Optimisation des réseaux

Nous traitons dans cette thèse plusieurs problèmes d'optimisation combinatoire qui se posent dans le cadre de l'optimisation des réseaux. Il convient donc de décrire brièvement l'optimisation des réseaux.

En réalité la tâche est ardue pour plusieurs raisons. En effet, d'une part les réseaux sont multiples (plusieurs technologies, plusieurs architectures). D'autre part, les problèmes sont différents. Enfin, les outils mathématiques qui sont utilisés sont également très divers.

Pour ne pas nous perdre dans les détails, nous allons nous contenter de présenter très brièvement les types de problèmes qu'on cherche généralement à résoudre pour optimiser des réseaux.

Un réseau a une structure. Le problème de base consiste à définir la structure ou la topologie du réseau. Ceci revient à imposer des contraintes de connexité, voire de k-connexité. On peut également considérer des contraintes de diamètre pour imposer que les communications se fassent en un temps limité (voir [1], [5], [34], [35], [55], [77]). D'autres problèmes de topologie consistent à localiser des équipements d'un type particulier (concentrateurs, noeuds d'accès, etc) [81], [126]. Construire un réseau sous la forme d'un ensemble de réseaux d'accès connectés par une dorsale est un autre exemple de problème de topologie : comment faire ce découpage d'une manière optimale ?. Construire donc la topologie d'un réseau donne lieu à un ensemble de problèmes combinatoires très divers et souvent difficiles à résoudre. Tout devient plus complexe si on intègre un aspect dynamique de configuration des réseaux : des noeuds qui se déplacent ou qui changent de comportement induisant un changement de la topologie. Ré-optimiser la topologie en respectant un certain nombre de contraintes techniques est un challenge d'actualité.

Une fois la topologie fixée, il faut relayer du trafic dans le réseau. En d'autres termes, il faut résoudre des problèmes de routage. Ceci se ramène souvent à des problèmes de flot dans un graphe. Plusieurs types de contraintes peuvent être prises en compte : routage de chaque demande sur un seul chemin [7], routage sur plus court chemin [8], [41], routage avec contraintes de délai [6], routage avec contrainte d'équité [105]. On peut également considérer des contraintes de sécurisation se traduisant par l'obligation de rerouter le trafic en cas de panne de certaines composantes du réseau [10], [91], [99]. Tous ces problèmes de routage peuvent

aussi être étudiés dans un contexte incertain, c'est à dire lorsque la matrice de trafic qu'on cherche à écouler dans le réseau n'est pas définie d'une manière précise. On peut par exemple supposer que la matrice de trafic varie dans un polytope et chercher un routage compatible avec toutes ces matrices [9], [107], [114].

Pour router du trafic sur une topologie déterminée, il faut installer des capacités de transmission. Ces capacités sont souvent choisies parmi un ensemble discret. L'optimisation du coût des capacités installées se ramène souvent à un problème de programmation linéaire en nombres entiers. Ces problèmes sont généralement appelés problèmes de dimensionnement. Ils sont en réalité souvent résolus en même temps que les problèmes de routage.

Un autre type de problèmes d'optimisation qui se posent dans le contexte des réseaux est le problème de tarification. En effet, un réseau offre des services qui doivent être facturés aux clients. Trouver le bon mode de tarification qui garantit par exemple une certaine équité ou un certain niveau de bénéfice à l'opérateur, se ramène à des problèmes d'optimisation et de théorie des jeux [16].

Les types de problèmes qu'on vient d'exposer peuvent être étudiés séparément ou d'une manière combinée. L'utilisation d'une technologie donnée (Internet, ATM, GSM, UMTS, SDH, etc.) induit des contraintes techniques particulières et donc des problèmes d'optimisation particuliers.

Les problèmes d'optimisation de réseaux que nous avons étudiés dans cette thèse sont des problèmes du type topologie.

1.2 Approche polyèdrale

Dans cette section on donne des notions générales sur la théorie des polyèdres, et pour plus de détails, le lecteur peut se référer à [117], [124].

1.2.1 Introduction

Soit E un ensemble fini tel que $|E| = n$. Soit $(w_e)_{e\in E}$ un vecteur de poids associé aux éléments de E. Soit $\mathbb{F} \subseteq 2^E$ une famille de sous ensembles de E. Si $F \in \mathbb{F}$ alors on note le poids de F par $w(F) = \sum_{e\in F} w(e)$. Le problème qu'on peut poser ainsi, est de chercher un ensemble noté par F^* dans $\mathbb{F}$ de telle sorte que $w(F^*)$ soit optimum. Ce problème est appelé un ***problème d'optimisation combinatoire***.

Pour illustrer notre propos, appuyons-nous sur l'exemple du problème de recherche d'un plus court chemin dans un graphe, où E est l'ensemble des arcs d'un graphe et w définit la fonction des poids sur les arcs : $w : E \to \mathbb{R}^+$, et $\mathbb{F}$ sera l'ensemble de tous les chemins entre l'origine et la destination qu'on souhaite.
D'autres exemples d'optimisation combinatoire peuvent être consultés dans [17], [20].

1.2.2 Les ensembles et combinaisons convexes

Étant donné un graphe $G = (V,E)$, et soit $F \subseteq E$ un sous ensemble d'arêtes. On note par x^E le vecteur en $0-1$ dit d'*incidence* tel que $x^F \in \mathbb{R}^{\mathbb{E}}$, et $x^F(e) = 1$ si $e \in F$, sinon 0.
Un ensemble $\mathcal{C} \in \mathbb{R}^n$ est dit **convexe** s'il contient tous les segments en ligne entre leurs deux

extrémités. Autrement dit, $\lambda_1 x_1 + \lambda_2 x_2 \in \mathcal{C}$, pour tout $x_1, x_2 \in \mathcal{C}$, $\lambda_1, \lambda_2 \geq 0$ et que $\lambda_1 + \lambda_2 = 1$. La figure (1.1) représente quelques ensembles convexes et non convexes dans $\mathbb{R}^2$.

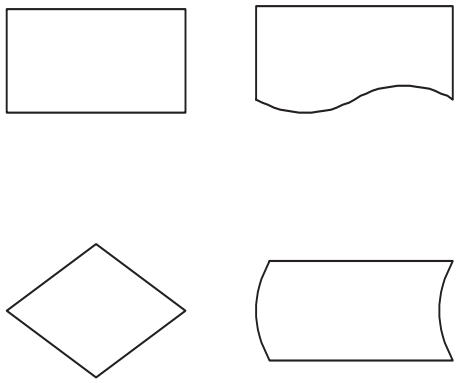

FIGURE 1.1 – Exemple d'ensembles convexes et d'ensembles non convexes

Une *combinaison linéaire* x des vecteurs $x_1, x_2, \ldots x_k \in \mathbb{R}^m$ est exprimée par $\sum_{i=1}^{i=k} \lambda_i x_i$. Si de plus on a $\sum_{i=1}^{i=k} \lambda_i = 1$ on parlera alors de combinaison *affine* des x_i. Si au même temps on a $\forall i = 1, \ldots, k : \lambda_i \geq 0$ alors on a une combinaison *convexe*. On dit alors que des vecteurs sont linéairement indépendants (respectivement. affinement indépendants) si on ne trouve pas de vecteur qui s'exprime en combinaison linéaire (combinaison affine) des autres vecteurs. Si S est un ensemble de vecteurs d'incidence dans $\mathbb{R}^m$, alors on note par $Conv(S)$ *l'enveloppe convexe* de S qui représente l'ensemble des combinaisons convexes des vecteurs de S.

