UNIVERSITE ASSANE SECK DE ZIGUINCHOR

UFR DES SCIENCES ET TECHNOLOGIES

DEPARTEMENT DE MATHEMATIQUES

MEMOIRE DE MASTER DE MATHEMATIQUE APPLIQUEE

Sujet :

PROBLEME D'AFFECTATION EN MARCHE PUBLIC

Présenté par : ALGASSIMOU DIALLO

Sous la direction de : DOCTEUR DAOUDA NIANG DIATTA

Sous la supervision de : PROFESSEUR MARIE SALOMON SAMBOU

Devant le jury ci-après

NOM	***GRADE***	***QUALITE***
Pr SALOMON SAMBOU	Professeur titulaire	Président de jury
Pr DIENE NGOM	Maitre de conférences	Examinateur
Dr EMANUEL N CABRAL	Maitre-assistant	Examinateur
Dr YOUSSOU DIENG	Maitre de conférences	Examinateur
Dr DAOUDA N DIATTA	Maitre-assistant	Directeur

REMERCIEMENT

J'aimerais en premier lieu remercier le Docteur Daouda Niang DIATTA, mon directeur de mémoire pour avoir accepté de me proposer un sujet et de me guider sur mes premiers pas à la recherche. Ses conseils m'ont permis d'aller au bout de ce travail avec détermination. En plus d'être mon professeur, il est devenu un mentor, un guide et un grand frère.

Je suis également très honoré que le Professeur Marie Salomon SAMBOU soit le président de jury de mon mémoire. Mes remerciements s'adressent également au Professeur Diène NGOM, au Docteur Youssou DIENG et au Docteur Emmanuel Nicolas CABRAL d'avoir accepté de juger ce travail et d'en être les examinateurs. Mes remerciements vont aussi à l'endroit de tous les professeurs du département de Mathématiques.

Je remercie ma mère, mon père ainsi que toute ma famille pour leur soutien. Je tiens particulièrement à remercier mes amis et frères de cœur Ibrahima WADE, Bella BALDE, Fatou SOUMBOUNDOU (souriante) ainsi que toute la bande des ***ENE*** pour leur soutien, leur amitié et surtout leur contribution dans la construction de l'homme que je suis.

A ceux -là, j'associerais très volontiers Mamadou Boye DIALLO, Amadou NDIAYE, Mohamed NIAMBA et Bafodé DIAWARA, pour toutes ces années que nous avons partagées. J'ai beaucoup appris avec vous. Merci infiniment. Je ne saurais terminer sans exprimer ma gratitude à tous ceux qui n'ont pas hésité à m'offrir leur aide à maintes reprises tout au long de mes études.

DEDICACE

- A ma mère Assiatou Diallo et à mon père Mamadou Dian DIALLO. Vos conseils
  m'ont toujours guidé. Que Dieu vous accorde longue vie.
- A tous mes frères et soeurs, mes oncles et tantes. Vous m'avez toujours encouragé à persévérer dans l'effort.
- A Mon Oncle et idole Mamadou Moudjitaba DIALLO. Tu as toujours été une référence pour moi.
- A toute la bande des ***ENE*** de KOLDA. L'ascension continue.
- A toute la famille BADIANE de KOLDA. Vous êtes ma famille de cœur.

Qu'il me soit permis aussi, de renouveler ma reconnaissance déférente à ma mère et à mes amis.

Table des matières

Prologue

1 Présentation du problème

Un marché public est un contrat conclu à titre onéreux, entre un acheteur public appelé autorité contractante et des personnes publiques ou privées, et qui répond à un certain nombre de critères consignés dans un cahier de charges, le dossier d'appel d'offres, établi par l'autorité contractante. Au Sénégal, la réglementation prévoit trois types de marchés : fournitures, prestations intellectuelles, services et travaux. Lorsque la subdivision du marché est susceptible de présenter quelques avantages techniques stratégiques ou financiers, les travaux, fournitures, prestations ou services sont répartis en lots (ou allotis) donnant lieu chacun à un marché distinct. Le code des marchés publics stipule, en son article 47 : «Le dossier d'appel d'offres fixe le nombre, la nature et l'importance des lots, ainsi que les conditions imposées aux candidats pour souscrire à un ou plusieurs lots et les modalités de leur attribution et indique que la Commission d'évaluation des offres attribuera les marchés sur la base de la combinaison des lots la moins disante.» La « combinaison la moins disante », autrement dit le marché sera attribué de telle sorte que la somme des prix de revient des différents lots soit minimale. Nous voilà donc confrontés à un problème d'optimisation, une minimisation pour être juste. Il arrive très souvent et pour des raisons stratégiques que le nombre total de lots pouvant être attribués à un candidat soit limité à un, deux, trois, etc, en dépit de la possibilité de postuler à tous les lots.

