_{v \in \mathcal{V}} z_v$$

$$s.t. \begin{cases} \displaystyle\sum_{(i,j) \in E} x_{i,j} = n - 1 & & \text{(27a)} \\ y^k_{ij} + y^k_{ji} = x_{ij} & \forall (i,j) \in E, k \in V & \text{(27b)} \\ \displaystyle\sum_{k \in \mathcal{V} \setminus \{i,j\}} y^j_{ik} + x_{i,j} = 1 & \forall (i,j) \in E & \text{(27c)} \\ \displaystyle\sum_{(i,j) \in \gamma(i)} x_{ij} - |\gamma(i)| z_i \leqslant 2 + k & \forall i \in \mathcal{V} & \text{(27d)} \\ x_{ij}, y^k_{ij}, y^k_{ji} \in \{0,1\} & & \text{(27e)} \end{cases}$$

9.2 Utilisation d'heuristiques

Dans [CGI09], trois heuristiques gloutonnes sont proposées pour résoudre de façon approchée (sans garantie de performance) le problème MBVST.

1. **Heuristique basée sur le poids des arêtes :** L'idée de cette heuristique est d'ajouter une fonction $\omega : E_G \to \mathbb{N}^+$ qui attribue un poids à chaque arête $\{u, v\} \in E_G$. Le poids $\omega(u, v)$ de l'arête $\{u, v\}$ est utilisé comme indicateur de la probabilité que cette arête crée un sommet de branchement si elle est sélectionnée pour être dans l'arbre de recouvrement. L'algorithme commence par assigner le

poids 1 à chaque arête du graphe G. À chaque itération, une arête $\{u^*, v^*\}$ ayant un poids minimum est sélectionnée à condition que cette arête ne soit pas préalablement dans l'arbre T, et que le sommet u^* et le sommet v^* soient dans deux composantes connexes différentes. Une fois cette arête sélectionnées, elle est ajoutée à l'arbre T et supprimée du graphe G. Le poids de chaque arête incidente à u^* ou à v^* est alors incrémenté de 1, ce qui a pour effet de diminuer leur probabilité d'être sélectionnée à l'itération suivante. L'algorithme s'arrête quand $|V_G| - 1$ arêtes sont sélectionnées.

2. **Heuristique basée sur les couleurs des sommets :** Cette approche est basée sur l'assignation d'une couleur à chaque sommet du graphe $c : v \longrightarrow \{V(ert), B(leu), J(aune), R(ouge)\}$. La couleur $c(v)$ d'un sommet v indique si ce sommet deviendra sommet de branchement quand une arête incidente à celui-ci est sélectionnée pour être insérée dans l'arbre de recouvrement. Une couleur est attribuée à un sommet comme suit :

$$V, \text{ si } d_T(v) = 0 \tag{28}$$
$$B, \text{ si } d_T(v) = 1 \tag{29}$$
$$J, \text{ si } d_T(v) = 2 \tag{30}$$
$$R, \text{ si } d_T(v) \geqslant 3 \tag{31}$$

   L'algorithme assigne la couleur verte à chaque sommet $v \in V_G$. À chaque itération, une arête est sélectionnée en fonction des couleurs de ces extrémités pour être insérée dans l'arbre de recouvrement. Notons que le nombre de nouveaux sommets de branchement créés après l'insertion de l'arête $\{u, v\}$ est le nombre de sommets jaunes constituants ses deux extrémités. L'algorithme choisit ainsi une arête ayant un nombre minimum d'extrémités jaunes. Afin de minimiser le nombre de nouveaux sommets jaunes potentiels, l'algorithme sélectionne, parmi les arêtes ayant un nombre minimum d'extrémités jaunes, celles ayant le moins de sommets bleus comme extrémités. L'algorithme s'arrête quand $|V_G| - 1$ arêtes sont sélectionnées.

3. **Heuristique basée sur la combinaison des deux approches précédentes :** Cette approche propose une sélection d'arêtes encore plus précise en combinant les deux approches précédentes. À chaque itération, l'ensemble des arêtes ayant un poids minimum est sélectionné. Parmi ces arêtes, celles ayant un nombre minimum d'extrémités coloriées en jaune sont gardées. Parmi ces dernières, celles ayant un nombre minimum d'extrémités coloriées en bleu sont enfin sélectionnées pour être dans l'arbre de recouvrement.