1.2.3 Programmation Linéaire et Optimisation Combinatoire

Comme on l'a déjà défini ci-dessus, un problème d'optimisation combinatoire est de la forme suivante :

$$POC = \max\{w(F) = \sum_{e \in F} w(e), F \in \mathbb{F}\} \tag{1.1}$$

Afin de bien pouvoir poser les problèmes à traiter sous forme mathématique, on propose la notation suivante : pour $F \in \mathbb{F}$, on associe un vecteur $x^F \in \{0,1\}^{|E|}$, appelé ***vecteur d'incidence***, et dont les valeurs sont données par

$$x_i^F = \begin{cases} 1, & i \in F\,; \\ 0, & i \in E \setminus F. \end{cases}$$

En utilisant cette notation, on peut alors formuler un problème d'optimisation combinatoire, en un programme en 0-1. Considérons maintenant l'enveloppe convexe de tous les vecteurs d'incidence. Si on dispose d'une description polyèdrale de cette enveloppe convexe, alors maximiser la fonction linéaire wx sur cet ensemble est strictement équivalent à la maximiser sur l'ensemble des vecteurs d'incidence. On sait, en effet, que tout programme linéaire atteint son optimum en au moins un point extrême.

La difficulté de cette approche réside dans la caractérisation polyèdrale de l'enveloppe convexe. Notons qu'en pratique, une description partielle pourrait suffir lorsqu'on applique une méthode du type Branch&Cut qu'on décrira dans la suite.

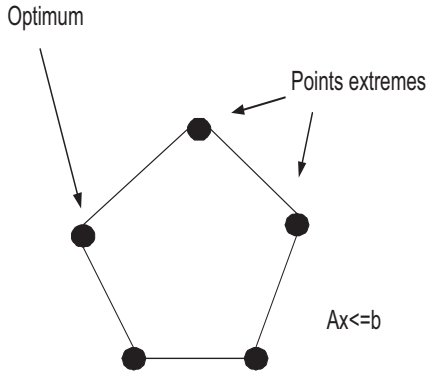

FIGURE 1.2 – Représentation d'un polyèdre

1.2.4 Polyèdres, Faces et Facettes

Étant donnés une matrice $A_{m*n} \in \mathbb{R}^{m*n}$ et un vecteur $b \in \mathbb{R}^m$. On définit un **polyèdre** P par $\{x \in \mathbb{R}^m; Ax \leq b\}$. P représente alors l'intersection d'un nombre fini de demi-espaces dans $\mathbb{R}^m$. On dit que le polyèdre P est un **polytope** s'il est borné. La dimension d'un polyèdre P notée par $dim(P)$ est égale au nombre maximum de vecteurs affinement indépendants moins un. En fonction du rang de la matrice A, on peut d'abord distinguer la notation $A^=$ qui représente une sous matrice de A des inégalités satisfaites à l'égalité par tous les vecteurs de P, alors on peut redéfinir la dimension de P, par $dim(P) = m - rang(A^=)$. Si $dim(P) = n$ alors on dit que le polyèdre est de *pleine dimension*.

On dit qu'une inégalité $ax \leq \alpha$ est *valide* si elle est satisfaite par tous les vecteurs de P. Ainsi, l'ensemble des inégalités valides de P satisfaites à l'égalité est défini par une **facette** F, et tel que $F = \{x \in P; ax = \alpha\}$. La dimension d'une facette F est $dim(F) = dim(P) - 1$.

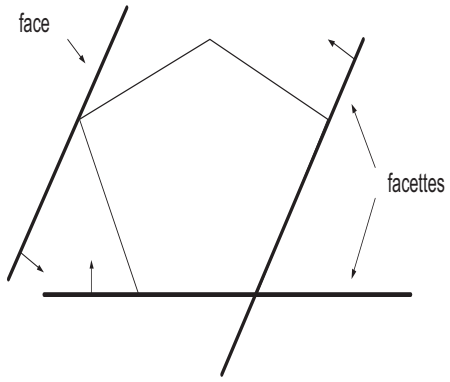

FIGURE 1.3 – Polyèdre , face et facettes

Il est difficile de décrire l'ensemble des facettes de l'enveloppe convexe des solutions réalisables d'un problème combinatoire NP-Difficile. Ceci est une conséquence de l'équivalence

entre la séparation et l'optimisation. Cependant, même si le problème combinatoire est facile à résoudre, il peut être difficile d'énumérer toutes les facettes. En effet, dans certains cas, elles sont en nombre exponentiel. Dans d'autres cas, on ne connaît même pas la forme de ces facettes.

1.2.5 Méthode de Coupes

Pour un problème d'optimisation combinatoire donné, il est en général difficile de caractériser le polyèdre associé par un système d'inégalités linéaires. De plus, même si ce dernier est caractérisé, le système décrivant le polyèdre peut contenir un nombre exponentiel d'inégalités et donc reste inexploitable pour être résolu comme un programme linéaire. Cependant, une description partielle du polyèdre garantie par la ***méthode de coupes*** peut être suffisante pour résoudre le problème jusqu'à l'optimum.

Soit le problème d'optimisation combinatoire suivant

$$\mathbb{P} = \max\{wx : Ax \leq b, \;\; x \;\; entier\} \tag{1.2}$$

et soit P l'enveloppe convexe des solutions de (1.2). On rappelle que ce problème est équivalent au programme $\max\{wx : x \in P\}$, et que si les inégalités du système $Ax \leq b$ sont suffisantes pour décrire complètement P, alors tous les points extrêmes du polyèdre $\{x \in \mathbb{R}^n : Ax \leq b\}$ sont des entiers, et par conséquent le problème (1.2) est équivalent à la relaxation $\max\{wx : Ax \leq b\}$. Néanmoins, ce n'est pas toujours le cas, et le polyèdre $\{x \in \mathbb{R}^n : Ax \leq b\}$ peut bien avoir des points extrêmes fractionnaires. En d'autres mots, il existe une contrainte valide violée par cette solution optimale fractionnaire. Cette dernière peut être ajoutée au système $Ax \leq b$, afin de trouver une nouvelle relaxation linéaire plus forte.

Dans la méthode de coupes, l'étape de la génération des contraintes valides est la plus difficile. L'une des premières techniques dont la finalité est d'identifier les contraintes valides pour des programmes en variables entières, a été introduite par Gomory. Ensuite, Chvátal a généralisé cette technique, pour obtenir la méthode bien connue de Chvátal-Gomory.

1.2.6 Méthode de Branch&Cut

La méthode qu'on décrit ici, permet de construire un arbre de résolution (arbre de Branch&Cut), où chaque sommet de l'arbre correspond à un sous-problème, et le problème initial est associé à la racine de l'arbre.

Le principe de cette méthode commence par résoudre initialement la relaxation du problème (1.2). Si la solution trouvée est optimale et contient une variable fractionnaire, qui est sensée être entière, alors un algorithme à plans coupants est appliqué afin de trouver de nouvelles inégalités valides (quelques unes peuvent suffire). Ces dernières sont satisfaites par tous les points entiers, mais violées par la solution courante. Ces inégalités seront ajoutées au programme courant.
Par la suite, on utilise une méthode de branchement qui consiste à obtenir de nouveaux noeuds. On résout chaque programme associé à un noeud en y ajoutant un certain ensemble de coupes

qui peuvent être valides localement au niveau du noeud courant ainsi que du sous-arbre issu de lui, ou globalement au niveau de tous les noeuds de l'arbre de Branch&Cut.
Lors de la résolution d'un programme associé à un noeud de l'arbre, on calcule une évaluation. Si cette évaluation est supérieure (pour un problème de minimisation) à la borne supérieure calculée avant, alors on coupe ce noeud. Dans le cas contraire, il est exploité à son tour.