Au terme de l'attribution, tout candidat qui se voit attribuer un nombre de lots dépassant la limite fixée, devra céder des lots. Dans ce cas, il revient exclusivement à l'autorité contractante le droit de faire céder ou de réattribuer les lots dans le but de minimiser le prix de revient total du marché, selon le contexte. C'est justement cette limitation du nombre de lots susceptibles d'être attribués à un seul candidat qui rend le problème intéressant et constitue la contrainte liée à l'optimisation sus évoquée.

2 Modélisation mathématique du problème et motivations

Considérons un marché subdivisé en p lots L_1, L_2, ..., L_p auxquels ont postulé les n candidats $C_1, C_2, ..., C_n$. Notons c_{ij} le prix proposé au lot L_i par le candidat C_j. Appelons $X = (x_{i,j})_{(i,j)\in[\![1,p]\!]\times[\![1,n]\!]}$ la matrice d'attribution définie de la manière suivante :

$$\text{Si le lot } L_i \text{ est attribué au candidat } C_j, \;\; x_{i,j} = 1. \;\; \text{Sinon, } x_{i,j} = 0.$$

Ainsi, $X \in \mathcal{M}_{p,n}(\{0,1\})$ espace des matrices à p lignes, n colonnes et à coefficients dans $\{0,1\}$.

Désignons par $r \in [\![1, p]\!]$ le nombre maximum de lots pouvant être attribué à un candidat.

Remarque.

- On peut supposer que chaque candidat postule à chaque lot quitte à admettre que $c_{i,j} = M$ avec M exagérément grand lorsque le candidat C_i ne compétit pas sur le lot L_j.
- Dans la réalité, on a $n \geqslant p$ puisque le nombre de candidats dépasse souvent de loin le nombre de lots.

Venons-en aux équations :
Chaque lot ne peut être attribué qu'à un unique candidat se traduit par :

$$\forall i \in \{1, 2, ..., p\}, \qquad \sum_{j=1}^{n} x_{i,j} = 1 \qquad (\mathcal{E}_1)$$

Un candidat ne peut gagner plus de r lots :

$$\forall j \in \{1, 2, ..., n\}, \qquad \sum_{i=1}^{p} x_{i,j} \leqslant r \qquad (\mathcal{E}_2)$$

Pour chaque $i \in \{1, 2, ..., p\}$, le prix de revient du lot L_i sera :

$$P_i = \sum_{j=1}^{n} c_{i,j}\, x_{i,j} \qquad (\mathcal{E}_3)$$

Enfin le prix de revient total du marché sera :

$$P = \sum_{i=1}^{n} P_i \quad = \sum_{i=1}^{p} \sum_{j=1}^{n} c_{i,j} x_{i,j} \qquad (\mathcal{E}_4)$$

L'objectif de notre travail est de proposer un algorithme ad-hoc qui résout le problème de minimisation du prix de revient total P défini dans l'équation $\mathcal{E}_4$, sous les contraintes des équations $\mathcal{E}_1$ et $\mathcal{E}_2$ en laissant varier la matrice d'attribution $X = (x_{i,j})_{(i,j) \in [\![1,p]\!] \times [\![1,n]\!]}$.

$$(\mathcal{P}): \begin{cases} \min \quad \sum_{i=1}^{p} \sum_{j=1}^{n} c_{i,j} x_{i,j} \\ X = (x_{i,j}) \in \mathcal{M}_{p,n}(\{0,1\}) \\ \forall i \in \{1, 2, ..., p\}, \sum_{j=1}^{n} x_{i,j} = 1, \\ \forall j \in \{1, 2, ..., n\}, \sum_{i=1}^{p} x_{i,j} \leqslant r. \end{cases}$$

Il y a un demi-siècle, Harold W. Kuhn a publié deux articles célèbres [12] [13] présentant l'algorithme Hongrois, la première méthode en temps polynomial pour résoudre le problème d'affectation qui est un cas particulier du problème $(\mathcal{P})$. Ce résultat historique a permis pour la première fois de résoudre des problèmes du monde réel qu'aucun ordinateur ne pouvait jusqu'alors gérer. L'algorithme Hongrois et d'autres résultats fondamentaux sur la programmation entière et linéaire ont donné naissance à un nouveau domaine de recherche, aujourd'hui connu sous le nom d'optimisation combinatoire.