Des séries de tests sur des graphes aléatoires ont permis d'évaluer les performances des trois heuristiques. Sans surprise, l'heuristique basée sur la combinaison des deux approches donne les meilleurs résultats. Malgré le fait qu'il n'existe pas de garantie de performance, elle retourne, dans le pire des cas, une solution approchée de coût égal à trois fois celui de la solution optimale. Elle retourne une solution optimale dans 22% des instances testées.

Dans [SSM+11], une heuristique basée sur les travaux de [DK97] est proposée. Cette heuristique commence par calculer un arbre de poids minimum T du graphe G. Cet arbre peut contenir plusieurs sommets de branchement. L'arbre T sera modifié de manière itérative pour réduire le nombre de sommets de branchement. À chaque itération, une liste d'arêtes L_{cut} à remplacer est constituée. Cette liste est formée des arêtes dont au moins une extrémité est un sommet de branchement. Une arête de cette liste est sélectionnée en se basant sur deux critères. Le premier critère α est le nombre d'extrémités de l'arête qui sont sommets de branchement. Le deuxième critère σ est la somme des degrés de ses extrémités. L'arête de L_{cut} ayant la plus grande valeur de α (et la plus grande valeur σ en cas d'égalité) est retirée de l'arbre T, ceci divisant l'arbre T en deux composantes connexes. Chaque arête retirée est sauvegardée dans une liste L_{list} afin de ne pas la réinsérer dans l'arbre et d'éviter ainsi les cycles. La liste L_{rep} contient toutes les arêtes qui peuvent connecter les deux composantes connexes et qui ne sont pas dans L_{list}. L'arête de L_{rep} ayant la plus petite valeur de α (et la plus petite valeur de σ en cas d'égalité) est ajoutée à l'arbre T permettant ainsi de connecter les deux composantes.

Sans garantie de performance, il est difficile de hiérarchiser les performances des heuristiques. Cette heuristique retourne cependant un meilleur résultat que les trois précédentes dans 78% des instances testées.

10 Réduction de la complexité

Nous proposons, dans ce qui suit, une partition des sommets de l'instance de ce problème basée sur les travaux de [LMSP17]. Soit $G = (V, E)$ un graphe connexe, nous notons par $d_G(v)$ le degré de v dans G, et par $C_G(v)$ le nombre de composantes connexes dans le graphe $\mathscr{G}$ privé de v. Nous décomposons les sommets de $\mathscr{G}$ en fonction de leur possibilité d'être des sommets de k-branchement dans la solution optimale du problème k-MBVST. Ainsi, l'ensemble V est partitionné en 4 ensembles V_0, V_1, V_2, V_3 comme suit :

$$V_0 = \{v \in V : d_G(v) = 1\}$$
$$V_1 = \{v \in V : (1 < d_G(v) < 3)\}$$
$$V_2 = \{v \in V : (d_G(v) > k+2) \text{ et } (C_G(v) \leqslant 2\}$$
$$V_3 = \{v \in V : (C_G(v) > 2)\}$$

Clairement, les sommets de V_0 et de V_1 n'ont pas un degré suffisant pour être des sommets de k-branchement dans tout arbre couvrant de G. Supprimer un sommet de l'ensemble V_3 décompose le graphe G en au moins $k+3$ composantes connexes, ses sommets sont donc nécessairement des sommets de k-branchement dans tout arbre couvrant de G. Bien que la suppression d'un sommet de l'ensemble V_2 ne décompose le graphe G qu'en $k+2$ composantes connexes au plus, un tel sommet peut être sommet de k-branchement dans une solution optimale pour le problème k-MBVST. Pour trouver une telle solution, il suffit donc de décider quels sont les sommets de V_2 qui sont des sommets de k-branchement dans une solution optimale.