On arrête cet algorithme lorsqu'on ne peut plus exploiter un noeud de l'arbre. On prend alors la meilleure solution trouvée.

Remarque : Les coupes qu'on ajoute aux programmes associés aux noeuds de l'arbre de Branch&Cut peuvent être générées jusqu'à épuisement.

1.2.7 Séparation et Optimisation

Considérons un problème d'optimisation combinatoire du type

$$\mathbb{P} = \max\{wx : Ax \leq b,\ \ x\ \ entier\} \tag{1.3}$$

Le problème de séparation qui lui est associé est défini comme suit :

Pour une solution donnée x^*, vérifier si x^* appartient à l'enveloppe convexe des solutions entières du système $Ax \leq b$, et sinon déterminer une inégalité valide pour cette enveloppe convexe qui soit violée par x^*.

Ce problème est appelé ***problème de séparation*** associé à l'enveloppe convexe des solutions entières du système $Ax \leq b$.
Si x^* ne vérifie pas le système $Ax \leq b$ alors il existe un hyperplan séparant x^* et le polyèdre $Ax \leq b$.
Un théorème fondamental prouvé dans [60] stipule que la séparation et l'optimisation sont équivalentes au niveau de la complexité. En d'autres termes, si on dispose d'un algorithme de séparation, alors on peut optimiser en faisant appel à cet algorithme un nombre polynomial de fois. Ici la notion de polynomialité s'entend en fonction de la taille maximale de codage d'une facette (ou d'un sommet) de l'enveloppe convexe.

1.3 Problèmes combinatoires importants

1.3.1 Les problèmes de plus court chemin

Les problèmes du plus court chemin dans un graphe sont sans doute parmi les problèmes les plus importants dans l'optimisation combinatoire. Ces problèmes ont plusieurs champs d'application. Selon différents cas, ils peuvent être difficiles à résoudre si les poids des arêtes sont arbitraires. Par exemple, si les poids des arêtes d'un graphe $G = (V,E)$ sont tous égaux à -1, alors la recherche des $s-t$ chemins de poids $1-|V(G)|$ est simplement la recherche des $s-t$ chemins Hamiltoniens, dont le problème de décision est difficile au sens de la théorie de la complexité.
On distingue plusieurs cas de recherche de plus court chemin :

– Étant donnés deux sommets u et v, on s'intéresse à la recherche d'un plus court chemin de u à v.
– Étant donné un sommet u, on s'intéresse à la recherche d'un plus court chemin de u vers tous les autres sommets.
– Pour tout couple de sommets (u,v), on cherche le plus court chemin entre u et v.

Pour tout cas de ces problèmes, il existe différents algorithmes de résolution selon les pondérations des arêtes.
Soit $G = (V,E)$ un graphe orienté, d'ordre n et de taille m, et soit $w : E \to \mathbb{R}$ une application représentant les poids (coûts) sur les arêtes de G. Sans perte de généralités, on suppose ici que le graphe G est simple et connexe.

1.3.1.1 L'algorithme de Dijkstra

Soient u et v deux sommets quelconques de V, et on s'intéresse dans cette section à la recherche d'un plus court chemin entre u et v. Au départ, on va supposer que ce chemin existe, et que les pondérations sur les arêtes sont toutes positives : $w : E \to \mathbb{R}^+$, afin d'éviter les circuits absorbants.

La résolution du problème de plus court chemin par l'algorithme de Dijkstra est alors détaillée ci-dessous :
Á chaque étape, on sélectionne un sommet t et on fixe la valeur du plus court chemin de u à t.
On dispose alors de deux ensembles de sommets :

1. *Sommets examinés* : sera l'ensemble des sommets pour lesquels on connaît la valeur du plus court chemin à partir de u.
2. *Sommets à-traiter* : sera l'ensemble des sommets qui ont au moins un prédécesseur dans l'ensemble des sommets examinés.

Le critère de sélection d'un sommet dans l'ensemble *à-traiter* est de choisir le sommet t qui minimise le coût du chemin de u à t.
La sélection d'un sommet dans l'ensemble *à-traiter* pénalise à chaque fois l'algorithme de Dijkstra au sens de la complexité. En effet, cette dernière est de $O(n^2)$.

1.3.1.2 L'algorithme de Bellman

On a vu précédemment que l'algorithme de Dijkstra fonctionne sur des graphes avec circuits mais de coûts positifs. Dans ce qui suit, on considère des graphes sans circuits, mais de coûts réels et de signes quelconques. On note par $\mu(u,t)$ le plus court chemin de u à t.
L'algorithme de Bellman calcule le plus court chemin d'un sommet u vers tous les autres sommets t, et ses détails sont donnés ci-dessous :
– On numérote les sommets de 1 à n : tri topologique (tous les antécédents du sommet j auront des numéros inférieurs à j).
– Pour tout sommet y prédécesseur de t, on calcule $\mu(u,t) = \min\{\mu(u,y) + w(y,t)\}$

L'algorithme de Bellman est aussi polynomial et de complexité de $O(n+m)$.

1.3.1.3 L'algorithme de Bellman-Ford

Dans cette partie, on s'intéresse à la recherche d'un plus court chemin d'un sommet aux autres sommets, sur un graphe dont les pondérations peuvent être quelconques, et l'existence de circuits dans ce graphe n'est pas exclue.
On présente alors l'algorithme de Bellman-Ford, qu'on décrit ci-dessous :

C'est un algorithme qui fonctionne sur plusieurs étapes. Á l'étape k on cherche de u vers tout sommet t un plus court chemin dans G ayant au plus k arcs, et la longueur d'un tel chemin. Dans le graphe G (d'ordre n) les chemins élémentaires ont tous $n-1$ arcs, et l'ensemble des longueurs des plus courts chemins trouvés à l'étape $n-1$ doit être identique à celui de l'étape n, à l'exception du cas où le graphe contient un circuit absorbant. Pour savoir si on est dans un tel cas, on utilise une variable booléene qui nous informe si le passage de l'étape $k-1$ à l'étape k a fait varier la longueur des chemins. Si k vaut n et que aucune distance n'a varié par rapport à l'étape précédente, alors le graphe ne contient pas de circuit absorbant, et les distances trouvées donnent la longueur des plus courts chemins de u vers les autres sommets. Dans le cas contraire dans lequel on constate un changement et que k est égal à n, alors il existe un circuit absorbant accessible à partir de u.
Cet algorithme est de complexité de $O(nm)$.

1.3.2 Problème de l'arbre couvrant de poids minimum

Dans cette partie, on suppose qu'on a un graphe $G = (V,E)$ non orienté et connexe. Les arêtes de G sont pondérées (coûts, poids,...).
Le problème de l'arbre couvrant de poids minimum du graphe G consiste à chercher un graphe partiel de G qui soit un arbre et de coût minimum.

Pour résoudre ce problème, on propose alors deux algorithmes de complexités différentes : l'algorithme de Kruskal, et l'algorithme de Prim.