Les problèmes d'optimisation combinatoire sont simples en apparence. En général, leur résolution s'avère très difficile et a nécessité la création d'outils mathématiques et algorithmiques qui constituent un corpus de connaissances ayant à la fois une grande richesse théorique et une forte cohérence interne. Le mot combinatoire évoque, outre le caractère fini de l'ensemble objet de l'étude, le cardinal très grand de cet ensemble ; ou plutôt l'existence d'une « structure » qui, à partir d'un nombre limité d'éléments, permet d'engendrer une quantité astronomique de situations à comparer. De manière générale, le schéma suivant est typique des problèmes d'optimisation combinatoire : on se donne un ensemble fini E de cardinal n mais on s'intéresse à une partie $\mathcal{P}(E)$, famille de sous ensemble de E ; $\mathcal{P}(E)$ est de cardinal 2^n. Il est alors clair que la procédure triviale de recherche de solution par énumération de l'ensemble des cas, si elle est théoriquement possible, conduit à des procédures non efficaces.

3 Approche et travail effectué dans ce mémoire

L'objectif de ce mémoire est de proposer et implanter un algorithme ad-hoc efficace de résolution du problème d'optimisation $(\mathcal{P})$. Pour ce faire, nous allons procéder de la manière suivante :

1. **Notion préliminaire.** Dans cette partie, nous introduisons les notions de permutations, graphes, réseaux, ..., indispensables à la compréhension des méthodes de résolution de notre problème. Nous allons surtout insister sur la notion de couplage en définissant les différents types de couplages puis énoncer quelques théorèmes fondamentaux liés à cette notion et étudier deux méthodes de constructions de certains couplages.

2. **Étude d'un cas particulier bien connu du problème.** Il s'agit du cas où les contraintes $\mathcal{E}_2$ sont remplacées par les équations

$$\forall j \in \{1, 2, ..., n\}, \sum_{i=1}^{n} x_{i,j} = 1.$$

   Ces dernières équations imposent à chaque candidat de gagner exactement un lot. Ce problème est connu sous le nom de problème de somme linéaire d'affectation (PSLA). Nous allons commencer par définir le problème. Ensuite, nous allons faire une interprétation fonctionnelle, matricielle et graphique du problème. Nous allons aussi proposer une approche géométrique du PSLA dans laquelle nous mettrons en évidence son équivalence avec un programme linéaire continue. Enfin, après avoir montré l'inefficacité de la méthode du simplexe dans la résolution de notre problème, nous allons présenter et implémenter deux approches de la méthode **Hongrois** qui est la première méthode de résolution en temps polynomial du PSLA.

3. **Généralisation des algorithmes de résolution de problèmes d'affectation.** Dans cette partie, nous allons généraliser l'algorithme Hongrois pour résoudre le problème ($\mathcal{P}$) sus évoqué. Pour résoudre un problème d'affectation, l'algorithme Hongrois, à travers un système de chemins d'augmentations, effectue le processus d'affectation en respectant les contraintes du problème, c'est à dire chaque lot est affecté à un seul candidat et vice versa. Notre idée est de permettre aux candidats d'avoir plus d'un lot. Ainsi, tant qu'un candidat n'a pas atteint le nombre d'affectations maximum, l'algorithme peut l'utiliser pour affecter un nouveau lot.

4. **Implémentation des différents algorithmes.** Pour l'implémentation des algorithmes étudiés et celui adapté, nous avons utilisés le langage de programmation **Python**. C'est un langage qui dispose de structures de données de haut niveau et permet, grâce à ses bibliothèques spécialisées, une approche simple mais efficace de la programmation scientifique. Nous avons travaillés essentiellement avec la bibliothèque **networks** pour la manipulation des graphes et la bibliothèque **numpy** pour celle des matrices. Aussi, nous avons utilisés les objets **set** et **list** pour les ensembles et les **dictionnaires** pour les étiquetages.

Chapitre 1
Notions préliminaires

Dans ce chapitre, nous rappelons quelques notions mathématiques nécessaires à la compréhension des problèmes abordés dans ce mémoire. Pour une étude détaillée de ces notions, le lecteur peut consulter les ouvrages [3][9]

1.1 Les permutations

En mathématique, la notion de **permutation** exprime l'idée de réarrangement d'objets discernables. Une permutation d'objets distincts, rangés dans un certain ordre, correspond à un changement de l'ordre de succession de ces objets. La permutation est une des notions fondamentales en combinatoire. Elle intervient naturellement dans les problèmes d'optimisation combinatoire.

Définition 1.1. (permutation.) *Soit U un ensemble fini. On appelle permutation de U, une application bijective de U dans U. On note S_n l'ensemble des permutations d'un ensemble fini à n éléments. Le cardinal de S_n est $n!$.*

Les permutations sont décrites en plaçant les éléments qui vont être permutés dans l'ordre naturel sur une première ligne, et leur image respective sur une deuxième ligne. Ainsi, une permutation φ d'un ensemble $U = \{1, 2, \ldots, n\}$ s'écrit :

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ \varphi(1) & \varphi(2) & \ldots & \varphi(n) \end{pmatrix}$$

Il arrive également qu'une permutation φ, d'un ensemble $U = \{1, \ldots, n\}$ à n éléments, soit désignée par la liste $(\varphi(1), \ldots, \varphi(n))$ des images de $1, \ldots, n$. Ainsi, la permutation

$$\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 1 & 3 & 7 & 6 & 4 & 5 \end{pmatrix}$$

peut être désignée par $\varphi = (2, 1, 3, 7, 6, 4, 5)$.