Travail à réaliser

1. Programmer les programmes linaires basés sur le flot ainsi que celui de Martin en utilisant la libraire Pulp, tout en prenant en compte la partition des sommets.
2. Résoudre les PL en utilisant Cplex sur toutes les instances du fichier Instances (si possible. Time limit = 60 secondes).
3. Programmer un schéma de résolution basé sur la programmation linéaire et les bases de cycles (je vais l'expliquer au tableau).
4. prouver que ce schéma de résolution est exact ou bien approchée (heuristique).
5. Appliquer au moins deux algorithmes d'apprentissage sur un échantillon des graphes (30%) que vous avez résolu de manière exacte en choisissant de bonnes variables explicatives. LE but ici est de prédire avec une certaines probabilité quelles sont les arêtes qui feront partie de la solution.
6. Construire un algorithme qui utilise la prédiction d'arête de pour trouver la meilleure solution possible.
7. Faire une comparaison globale (qualité et temps de calcul) sur tous les algorithmes que vous avez codés.
8. Réaliser un rapport de votre travail en Latex.

Références

[Ber85] C. Berge. *Graphs and Hypergraphs.* Elsevier Science Ltd, 1985.

[BGPea97] B. Beauquier, L. Gargano, S. Perennes, and et al. Graph problems arising from wavelength-routing in all-optical networks. Proc. 2nd Workshop on Optics and Computer Science, Geneva, 1997.

[BMMN95] M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser. Optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995.

[BST99] C. Bazgan, M. Santha, and Z. Tuza. On the approximation of finding a(nother) hamiltonian cycle in cubic hamiltonian graphs. *Journal of Algorithms*, 31(1) :249–268, 1999.

[CCGG13] F. Carrabs, R. Cerulli, M. Gaudioso, and M. Gentili. Lower and upper bounds for the spanning tree with minimum branch vertices. *Computational Optimization and Applications*, 56(2) :405–438, 2013.

[CGI09] R. Cerulli, M. Gentili, and A. Iossa. Bounded-degree spanning tree problems : models and new algorithms. *Comput. Optim. Appl*, 42 :353–370, April 2009.

[Dee92] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford, CA, USA, 1992. UMI Order No. GAX92-21608.

[DH68] N. Deo and L. Hakimi. The shortest generalized hamiltonian tree. In 6^{th} *Annual Allerton Conference*, pages 879–888, Illinois, USA, 1968.

[Dir52] G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, s3-2(1) :69–81, 1952.

[DK97] N. Deo and N. Kumar. Computation of constrained spanning trees : A unified approach. In P. Pardalos, D. Hearn, and W. Hager, editors, *Network Optimization*, volume 450 of *Lecture Notes in Economics and Mathematical Systems*, pages 196–220. Springer Berlin Heidelberg, 1997.

[FKK+08] E. Flandrin, T. Kaiser, R. Kužel, H. Li, and Z. Ryjáček. Neighborhood unions and extremal spanning trees. *Discrete Mathematics*, 308(12) :2343 – 2350, 2008.

[GH03] L. Gargano and M. Hammar. There are spanning spiders in dense graphs (and we know how to find them). In J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 802–816. Springer Berlin Heidelberg, 2003.

[GHH+04] L. Gargano, M. Hammar, P. Hell, L. Stacho, and U. Vaccaro. Spanning spiders and light-splitting switches. *Discrete Mathematics*, 285(1–3) :83 – 95, 2004.

[GHSV02] L. Gargano, P. Hell, L. Stacho, and U. Vaccaro. Spanning trees with bounded number of branch vertices. In P. Widmayer, S. Eidenbenz, F. Triguero, R. Morales, R. Conejo, and M. Hennessy, editors, *Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 355–365. Springer Berlin Heidelberg, 2002.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman Co, New York, NY, USA, 1979.

[Gre92] P. E. Green. *Fiber Optic Networks*. Prentice-Hall, Cambridge, 1992.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[LMSP17] M. Landete, A. Marín, and J. Sainz-Pardo. Decomposition methods based on articulation vertices for degree-dependent spanning tree problems. *Computational Optimization and Applications*, 68(3) :749–773, Dec 2017.

[Mar91] R.Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Oper. Res. Lett.*, 10(3) :119–128, 1991.

[MOY14] H. Matsuda, K. Ozeki, and T. Yamashita. Spanning trees with a bounded number of branch vertices in a claw-free graph. *Graphs and Combinatorics*, 30(2) :429–437, 2014.