1.3.2.1 Algorithme de Kruskal

Le principe de fonctionnement de cet algorithme est simple. En effet, il suffit de prendre les arêtes de poids minimum une à une, et vérifier à chaque fois que l'arête en cours ne constitue pas un cycle avec les arêtes prises avant.
On peut alors le détailler ainsi :

1. Trier les arêtes par ordre des poids croissants.
2. Procéder dans l'ordre du tri ainsi : prendre la première arête non examinée, et vérifier si elle forme un cycle avec les arêtes précédemment choisies, si oui la rejeter, sinon la garder, et continuer cette deuxième étape jusqu'à retenir $n-1$ arêtes.

La complexité de l'algorithme de Kruskal est de l'ordre de $O(n^2 + mlog_2 m)$.

1.3.2.2 Algorithme de Prim

Dans cette partie, on va donner un deuxième algorithme pour la recherche d'un arbre couvrant de coût minimum, sans passer par le tri des poids des arêtes. En effet l'algorithme de Prim fonctionne ainsi : on pose T l'arbre en cours de construction. À chaque étape on ajoute à T un

sommet et une arête. À une étape donnée, on pose R l'ensemble des sommets qui ne sont pas dans T. À chaque sommet u de R on associe $proche(u)$ qui est le sommet de T tel que le poids de l'arête $(u, proche(u))$ soit minimum sur les autres sommets de T. On pose $d(u)$ (distance) le poids de cette arête. On choisit ainsi de faire entrer dans T le sommet u de distance minimum. Ce sommet u va servir comme *pivot* pour l'étape suivante. Ensuite, on met à jour les attributs des sommets r qui restent dans R et qui sont voisins du *pivot*, en comparant l'ancienne distance $d(r)$ à $w(pivot, r)$. Dans le cas où $d(r) > w(pivot, r)$ alors on met à jours $proche(r) = pivot$ et $d(r) = w(pivot, r)$.
La complexité de l'algorithme de Prim est de l'ordre de $O(n^2)$.

1.3.3 Le problème de flot maximum-coupe minimum

Dans cette sous section on considère un graphe $G = (V, E)$ orienté, et soient s et t deux sommets fixés de G. Soit $c : E \to \mathbb{R}^+$ une fonction dite de *capacité* sur les arcs du graphe. On distingue deux sommets s et t qui seront appelés ***source*** et ***puits***.

Définition 1.1: Soit $R = (V, E, c)$ un réseau, s et t les sommets source et puits. Un ***flot*** f est une application de $V \times V$ dans $\mathbb{R}$ vérifiant :

1. $f(x,y) = -f(y,x), \forall (x,y) \in E$.
2. $f(e) \leq c(e), \forall e \in E$.
3. $f(v,V) = 0, \forall v \in V \setminus \{s,t\}$.

D'une manière concrète, on dispose d'un graphe orienté dont les arêtes sont étiquetées par des nombres. On cherche à étiqueter par des nombres inférieurs ou égaux à C en assurant la nullité de leur somme orientée en tout point n'étant pas la source ou le puits.

La valeur du flot f est alors $|f| = f(s,V)$. On va s'intéresser à la recherche d'un flot f maximisant $|f|$.

1.3.3.1 La méthode de Ford-Fulckerson

Pour décrire cette méthode, on va utiliser le graphe des écarts, les chemins améliorants et les coupes, qu'on définit d'abord.

Soit $R = (V, E, c)$ un réseau, s et t sont la source et le puits respectivement, f un flot sur le réseau. La capacité résiduelle d'un arc u est notée par $c_f = c(u) - f(u)$.

Définition 1.2: (***Le réseau des écarts***). On appelle par ***le réseau des écarts*** le graphe $R_f = (V, E_f, c_f)$ où $E_f = \{(x,y) \in V \times V, c_f(xy) > 0\}$.
En d'autres mots, pour un arc donné u, si $c_f(u) = 0$, alors on remplace u par un arc étiqueté par $c_f(u)$, sinon on le supprime.

Définition 1.3: (***Chemin améliorant***). Un ***chemin améliorant*** noté μ, est un chemin de s à t dans R_f.

Définition 1.4: (***Capacité résiduelle***). La ***capacité résiduelle*** de μ est notée par $c_f(\mu) = min\{c_f u | u \in \mu\}$.

Définition 1.5: (***Coupe***). Une ***coupe*** dans $R = (V, E, c)$ est une partition de V en $(S, \overline{S})$ avec $s \in S$ et $t \in \overline{S}$. La capacité de cette coupe est $c(S, \overline{S}) = \sum_{i \in S, j \in \overline{S}} c(i,j)$.
Le théorème suivant établit le lien entre la coupe et le flot dans un réseau.

Théorème 1.1: ***(Ford et Fulckerson)***. Soit f un flot dans un réseau $R = (V,E,c)$. Il y a équivalence entre les trois propositions suivantes :

1. f est un flot maximum.
2. Il n'existe pas de chemin améliorant.
3. Il existe une coupe $(S,\overline{S})$ telle que $|f| = c(S,\overline{S})$.

La méthode de Ford-Fulckerson consiste à démarrer d'un flot nul, et à l'améliorer en lui ajoutant f_μ tant qu'il existe un chemin améliorant μ. Elle est d'une complexité algorithmique de $O(f * |E|)$

Il existe un autre algorithme dit Algorithme d'Edmonds-Karp, qui consiste à choisir dans R_f à chaque étape de la méthode de Ford-Fulckerson, un plus court chemin en nombre d'arcs. Cet algorithme se base sur un graphe non orienté, et ne calcule pas les graphes des écarts. En d'autres mots, il ne parcourt que les les arcs de capacité résiduelle strictement positive. Il est d'une complexité de $O(nm^2)$.

1.3.3.2 La Méthode des préflots- Algorithme de Goldberg-Tarjan

Définition 1.6: (***Préflot***). Soit $R = (V,E,c)$ un réseau de source s et de puits t. Un ***préflot*** est une application $f : E \to \mathbb{R}$, et qui vérifie les trois propriétés suivantes :

- $f(x,y) = -f(y,x)$;
- $f(u) \leq c(u), \forall u \in E$;
- $f(V,v) \geq 0, \forall v \in V \backslash \{s\}$

Le flot en ***excès*** en un sommet v est défini par $e(v) = f(V,v)$. Si $e(v) > 0$, alors v est dit un sommet ***excédentaire***.

Définition 1.7: (***Fonction de hauteur***). Soit le réseau $R = (V,E,c)$ de source s et de puits t. Soit f un préflot.
Une fonction $h : V \to \mathbb{N}$ est dite une ***Fonction de hauteur*** si :

- $h(s) = |V|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1, \forall (u,v) \in E_f$

L'algorithme de Goldberg-Tarjan se base sur deux opérations importantes :

1. Opération "*pousser*".
2. Opération "*élever*".

L'opération ***pousser*** qui s'applique sur un arc (u,v) peut être appliquée si :

- $e(u) > 0$ (u est excédentaire)
- $c_f(u,v) > 0$ ((u,v) n'est pas saturé par le flot)
- $h(u) = h(v) + 1$ (on pousse le flot de haut en bas)

L'algorithme 1 résume la fonction $pousser(u,v)$:

On dit que la poussée est ***saturante*** si après cette opération, l'arc (u,v) est saturé, donc $c_f(u,v) = 0$.