Définition 1.2. (matrice de permutation) *Une matrice carrée de taille $n \times n$, $X = (x_{ij})$ est dite de permutation si elle vérifie les propriétés suivantes :*

- $\forall (i, j) \in [\![1, n]\!] \times [\![1, n]\!], x_{ij} = \{0, 1\}$*;*
- $\forall i \in [\![1, n]\!]$, $\sum_{j=1}^{n} x_{ij} = 1$ *(il y a un et un seul 1 par ligne);*

- $\forall j \in [\![1,n]\!]$, $\sum_{i=1}^{n} x_{ij} = 1$ *(il y a un et un seul 1 par colonne).*

A chaque permutation φ d'un ensemble $U = \{1, 2, ..., n\}$ correspond une unique matrice de permutation de taille $n \times n$, $X_\varphi = (x_{ij})$ définie par :

$$x_{ij} = \begin{cases} 1 & \text{si} \quad \varphi(i) = j \\ \\ 0 & \text{sinon.} \end{cases}$$

Exemple. La permutation définie par $\varphi = (2, 4, 3, 1)$ correspond à la matrice de permutation

$$X_\varphi = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

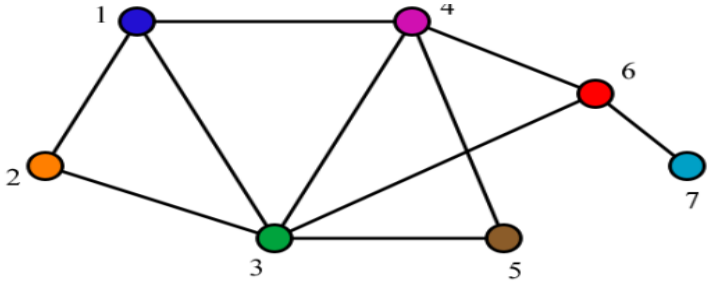
1.2 Graphe

De manière générale, les graphes permettent de représenter les relations entre les éléments d'un ensemble. Ils constituent donc une méthode de pensée qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommets et d'arcs : réseaux de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques, etc. Dans cette section, nous introduisons quelques notions de bases de la théorie des graphes. Pour plus d'informations sur le sujet, le lecteur pourra consulter l'ouvrage [9].

Définition 1.3. (graphe simple) *Un graphe simple fini $G = (S; E)$ est défini par la donnée des ensembles $S = \{s_1, s_2, ..., s_n\}$ (ses sommets) et $E = \{e_1, e_2, ..., e_m\}$ (ses arêtes).*

Une ***arête*** *e est définie par une paire* ***non ordonnée*** *de sommets, appelés les extrémités de e. Si l'arête e relie les sommets u et v, on dira que ces sommets sont adjacents, ou incidents avec e, ou bien que l'arête e est incidente avec les sommets u et v.*

Exemple.



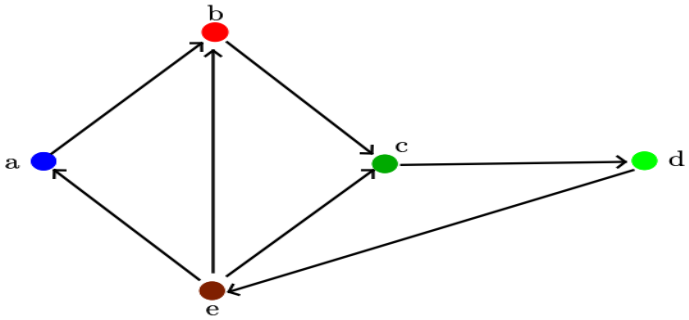
$S = \{1, 2, 3, 4, 5, 6, 7\}$
$E = \{(1,2), (1,3), (1,4), (2,3), (3,4), (3,5), (3,6), (4,5), (4,6), (6,7)\}$

Figure 1.1. Graphe simple

Définition 1.4. (graphe orienté) *Un graphe orienté fini $G=(S;E)$ est défini par les ensembles $S=\{s_1,s_2,\ldots,s_n\}$ (ses sommets) et $E=\{e_1,e_2,\ldots,e_m\}$ (ses arcs). Un arc e est défini par une paire* ***ordonnée*** *de sommets.*

Ainsi l'arc $e=(u,v)$ va de u vers v. On dit aussi que u est l'extrémité initiale de e ou prédécesseur de v et v est l'extrémité finale de e ou successeur de u.