[SSM+11] D. Silva, R. Silva, G. Mateus, J. Gonçalves, M. Resende, and P. Festa. An iterative refinement algorithm for the minimum branch vertices problem. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 421–433. Springer Berlin Heidelberg, 2011.

[ZOK93] W. De Zhong, Y. Onozato, and J. Kaniyil. A copy network with shared buffers for large-scale multicast. *ATM Switching," IEEE/ACM Transactions on Networking*, pages 157–165, 1993.

A Notations

On note $G = (V_G, E_G)$ un graphe pouvant être orienté ou non orienté. L'ensemble V_G est un ensemble fini non vide d'éléments, les éléments de V_G sont appelés **sommets**. E_G est un ensemble tel que $E_G \subseteq V_G \times V_G$, les éléments de E_G sont appelés **arcs** (notés $e = (u, v)$) dans le cas orienté et **arêtes** (notées $e = \{u, v\}$) dans le cas non-orienté. On note $n =\mid V_G \mid$ le nombre de sommets (la taille du graphe) et $m =\mid E_G \mid$ le nombre d'arêtes du graphe G. Si une fonction $C : E_G \rightarrow \mathbb{R}_*^+$ associe un coût $C_G(e)$ à chaque arête $e \in E_G$ alors le graphe G est dit arêtes pondérées. Si une fonction $C' : V_G \rightarrow \mathbb{R}_*^+$ associe un coût $C'_G(v)$ à chaque sommets $v \in V_G$ alors le graphe est dit sommets pondérés.

Degré et voisinage Un sommet $v \in V_G$ est voisin du sommet $u \in V_G$ dans G si l'arête $\{v, u\} \in E_G$. On note $\lambda_{\mathbf{G}}(\mathbf{v}) \subseteq V_G$ l'ensemble des voisins de v dans G. On appelle degré d'un sommet v le cardial de l'ensemble de ses voisins et le note $\mathbf{d_G}(\mathbf{v}) = |\lambda_G(v)|$. Plus généralement, soit un ensemble de sommets $X \subseteq V_G$, $\mathbf{d_G}(\mathbf{X}) = \sum_{v \in X} d_G(v)$. On note $\Delta(\mathbf{G}) = max\{d_G(v) \setminus v \in V_G\}$ le plus grand degré dans G et par $\delta(\mathbf{G}) = min\{d_G(v) \setminus v \in V_G\}$ le plus petit degré dans G. On note $\mathbf{V_d}(\mathbf{G})$ l'ensemble des sommets de degré d dans G. Un graphe G est **d-regulier** si tous les sommets de G ont un degré égal à d ($V_d(G) = V(G)$). Un graphe 3-$regulier$ est aussi appelé **graphe cubique**. Si $d_G(v) = 1$ alors le sommet v est appelé **feuille** et si ce sommet a un degré $d_G(v) > 1$ alors il est dit **sommet interne**. On note $\mathbf{F}(\mathbf{G})$ l'ensemble des feuilles de G et $\mathbf{IN}(\mathbf{G})$ l'ensemble des sommets internes. On a par définition $F(G) \cup IN(G) = V_G$. On note $\mathbf{L}(\mathbf{G}) = |F(G)|$

Chemins et parcours Un chemin de u à v dans G, noté $\mathbf{CH_G}(\mathbf{u}, \mathbf{v})$, est une séquence de sommets $< v_0, ..., v_k >$ avec $v_0 = u$ et $v_k = v$ et telle que $\forall i, 0 \leqslant i < k$, $\{v_i, v_{i+1}\} \in E_G$. Le chemin $CH_G(u, v) =< v_0, ..., v_k >$ est dit de **longueur** $k-1$. Les sommets u et v sont les **extrémités** de $CH_G(u, v)$. On note par $\ell(\mathbf{G})$ la longueur du chemin le plus long dans G. Un chemin est élémentaire si chacun des sommets de la séquence le composant y est présent une seule fois, il est non-élémentaire sinon. Un chemin est dit **hamiltonien** s'il contient tous les sommets du graphe et qu'il est élémentaire. Un graphe contenant un chemin hamiltonien est appelé graphe hamiltonien. Un parcours $\mathbf{P}(\mathbf{G})$ d'un graphe G est un chemin qui contient tous les sommets du graphe une ou plusieurs fois et qui peut passer plusieurs fois par chaque arête. Si un parcours passe au plus une fois par chaque arête alors il est élémentaire. Si, par contre, il passe plus d'une fois par chacune des arêtes de G alors il est dit non-élémentaire. S'il passe une et une seule fois par chaque arête de G, le **parcours est dit eulérien**. Un graphe ayant un tel parcours est un **graphe eulérien**. Un graphe est **connexe** s'il existe un chemin entre chaque paire de ses sommets.