La deuxième opération dite ***élever***, s'applique si :

- $e(u) > 0$ (u est excédentaire)
- $\forall (u,v) \in E_f, h(u) \leq h(v)$
- $u \neq t$

```
1  Fonction : pousser(u,v);
2  d_f = min(e(u), c_f((u,v)));
3  f(u,v) = f(u,v) + d_f;
4  f(v,u) = −f(u,v);
5  e(u) = e(u) − d_f;
6  e(v) = e(v) + d_f;
```

Algorithm 1: Algorithme de l'opération **pousser**

```
1  Fonction : elever(u,v);
2  h(u) = 1 + min{h(v)|(u,v) ∈ E_f};
```

Algorithm 2: Algorithme de l'opération **élever**

L'algorithme 2 résume la fonction Élever.

Ensuite, on définit la fonction *init-préflot* (algorithme 3) qui va saturer tous les arcs issus de la source s.

```
1  Fonction : init − preflot(R,s,t);
2  for u ∈ V do
3      h(u) = 0; e(u) = 0;
4  end
5  for (u,v) ∈ E do
6      f(u,v) = 0; f(v,u) = 0;
7  end
8  h(s) = |V|;
9  for v ∈ Successeur(s) do
10     f(u,v) = c(s,v); f(v,s) = −c(s,v); e(v) = f(s,v);
11 end
```

Algorithm 3: Algorithme d'initialisation du préflot

L'algorithme 4 est le corps principal de l'algorithme de préflot, dit aussi l'algorithme de ***Push/Relabel***. On peut trouver différentes façons pour l'implementer, et chaque façon nous donne une complexité différente de l'algorithme :

- FIFO : qui donne une complexité de $O(n^3)$.
- Plus haut degré : une complexité de $O(n^2\sqrt{m})$.

1.3.3.3 L'algorithme de Gomory-Hu

L'algorithme de Gomory-Hu est aussi une autre méthode de calcul de flot maximum-coupe minimum qui opère sur un graphe en le transformant en un arbre qui va représenter les flots maximums entre chaque deux sommets du graphe G.

Cet algorithme est d'une nature récursive. Il consiste à diviser l'ensemble des sommets du graphe G en deux sous ensembles R et S, et faire pareillement par la suite sur ces deux derniers ensembles. Initialement, il est appliqué sur tous les sommets du graphe.

Au départ, on sélectionne d'une manière aléatoire deux sommets du graphe, et on applique

1 $init - preflot$ **while** $\exists$ une opération pousser ou élever **do**
2 sélectionner une opération valide et l'appliquer

Algorithm 4: Algorithme de préflot

par exemple l'algorithme de Ford-Fulckerson pour calculer un flot maximum-coupe minimum entre ces deux sommets. Après ce calcul, on obtient deux sous-ensembles de sommets (R et S). Chacun des deux sommets sélectionnés au départ va appartenir à un des deux ensembles (R ou S). On continue encore cette procédure en choisissant à chaque fois deux sommets des deux dernières parties formées, et on calcule de nouveau un flot maximum entre ces deux sommets. Ainsi on aura calculé $n-1$ flots maximums dans ce graphe, et le résultat est l'arbre de Gomory-Hu, qui nous donne le flot maximum (coupe minimum) entre chaque couple de sommets de G. La complexité de cet algorithme est de l'ordre de $O(n^2\sqrt{m})$.

1.3.4 Les T-join et T-coupe

Étant donné un graphe non orienté $G = (V,E)$, et un sous-ensemble de sommets T de V, tel que T est de cardinalité paire. Un sous ensemble d'arêtes F est dit un **T-Join** s'il coincide avec l'ensemble des sommets de degrés impairs dans le graphe $G' = (V,F)$. Une **T-coupe** est un ensemble d'arêtes de la forme de $\delta(W)$ où W est un ensemble de sommets tels que la cardinalité de $|T \cap W|$ est impaire.

1.3.4.1 Le problème de T-coupe de poids minimum

Il s'agit de chercher dans le graphe G la T-coupe de poids minimum. Grâce à l'algorithme de Gomory-Hu décrit ci-dessus, on construit un arbre de coupes, et on choisit parmi elles, celles qui représentent des T-coupes, ensuite on choisit la moins coûteuse [80].

1.3.4.2 Le problème du T-join de poids minimum

Sur les arêtes du graphe G, on définit la fonction des coûts $w : E \to \mathbb{R}$. Soit T un sous ensemble de V de cardinalité paire.

On s'intéresse dans cette partie à la recherche d'un T-join de poids minimum. Ce problème peut être la généralisation de plusieurs problèmes d'optimisation combinatoire. On cite par exemple :

- Si les poids des arêtes du graphe G sont positifs, et que T est l'ensemble de sommets de degrés impairs, alors on obtient le problème du postier chinois non orienté.
- Si $T = V$, alors le T-Join de cardinalité $\frac{|V|}{2}$ est en effet un couplage parfait du graphe G, et donc le problème du T-join de poids minimum peut se réduire au problème du couplage parfait de poids minimum.

Edmonds et Johnson [80] ont montré que, dans le cas où les poids des arêtes du graphe G sont positifs, le problème du T-join de poids minimum peut être résolu en $O(n^3)$.

1.3.5 Le problème de couplage

Dans cette section, on s'intéresse à l'existence d'un ensemble d'arêtes deux à deux disjointes recouvrant tous les sommets d'un graphe.
Un ***couplage*** de $G=(V,E)$ est un ensemble d'arêtes $M \subset E$ tel que deux arêtes de M n'ont pas d'extrémités communes. En d'autres mots, dans le graphe partiel $G'=(V,E')$, on a $d_{G'}(v) \leq 1$, $\forall v \in V$.

Les sommets v pour lesquels on a $d_{G'}(v)=1$ sont dits ***M-saturés***. Un couplage M est dit ***Maximal*** si $\forall e \in E \backslash M$, $M \cup \{e\}$ n'est pas un couplage. Un couplage M est ***Maximum*** si pour tout couplage M', on a $|M'| \leq |M|$. On dit finalement qu'un couplage M est ***Parfait*** s'il sature tous les sommets du graphe G.
Une chaîne est ***M-alternée*** si ses arêtes sont alternativement dans M et dans $E \backslash M$.

Le problème de couplage est l'un des problèmes de base. Il consiste à trouver un couplage de cardinalité maximum, ce qui signifie qu'on cherche à couvrir un nombre maximum de sommets. Chaque couple de sommets seront reliés par une et une seule arête, et deux arêtes du couplage n'auront jamais la même extrémité. Tutte et Berge ont montré que pour un graphe non orienté, la cardinalité d'un couplage maximum est égale à $\min_{V' \subseteq V} \frac{|V|+|V'|-odd(V \backslash V')}{2}$ où $odd(V \setminus V')$ représente le nombre de composantes dans le sous graphe $G-V'$ qui ont un nombre impair de sommets (voir [60]). Edmonds a donné une caractérisation polyèdrale du problème de couplage :

$$\begin{aligned} x(e) &\geq 0, \quad \forall e \in E \\ x(\delta(v)) &\leq 1, \quad \forall v \in V \\ x(E(W)) &\leq \lfloor \tfrac{|W|}{2} \rfloor \quad \forall W \subseteq V, |W| impair \end{aligned}$$

Si on suppose que les arêtes du graphe G sont pondérées, on peut alors chercher un couplage maximum de poids minimum. Et si de plus on veut que notre couplage soit parfait, et que les arêtes sont de poids minimum, on s'intéressera à un couplage parfait de poids minimum.

Dans le cas ou le graphe G est un graphe biparti, la recherche d'un couplage maximum peut se faire avec la technique du chemin augmentant. En effet, on suppose qu'on a un couplage de départ M, et que les deux parties des sommets de G sont U et W, et on oriente toute arête (u,w) avec $u \in U$ et $v \in W$, ainsi :

- si $e \in M$, alors on oriente e de w vers u.
- sinon, on l'oriente de u vers w.