Exemple.



$S=\{a,b,c,d,e\}, E=\{(a,b),(b,c),(c,d),(d,e),(e,b),(e,a),(e,c)\}$

Figure 1.2. Graphe orienté

Remarque. Lorsqu'on donne un sens aux arêtes d'un graphe simple, on obtient un graphe orienté.

Définition 1.5. (graphe connexe, graphe complet) *Soit $G=(S;E)$ un graphe.*

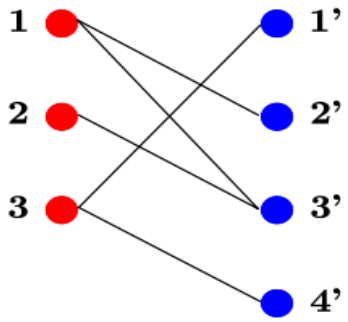
- *G est dit connexe s'il est possible, à partir de n'importe quel sommet, de rejoindre tous les autres en suivant les arêtes.*
- *G est dit complet si chaque sommet du graphe est relié directement à tous les autres sommets.*

Définition 1.6. (graphe partiel, sous graphe) *Soit $G=(S;E)$ un graphe.*

- *Le graphe $G'=(S;E')$ est un graphe partiel de G, si $E'\subseteq E$. Autrement dit, on obtient G' en enlevant une ou plusieurs arêtes au graphe G.*
- *Soit $A\subseteq S$. Le sous graphe de G' induit par A est le graphe $G'=(A;E(A))$ dont l'ensemble des sommets est A et l'ensemble des arêtes $E(A)$ est formé de toutes les arêtes de G ayant leurs deux extrémités dans A.*

Définition 1.7. (graphe biparti) *Un graphe fini $G=(S;E)$ est dit biparti si S peut être partitionné en deux sous-ensembles disjoints U et V tel que chaque arête de E relie un sommet de U à un sommet de V et qu'il n'y ait pas d'arête qui relie deux sommets d'un même sous-ensemble. On le notera $G=(U,V;E)$.*

Exemple.



$U = \{1, 2, 3\}, V = \{1', 2', 3', 4'\}$
$E = \{(1, 2'), (1, 3'), (2, 3'), (3, 1'), (3, 4')\}$

Figure 1.3. Graphe biparti

Remarque. Un graphe biparti n'est rien d'autre que le graphe d'une correspondance.

Définition 1.8. (degré) *Le degré d'un sommet $s \in S$, noté $d(s)$, est le nombre d'arêtes incidentes à s. Le degré du graphe est le degré maximum de tous les sommets. Un graphe est dit k-régulier si tous ses sommets sont de degré k.*

Exemple. le graphe simple de la Figure 1.1 est de degré 5.

Comme pour les permutations, à chaque graphe fini correspond une unique matrice à coefficients dans $\{0, 1\}$. Il est plus pratique d'utiliser la représentation matricielle d'un graphe lorsqu'on souhaite le traiter automatiquement par des algorithmes. Dans le cas des graphes bipartis, la définition suivante permet d'accéder à une représentation matricielle.

Définition 1.9. (matrice d'adjacence d'un graphe.) *Soit $G = (U, V; E)$ un graphe biparti fini. On appelle matrice d'adjacence $A=(a_{ij})$ du graphe G la matrice de taille $|U| \times |V|$ définie par :*

$$a_{ij} = \begin{cases} 1 & \text{si } (i, j) \in E \\ 0 & \text{sinon.} \end{cases}$$

Définition 1.10. (le permanent d'un graphe biparti) *Soit $G = (U, V; E)$ un graphe biparti avec $|U| = |V|$. On appelle permanent de G le nombre d'applications bijectives distinctes que l'on peut extraire de la correspondance représentée par G. On le note* $\mathrm{per}(G)$ *ou* $\mathrm{per}(A)$ *si A désigne la matrice d'adjacence de G.*

Proposition 1.11. *Soit $G = (U, V; E)$ un graphe biparti avec $|U| = |V| = n$ de matrice d'adjacence $A = (a_{i,j})_{(i,j)\in[\![1,n]\!]\times[\![1,n]\!]}$. On a :*

$$\mathrm{per}(A) = \sum_{\varphi \in S_n} a_{1\varphi(1)} a_{2\varphi(2)} \ldots a_{n\varphi(n)} \tag{1.1}$$

Remarque. Le produit $a_{1\varphi(1)}a_{2\varphi(2)}...a_{n\varphi(n)} = 1$ si φ est une permutation et 0 sinon. Ainsi per(G) compte bien le nombre de bijections différentes que l'on peut extraire de la correspondance représentée par le graphe biparti de matrice d'adjacence A. Déterminer per(G) est un problème NP-difficile.