Cycles Un cycle dans G est un chemin $CH_G(u, v)$ tel que $v = u$. Si un cycle contient tous les sommets du graphe, alors ce cycle est dit hamiltonien. Un graphe sans cycle est dit acyclique.

Sous-graphes Un sous graphe G' de G est un graphe tel que $V_{G'} \subseteq V_G$ et $E_{G'} \subseteq E_G$. Si $V_{G'} \subset V_G$ et $E_{G'} = \{\{u, v\} \in E_G \mid u \in V_{G'}\ et\ v \in V_{G'}\}$ alors G' est un **sous-graphe induit** de G.

Arbres Un arbre $T = (V_T, E_T)$ est un graphe connexe acyclique. Si $V_T = V_G$ et $E_T \subseteq E_G$ alors T est dit **arbre de recouvrement** de G.

Sommets de branchement Un sommet de branchement est un sommet de degré strictement supérieur à deux ($d_G(v) > 2$). On appelle $\mathbf{SB}(\mathbf{G})$ l'ensemble des sommets de branchement du graphe G. On note $\mathbf{s}(\mathbf{G}) = |s(G)|$ le nombre de sommets de branchement de G. On note $\mathbf{m}(\mathbf{G})$ la somme des degrés des sommets de branchement de G ($m(G) = \sum_{v \in SB(G)} d_G(v)$). On dénote par $\mathbf{s}^*(\mathbf{G})$ le plus petit nombre de sommets de branchement que peut avoir un arbre de recouvrement de G. On note par $\mathbf{m}^*(\mathbf{G})$ la somme minimum des degrés des sommets de branchement que peut avoir un arbre de recouvrement de G. Enfin, nous notons par $\mathbf{L}^*(\mathbf{G})$ le nombre de feuilles minimum que peut avoir un arbre de recouvrement de G.

Araignées Une araignée est un arbre de recouvrement de celui-ci ayant au plus un sommet de branchement. Si, de plus, la distance entre toute paire de feuilles est au plus égale à deux, alors on parlera **d'étoile**.

Sommets indépendants et graphes bipartis Deux sommets $(v,u) \in V_G^2$ sont dits indépendants dans G s'il n'existe pas d'arête $\{v,u\} \in E_G$ les reliant. Un ensemble de sommets est dit indépendant si chaque paire de sommets de l'ensemble est indépendante. On note par $\delta_{\mathbf{k}}(\mathbf{G})$ la somme minimum des degrés de k sommets indépendants dans G. On note $\delta_{\mathbf{k}}^{\mathbf{u}}(\mathbf{G}) = min \sum_{v \in I} d_G(v)$ tel que I est un ensemble de sommets de G de taille k et $I \cup \{u\}$ est indépendant. Notons que $\delta_1^u(G)$ est simplement le degré minimum des sommets non-adjacents à u. Un graphe $G = (V_G, E_G)$ dont l'ensemble de sommets peut être partitionné en deux ensembles $V1_G$ et $V2_G$, tel que $V1_G = V2_G \cup + V1_G \cap V2_G = \emptyset$, chacun indépendant est appelé graphe biparti. On note $G = (V1_G, V2_G, E_G)$ un tel graphe. Remarquons que tout arbre est un graphe biparti.

Graphes complets Un graphe $G = (V_G, E_G)$ est dit complet si pour tout $(u,v) \in V_G^2$, $\{u,v\} \in E_G$. On note un tel graphe K_n, où n représente sa taille. Un graphe biparti $G = (V1_G, V2_G, E_G)$ est dit complet si $\forall u \in V1_G, \forall v \in V2_G$, $\{u,v\} \in E_G$. Un graphe biparti complet est noté K_{n_1,n_2}, où n_1 et n_2 représentent respectivement les cardinaux de ses deux ensembles de sommets.