Ainsi on obtient un graphe orienté noté D dont les sommets sont $U' = U \setminus \cup M$ et $W' = W \setminus \cup M$. On cherche par la suite dans D un chemin augmentant (s'il existe) à partir d'un sommet quelconque de U' vers un autre sommet de W'. De cette façon, on peut trouver un couplage plus large que M.

1.3.5.1 Le problème de b-couplage

Le problème de **b-couplage** est considéré comme une généralisation du problème de couplage. En effet, et étant donné $b : V \to \mathbb{Z}^+$, le b-couplage est alors une fonction $x : E \to \mathbb{Z}^+$ et tel que $x(\delta(v)) \leq b(v)$, $\forall v \in V$. Un b-couplage parfait consiste alors à remplacer cette inégalité par $x(\delta(v)) = b(v)$, $\forall v \in V$. Le polytope des b-couplages (avec des capacités de 1) est défini par [80] :

$$0 \leq x(e) \leq 1, \quad \forall e \in E$$
$$x(\delta(v)) \leq b(v), \quad \forall v \in V$$

$$x(E(W)) + x(F) \leq \lfloor \frac{b(W) + |F|}{2} \rfloor \quad \forall W \subseteq V, F \subseteq \delta(W), (b(W) + |F|) \ \textit{impair} \tag{1.4}$$

Les contraintes (1.4) (dites de blossom) sont importantes. En effet, on les trouve dans les enveloppes convexes de plusieurs problèmes d'optimisation NP-Difficiles comme le problème du voyageur de commerce (TSP). À cause de ce fait, on va s'intéresser à leur séparation.

Algorithme de séparation de Padberg&Rao

L'algorithme de Padberg&Rao consiste à séparer en temps polynomial les inégalités (1.4). Les étapes de cet algorithme sont les suivantes :

1. Construire un nouveau graphe $\hat{G}$ avec de nouveaux poids $\hat{x}$ sur les arêtes.
2. Affecter un ordre de parité (pair/impair) aux sommets de $\hat{G}$, et soit T l'ensemble des sommets impairs.
3. Calculer une coupe minimum impaire dans $\hat{G}$:
   - Calculer un arbre de coupes dans le graphe $\hat{G}$.
   - Vérifier sur toutes les $|T|-1$ arêtes de l'arbre de coupes, si son poids < 1, et que la coupe qui l'induit est impaire.

Pour mieux comprendre le fonctionnement de cet algorithme, on détaille ces étapes :

– *Construire un nouveau graphe $\hat{G}$ avec de nouveaux poids $\hat{x}$ sur les arêtes :*

Pour la construction du nouveau graphe $\hat{G}$, on prend chaque arête $e = (i, j)$ du graphe initial, et on la divise en deux, on y ajoutant un nouveau sommet k_e au milieu. On obtient deux arêtes dont les poids sont : $\hat{x}_{i,k_e} = x_e$ et $\hat{x}_{k_e,j} = 1 - x_e$. Le nouveau graphe $\hat{G} = (\hat{V}, \hat{E})$ aura $\hat{V} = |V| + |E|$, et $\hat{E} = 2|E|$.

– *Affecter un ordre de parité (pair/impair) aux sommets de $\hat{G}$, et soit T l'ensemble des sommets impairs :*

Soit $u \in \hat{V}$, alors u est impair si et seulement si :

- $u = k_e \in V$, ou bien
- $u \in V$, et que le nombre d'arêtes complémentaires incidentes à u est impair

dans le cas contraire u sera pair.

– *Calculer une coupe minimum impaire dans $\hat{G}$:*

Si la valeur d'une coupe minimum impaire est inférieure à 1 alors on a une violation des contraintes (1.4). Soit $T = \{sommets \quad impairs \quad \in \hat{V}\}$. Une coupe $\delta(U)$ est dite impaire si elle vérifie que la cardinalité $|U \cap T|$ est impaire. La recherche d'un arbre de coupes peut être effectuée grâce à l'algorithme de Gomory-Hu qui utilisera $|T|-1$ flots maximums (coupes minimums) qu'on peut calculer avec l'algorithme de Goldberg-Tarjan.
L'algorithme de Padberg&Rao est alors d'une complexité de l'ordre de $O(|E|^3 log(|V|))$.

1.3.6 Le problème du voyageur de commerce

Étant donné un graphe $G = (V, E)$ dont les arêtes sont pondérées par des poids $w : E \to \mathbb{R}$. Un circuit (cycle) Hamiltonien est un circuit (cycle) passant par tous les sommets de G une

et une seule fois. Le coût de ce circuit est la somme des coûts de ses arêtes. Le problème du voyageur de commerce (TSP pour Travelling Salesman Problem) consiste alors à trouver un circuit (cycle) Hamiltonien de G de coût minimum. C'est un problème typique d'optimisation combinatoire, avec quelques applications réelles, comme par exemple le problème de tournées de véhicules, ...
Le problème du voyageur de commerce est connu comme étant NP-Difficile [45].
Le modèle suivant présente l'une de ses formulations en programmation linéaire. À toute arête e on associe une variable x_e qui peut prendre deux valeurs : 1 si l'arête e fait partie de la solution, et 0 sinon.

$$\begin{gathered}\min \sum_{e\in E} w_e x_e \\ \sum_{i=1}^{n} x_{ij} = 1, \forall j \in V \\ \sum_{j=1}^{n} x_{ij} = 1, \forall i \in V \\ \sum_{i,j\in S} x_{ij} \leq |S| - 1, \forall S \subseteq V, 2 \leq |S| \leq n-1 \\ x_e \in \{0,1\}, \forall e \in E.\end{gathered} \tag{1.5}$$

La fonction objectif consiste à minimiser le coût de la tournée. Les deux premières contraintes consistent à assurer que le voyageur arrive et repart d'une ville, puis les troisièmes contraintes sont dites contraintes des sous tours, et elles consistent à éviter de revenir au point de départ avant la réalisation de la tournée Hamiltonienne.
Le nombre d'inégalités d'élimination des sous tours est proche de 2^n, ceci rend la résolution du modèle impossible, mais avec la relaxation linéaire de cette dernière formulation, on peut arriver à résoudre plusieurs cas.
Parmi les méthodes utilisées pour résoudre ce problème, on cite les méthodes exactes. Ces dernières sont beaucoup utilisées, à cause de leur efficacité. En effet, il s'agit de construire un algorithme de Branch&Cut dans lequel on propose de nouvelles inégalités valides pour le problème du TSP, comme les inégalités dites de *blossom* proposées par Edmonds, ou alors d'autres inégalités valides dites de peigne (*comb*) utilisées dans une méthode de Branch&Cut.

D'une autre part, les heuristiques aussi ont fait leurs preuves. L'heuristique du plus proche voisin est l'une des premières et plus simples proposées pour la résolution du problème de TSP. La méthode construit un cycle Hamiltonien en ajoutant un par un les noeuds à la fin des tours partiels. Cette heuristique trouve une solution en $O(n^2)$.

L'heuristique de Christofides est très efficace pour ce problème. En effet, elle garantit un rapport d'approximation de $\frac{3}{2}$ dans le cas du TSP Euclédien. Elle consiste à construire un arbre de poids minimum sur le graphe G, puis ajoute aux arêtes de cet arbre, celles qui réalisent un couplage de poids minimum entre les noeuds de degré impair. L'ensemble des arêtes forme ainsi un graphe connexe dans lequel chaque noeud a un degré pair. De ce dernier graphe, on peut extraire un cycle Eulérien puis éliminer les visites multiples à un noeud pour obtenir un cycle Hamiltonien.