Définition 1.12. (matrice d'adjacence complémentaire) *Soit $G = (U, V; E)$ un graphe biparti. La matrice d'adjacence complémentaire $B = (b_{ij})$ est une matrice de taille $|U| \times |V|$ définie par :*

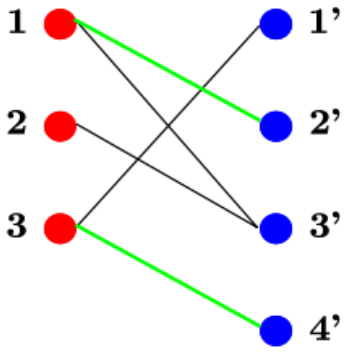
$$b_{ij} = \begin{cases} 0 & \text{si } (i,j) \in E, \\ 1 & \text{sinon.} \end{cases}$$

Remarque. Les indices (i, j) pour lesquels la matrice d'adjacence $A = (a_{i,j})$ d'un graphe biparti admet 0 comme entrée sont exactement ceux pour lesquels sa matrice d'adjacence complémentaire admet 1 comme entrée et vice versa.

1.2.1 Couplage

Définition 1.13. (couplage) *Soit $G = (U, V; E)$ un graphe biparti. Un couplage M de G est un sous-ensemble d'arêtes de G tel que chaque sommet de G rencontre au plus une arête dans M.*

Exemple.



$M = \{(1, 2'), (3, 4')\}$ est un couplage
$M' = \{(1, 3'), (2, 3')\}$ n'est pas un couplage

Figure 1.4. Couplage d'un graphe

Remarque. Soit $G = (U, V; E)$ un graphe biparti. Il est important de remarquer qu'un **couplage** M de G est le graphe d'une **fonction injective** extraite de la correspondance représentée par le graphe biparti G. Ainsi il y'a autant de couplage que de fonctions injectives extraites de la correspondance représentée par G.

Soient $G=(U,V;E)$ un graphe biparti et $M \subset E$ un couplage de G. Il est parfois possible de construire un nouveau couplage M' de G en adjoignant à M une ou plusieurs arêtes de $E \backslash M$.

Définition 1.14. (couplage maximal) *Soient $G=(U,V;E)$ un graphe biparti et $M \subset E$ un couplage de G. Le couplage M est dit maximal s'il ne peut être agrandi par adjonction d'une arête de $E \backslash M$.*

Exemple.

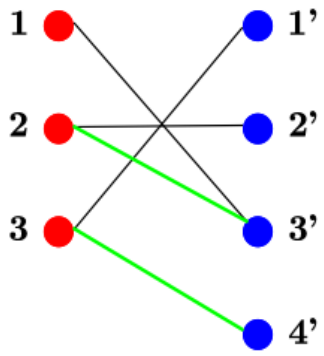


Figure 1.5. Couplage maximal

Soient $G=(U,V;E)$ un graphe biparti et $\mathcal{M}$ l'ensemble des couplages de G. L'ensemble $\mathcal{M}$ est fini car inclus dans $\mathcal{P}(E)$(ensemble des parties de E) qui est fini de cardinal $2^{|E|}$.

Définition 1.15. (couplage maximum) *Soit $G=(U,V;E)$ un graphe biparti et $\mathcal{M}$ l'ensemble des couplages de G. On appelle couplage maximum de G et on note $M_{\max}$ tout couplage de G de cardinal $\max\{|M|, M \in \mathcal{M}\}$. Ainsi*

$$|M_{\max}| = \max\{|M|, M \in \mathcal{M}\}.$$

Exemple.

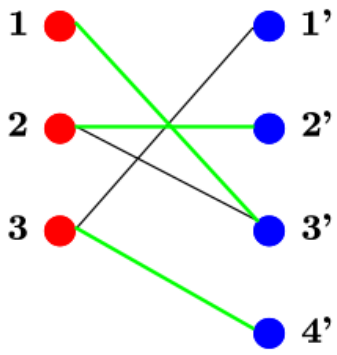


Figure 1.6. Couplage maximum

Définition 1.16. (couplage parfait) *Soit $G=(U,V;E)$ un graphe biparti tel que $|U|=|V|$. Un couplage M de G est dit parfait si $|M|=|U|$.*

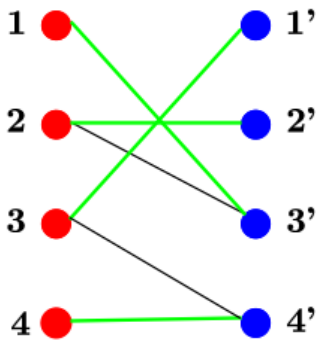


Figure 1.7. Couplage parfait

Remarque. Un couplage parfait peut être vu comme le graphe d'une application bijective extraite de la correspondance représentée par G. Décider de l'existence d'un couplage parfait sur un graphe biparti donné est une étape importante dans la résolution de la majeure partie des problèmes faisant intervenir des graphes bipartis. Le problème est connu sous le nom de **problème de couplage biparti parfait.** Remarquons que le nombre de couplages parfaits distincts que l'on peut extraire d'un graphe biparti G est égal à $\mathrm{per}(G)$.

1.3 Réseaux de transports

Les réseaux de transports peuvent être vus comme des graphes orientés munis d'une structure supplémentaire. La théorie résultante possède une grande variété d'applications dans la modélisation et l'analyse des réseaux de transport urbain, les réseaux de télécommunication, etc. Dans cette section, nous présentons quelques notions de bases sur ce sujet. Pour une étude détaillée, le lecteur peut consulter l'ouvrage [9].

Définition 1.17. (réseau) *Un réseau $\mathcal{N}=(N,A,q)$ est un graphe orienté $G=(N;A)$ muni d'une application d'arcs $q: A\to\mathbb{R}^+$ qui vérifie les conditions suivantes :*

- *Il existe un sommet $s\in N$, appelé source, qui n'a pas de prédécesseur;*
- *Il existe un sommet $t\in N$, appelé cible, qui n'a pas de successeur;*
- *Chaque sommet du graphe se trouve sur le chemin de la source à la cible;*
- *Il n'existe pas deux arcs opposés reliant deux mêmes sommets* ***(graphe anti-symétrique)****;*

Les éléments de N sont appelés des noeuds, ceux de A des arcs et q est appelé la capacité des arcs.

Exemple.

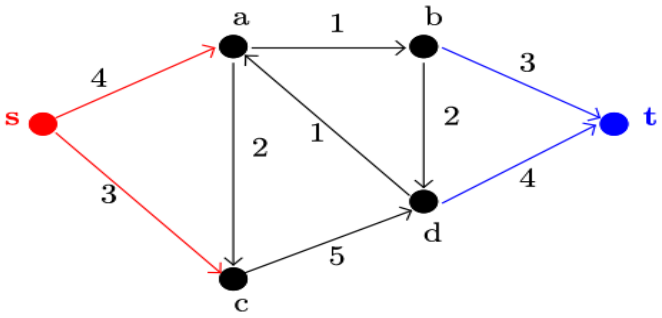


Figure 1.8. Réseau

1.3.1 Flux dans un réseau

Définition 1.18. (flux) *Soit $\mathcal{N}=(N;A,q)$ un réseau de source $s\in N$ et de cible $t\in N$. Un flux dans le réseau $\mathcal{N}$ ou un (s,t)-flux est une fonction $f\colon A\longrightarrow\mathbb{R}$ satisfaisant les conditions suivantes:*

$$\sum_{(i,j)\in A} f(i,j)=\sum_{(j,k)\in A} f(j,k),\forall j\in N\backslash\{s,t\} \tag{1.2}$$

$$0\leqslant f(i,j)\leqslant q(i,j),\forall\,(i,j)\in A \tag{1.3}$$

Remarque. La condition (1.2) exige que le flux entrant dans un noeud j soit égal au flux qui en sort. Elle est connue sous le nom de loi de conservation du flux. La condition (1.3), appelée contrainte de capacité, impose que le flux qui traverse un arc ne soit pas supérieur à sa capacité.

Définition 1.19. (valeur d'un flux) *La valeur $\mathcal{Z}(f)$ d'un flux dans un réseau $\mathcal{N}=(N;A,q)$ de source $s\in N$ et de cible $t\in N$ est définie par la formule suivante :*

$$\mathcal{Z}(f)=\sum_{(s,i)\in A} f(s,i)=\sum_{(j,t)\in A} f(j,t). \tag{1.4}$$

Remarque. Dans un réseau donné, la recherche de flux à valeur maximale est un problème fondamental étroitement lié aux questions de dimensionnement dans les applications pratiques comme les réseaux de télécommunications, de transports et d'assainissements.