1.4 Réseaux à composantes unicycliques

1.4.1 Les matroïdes

On dit que $M = (E, \mathfrak{F})$ est un matroïde, si et seulement si on a les trois propriétés suivantes :

1. $\emptyset \in \mathfrak{F}$.
2. Si $A \in \mathfrak{F}$ et que $B \subseteq A$, alors $B \in \mathfrak{F}$.
3. Si $A, B \in \mathfrak{F}$, et que $\mid B \mid > \mid A \mid$ alors $\exists e \in B \setminus A$, tel que $A \cup \{e\} \in \mathfrak{F}$.

Si $\mathfrak{F}$ ne satisfait que les deux premières propriétés précédentes, alors on parle d'un système d'indépendance. Une *base* de E est un ensemble maximal dans E, et toutes les bases d'un matroïde ont la même cardinalité. Une fonction *rang* notée par ρ est définie par :$\rho : 2^E \rightarrow \mathbb{Z}$ telle que

$$\rho(A) = \max\{|X|, X \subseteq A; etX \in \mathfrak{F}\}, A \subseteq E$$

On appelle un matroïde *dual* le matroïde $M^d = (E, \mathfrak{F}^d)$ défini par

$$\mathfrak{F}^d = \{J \subseteq E : E - J \quad contient \quad une \quad M - base\}$$

La fonction rang du matroïde dual est donnée par

$$\rho^d(A) = |A| + \rho(E - A) - \rho(E), A \subseteq E$$

Dans [85], [94], et [108], on peut distinguer plusieurs matroïdes. On en cite :

- **Le matroïde matriciel** : étant donnée une matrice réelle A_{m*n}. Soient
  $E = \{a_1, a_2, \ldots, a_n\}$ l'ensemble des colonnes de A et
  $\mathfrak{F} = \{J \subseteq E : \{a_j\}_{j \in J}$ est linéairement indépendant$\}$, alors on forme un matroïde matriciel.
- **Le matroïde graphique** : étant donné un graphe G. On pose E est l'ensemble d'arêtes de G, et si on pose : $\mathfrak{F} = \{J \subseteq E : G = (V, J)$ est une forêt $\}$ on obtient alors un matroïde graphique.
- **Le matroïde transversal** : étant donné un graphe $G = (V_1 \cup V_2, E)$, et on définit sur l'ensemble des sommets V_1 l'ensemble $\mathfrak{F} = \{J \subseteq V_1$: il existe un couplage dans G recouvrant $J\}$. Si V_2 est une collection des sous ensembles de V_1, et que les arêtes du graphe G traduisent l'appartenance des éléments à ces ensembles, alors un ensemble indépendant est dit transversal.

1.4.1.1 Les matroïdes bi-circulaires

Étant donné un graphe $G = (V, E)$, et si p est le nombre de composantes connexes de G, alors le nombre cyclomatique de G serait de $|E| - |V| + p$.

$M = (E, \mathfrak{F})$ est dit un **matroïde bi-circulaire** d'un graphe G, avec E qui sera l'ensemble des arêtes du graphe G, et les circuits de M seront les ensemble minimaux d'arêtes connexes dont le nombre cyclomatique correspondant est de 2. De plus, un ensemble d'arête est dit *indépendant* dans M si et seulement si le sous graphe induit par ce dernier contient au plus un cycle dans chaque composante.
Plus de détails sur le matroïde bi-circulaire se trouvent dans [94], [108].

Nous prouvons dans ce qui suit le fait que le matroïde bi-circulaire est bien un matroïde.

Proposition 1.1: Soit $G=(V,E)$ un graphe simple, d'ordre n. Alors $M=(E,\mathfrak{F})$ est un matroïde, tel que $\mathfrak{F}=\{I\subseteq E,\ I\ a\ ses\ composantes\ connexes\ qui\ contiennent\ au\ plus\ un\ cycle\}$.

Preuve:

- La première condition est évidente.
- Si on a $A\in\mathfrak{F}$, alors ceci veut dire que les composantes connexes de A contiennent au plus un cycle, et donc dire que $B\subseteq A$, ceci veut dire que B est soit un cycle, ou alors un arbre, mais dans tous les cas B reste une composante qui contient au plus un cycle, donc $B\in\mathfrak{F}$.
- Supposons que $A=\cup_{i=1}^{k}A_i$ qui sont les composantes connexes de A qui contiennent au plus un cycle. Alors, pour tout $i=1,\ldots,k$, on a $G_i=(V_i,A_i)$ tel que G_i est soit un arbre, ou une composante unicyclique. Ce qui nous amène à déduire que $n_A=\sum_{i=1}^{k}\mid V_i\mid=\mid A\mid+k'$, où $k'=1,\ldots,k$. On suppose aussi que $B=\cup_{j=1}^{t}B_j$, et donc $n_B=\sum_{j=1}^{t}\mid V_j\mid=\mid B\mid+t'$, où $t'=1,\ldots,t$. Sachant que $\mid B\mid>\mid A\mid$, on distingue alors deux cas :

  1. $t'>k'$: alors $\mid B\mid+t'>\mid A\mid+k'$ et donc $n_B>n_A$, ceci veut dire que B atteint plus de sommets que A, alors $\exists x$ atteint par B et non par A, et supposons que $e\in B$ qui contient x, donc $A\cup\{e\}\in\mathfrak{F}$.
  2. $n_B<n_A$: on suppose que les arêtes de B connectent chaque deux sommets de A de la même composante A_i, qui est un arbre ou une composante unicyclique. Raisonnons par l'absurde, et supposons qu'il n'y a pas de $e\in B\setminus A$, tel que $A\cup\{e\}\in\mathfrak{F}$, alors ceci veut dire que :
     - Soit $e\in B$, et alors e relie un sommet de la composante connexe unicyclique, avec un autre sommet de la même composante A_i.
     - Soit $e\in B$ et elle relie deux composantes unicycliques, d'où la violation de la troisième propriété du matroïde.

     Dans les deux cas, on trouve $\mid B\mid\leq\mid V_1\mid+\mid V_2\mid+\ldots+\mid V_k\mid$, d'où $\mid B\mid\leq\mid A\mid$ qui est en contradiction avec $\mid B\mid>\mid A\mid$ □

1.4.2 Partitionnement des réseaux en composantes unicycliques

Étant donné un graphe pondéré $G=(V,E)$ connexe et non orienté, et dont les poids des arêtes sont $w_e\geq 0$, $e\in E$. On suppose que $|V|=n$ et $|E|=m$. On s'intéresse à construire à partir du graphe G des composantes unicycliques de coût minimum.

1.4.2.1 L'algorithme glouton

Il est clair que nous sommes à la recherche d'une base optimale du matroïde bi-circualaire. On peut donc appliquer un algorithme glouton pour obtenir une telle base.

1.4.2.2 L'algorithme de couplage maximum de poids minimum

La deuxième solution, est d'utiliser un algorithme de recherche d'un couplage maximum de poids minimum, appliqué à un nouveau graphe construit à base de G. Ce nouveau graphe aura comme sommets ceux de G, ainsi que des nouveaux sommets qui sont toutes les arêtes de G. On construit donc un graphe biparti, d'une part on met les sommets de G, et de l'autre part,

```
1 Poser F = ∅ ;
2 w(e_1) ≤ w(e_2) ≤ ... ≤ w(e_m) ;
3 for i ← 1 to m do
4     if F ∪ {e_i} ∈ 𝔉 then
5         F := F ∪ {e_i} ;
6     end
7 end
```

Algorithm 5: Algorithme glouton

on met les arêtes comme étant des sommets. Quant aux arêtes de ce nouveau graphe biparti, seront construites en reliant un sommet i de $V(G)$ avec un autre sommet de la forme ij, avec ij est une arête dans le graphe G. On garde les mêmes pondérations que celles existantes sur G (voir Figure (1.4)). Sur ce nouveau graphe, on cherche alors un couplage maximum de poids minimum. Cette méthode s'avère intéressante pour répondre à notre besoin, puisque le résultat est un partitionnement en composantes connexes unicycliques et de poids total minimum.

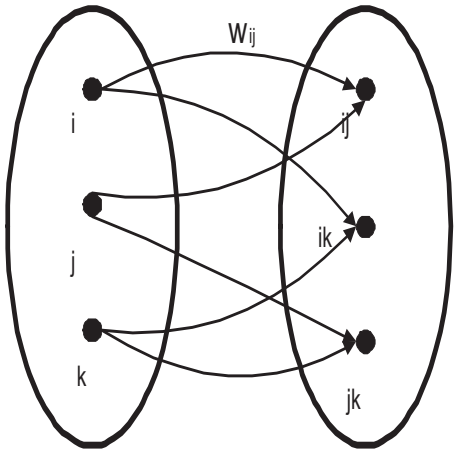

FIGURE 1.4 – Graphe biparti

Énoncé de l'algorithme

Soit U la partie gauche des sommets du nouveau graphe biparti, et W la partie droite.

1.5 Spectre des graphes

Étant donné un graphe G d'ordre n, et de taille m. On note par $A(G)$ ou simplement par A la matrice d'adjacence de G, et par L la matrice obtenue de la différence entre la matrice d'adjacence et la matrice des degrés D. L est connue sous le nom de *matrice d'admittance*, ou d'une manière générale *matrice du Laplacien*. On note aussi par $d_1, d_2, \ldots, d_n$ les degrés des

1 $M = \{e\}$, tel que e est l'arête de poids minimum orientée de U vers W ;
2 Orienter toutes les arêtes $e \in M$ de W vers U d'un poids $-w_e$;
3 Orienter toutes les arêtes e' qui ne sont pas dans M, de U vers W, d'un poids $+w_e$;
4 Poser $U_M = \{u \in U \setminus M\}$; et $W_M = \{u \in W \setminus M\}$;
5 Chercher tant que ça existe, le plus court chemin P de U_M vers W_M ;
6 Poser $M' = M\Delta P$ ($M\Delta P = (M \cup P) \setminus (M \cap P)$), et répéter les étapes 2, 3 et 4 ;
7 M' est le couplage recherché.

Algorithm 6: Algorithme de couplage sur un graphe biparti

sommets de G, ordonnés dans un ordre non-croissant $d_1 \geq d_2 \geq \ldots \geq d_n$ en posant $d_1 = \Delta$ et $d_n = \delta$ qui sont les degrés maximum et minimum respectivement.

On note en général les valeurs propres associées à la matrice d'adjacence par $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ qui sont ordonnées dans un ordre non-croissant, et par $\mu_1 \geq \mu_2 \geq \ldots \mu_{n-1} \geq \mu_n = 0$ les valeurs propres associées à la matrice L et qui sont aussi ordonnées dans le même ordre.

On remarque aussi que la valeur propre $\mu_n = 0$, et cela est facilement démontrable, en prenant un vecteur propre x dont toutes ses coordonnées sont égales à 1.
Ce dernier résultat sur la plus petite valeur propre nulle du Laplacien est en lien direct avec le nombre de composantes connexes du graphe G. En effet, la multiplicité de cette valeur propre représente le nombre de composantes connexes de G.

1.5.1 Quelques liens avec l'optimisation combinatoire

Il existe des problèmes d'optimisation combinatoire NP-Difficiles, et dont la solution est loin d'être trouvée pour des instances moyennes, et pour des temps raisonnables. C'est à ce stade que les valeurs propres interviennent afin de donner des bornes significatives aux solutions des problèmes traités.
Dans ce qui suit, nous donnerons quelques liens importants qui existent entre les valeurs propres extrêmes et des problèmes d'optimisation combinatoire. Plus de détails et d'autres liens peuvent être consultés dans [102], [103].

1.5.1.1 Lien avec le diamètre d'un graphe

Dans un graphe G, la distance entre deux sommets u et v notée par $d(u,v)$ est définie comme étant la longueur du plus court chemin joignant u et v. Le diamètre d'un graphe G est alors la distance maximum entre toute paire de sommets dans G. Le calcul du diamètre d'un graphe n'a pas que des intérêts théoriques mais possède aussi différents champs d'application. Lorsque les graphes sont utilisés comme modèles dans les réseaux de communication, le diamètre correspond alors aux retards dans le passage des messages dans le réseau, et donc joue un rôle important dans l'analyse des performances et l'optimisation des coûts.

Le diamètre est aussi lié aux valeurs propres. La proposition suivante le met en évidence.

Proposition 1.2: $D(G) \leq \lceil \frac{log(n-1)}{log(\frac{\mu_1+\mu_{n-1}}{\mu_1-\mu_{n-1}})} \rceil$

1.5.1.2 Lien avec le nombre isopérimétrique

Le problème du nombre isopérimétrique est l'un des plus anciens. Il consiste à trouver parmi toutes les courbes d'une longueur donnée, celle qui renferme la plus grande surface. En théorie des graphe, ce problème reste essentiellement le même. En effet, il consiste à calculer le nombre de sommets d'un ensemble S, et la surface interprétée par le nombre d'arêtes $e(S,\overline{S})$ séparant les ensembles S de $\overline{S}$, et le problème du nombre isopérimétrique est de s'intéresser au rapport $\frac{e(S,\overline{S})}{|S|}$.
Le nombre isopérimétrique d'un graphe G est donné par

$$i(G) = \min\{\frac{e(S,\overline{S})}{|S|} \mid S \subset V, 0 \leq |S| \leq \frac{n}{2}\}$$

Le calcul du nombre isopérimétrique d'un graphe est un problème NP-Difficile [101], [102]. Le lien de ce problème avec les valeurs propres extrêmes du Laplacien d'un graphe nous permet de borner ce nombre.

Proposition 1.3: Soit G un graphe pondéré d'ordre n. Δ est le plus grand degré dans G. On a alors

$$\frac{\mu_{n-1}}{2} \leq i(G) \leq \sqrt{2\Delta\mu_{n-1}}$$

L'inégalité $i(G) \leq \sqrt{2\Delta\mu_{n-1}}$ est dite de *Cheeger*.

1.5.1.3 Lien avec le problème de coupe maximum (Max-Cut)

Pour rappel, le problème de coupe maximum (Max-Cut) consiste à trouver une coupe de poids maximum dans un graphe. On note :

$$mc(G) = \max\{e(S,\overline{S}) \mid S \subset V(G)\}$$

Ce problème est classé NP-Difficile au sens de la complexité algorithmique [102].
La proposition suivante met en evidence une borne supérieure de la solution du problème de coupe maximum en fonction du rayon spectral du Laplacien correspondant.

Proposition 1.4: Étant donné un graphe G pondéré et d'ordre n. Alors

$$mc(G) \leq \frac{n}{4}\mu_1$$