1.3.2 Coupe d'un réseau

Définition 1.20. (coupe) *Soit $\mathcal{N}=(N,A,q)$ un réseau de source s et de cible t. Une (s,t)-coupe C du réseau $\mathcal{N}$ est une partition $(X,\bar{X})$ de l'ensemble des nœuds N telle que $s\in X$ et $t\in\bar{X}$.*

On désigne par $\delta^+(X)$ et $\delta^-(X)$ les sous-ensembles d'arcs suivants :

$$\delta^+(X):=\{(i,j)\in A\mid i\in X\,,j\in\bar{X}\}$$

$$\delta^-(X):=\{(i,j)\in A\mid j\in X\,,i\in\bar{X}\}$$

$\delta(X) = \delta^-(X) \cup \delta^+(X)$ *est appelé* ***cocycle*** *du réseau.*

Définition 1.21. (valeur d'une coupe) *Soit $\mathcal{N} = (N, A, q)$ un réseau de source s et de cible t et $C = (X, \bar{X})$ une (s,t)-coupe quelconque de $\mathcal{N}$. On appelle valeur de la (s,t)-coupe C la quantité :*

$$v(C) = \sum_{(i,j)\in\delta^+(X)} q(i,j). \tag{1.5}$$

Lemme 1.22. *Soient $\mathcal{N} = (N, A, q)$ un réseau de source s et de cible t et $C = (X, \bar{X})$ une (s,t)-coupe quelconque de $\mathcal{N}$. La valeur $\mathcal{Z}(f)$ d'un (s,t)-flux f dans le réseau $\mathcal{N}$ vaut :*

$$\mathcal{Z}(f) = \sum_{(i,j)\in\delta^+(X)} f(i,j) \quad - \sum_{(i,j)\in\delta^-(X)} f(i,j) \tag{1.6}$$

Démonstration. Ajoutons dans le réseau $\mathcal{N} = (N, A, q)$ un arc fictif (t, s) de capacité infinie qu'on appellera arc de retour. La loi de conservation du flux implique :

$$\forall j \in X, \sum_{(i,j)\in A} f(i,j) \quad - \sum_{(j,k)\in A} f(j,k) = 0.$$

Sommons ces égalités sur tous les sommets de X. Les flux sur les arcs ayant leurs deux extrémités dans X vont s'éliminer et, si $\mathcal{Z}(f)$ est le flux sur l'arc retour, il restera :

$$\mathcal{Z}(f) + \sum_{(i,j)\in\delta^-(X)} f(i,j) - \sum_{(i,j)\in\delta^+(X)} f(i,j) = 0.$$

Lorsqu'on transpose, on obtient le résultat annoncé. □

1.4 Théorème du mariage et existence de couplage

Dans la section 1.2 nous avons évoqué le problème du couplage parfait dans un graphe biparti et de son importance dans la résolution de certains problèmes d'optimisation combinatoire. Dans cette section nous allons énoncer des théorèmes fondamentaux liés à cette notion et les relations entre ceux-ci. Pour plus de détails le lecteur peut consulter [3].

Définition 1.23. (ensemble des voisins d'un sommet) *Soit $G = (U, V; E)$ un graphe biparti avec $U = \{1, 2, ..., n\}$ et $V = \{1, 2, ..., s\}$. Pour tout $i \in U$, l'ensemble des voisins de i, noté $N(i)$, désigne l'ensemble de tous les sommets connectés à i par une arête de E :*

$$N(i) := \{j \in V \mid (i,j) \in E\} \tag{1.7}$$

Soit U' une partie de U. On désigne par $N(U')$ l'union des voisins des éléments de U':

$$N(U') = \bigcup_{i\in U'} N(i). \tag{1.8}$$

Le théorème suivant donne une condition nécessaire et suffisante pour l'existence d'un couplage de $G=(U,V;E)$ de cardinal $|U|$.

Théorème 1.24. (Hall) *Soit $G=(U,V;E)$ un graphe biparti. Il est possible de coupler chaque sommet de U avec un sommet de V si et seulement si*

$$\forall\, U'\subseteq U,\ |U'|\leqslant|N(U')| \qquad \textbf{(condition de Hall)} \tag{1.9}$$

Si en plus, nous supposons que $|U|=|V|$, nous pouvons exprimer le Théorème 1.24 comme suit:

Théorème 1.25. (théorème du mariage) *Soit $G=(U,V;E)$ un graphe biparti avec $|U|=|V|$. Il existe un couplage parfait dans G si et seulement si G satisfait la condition de Hall.*

Démonstration. du Théorème 1.24

- Montrons que la condition est nécessaire.

  Supposons qu'il soit possible de coupler chaque sommet de U à un sommet de V. Soit $U'\subseteq U$. Puisqu'il est possible de coupler chaque sommet de U' à un sommet de V alors $N(U')$ a au moins autant d'éléments que U'. Par conséquent,

  $$|U'|\leqslant|N(U')|.$$

- Montrons que la condition est suffisante c'est à dire si la condition de Hall est vérifiée, il est possible de coupler chaque sommet de U à un sommet de V. Pour cela nous allons procéder par récurrence sur le cardinal de U. Supposons que la condition de Hall soit vérifiée.

  Si $|U|=1$, alors l'élément peut être couplé à un de ses voisins au choix.

  Supposons qu'il est possible de coupler tout les éléments de U de tout graphe $G=(U,V;E)$ qui remplit la condition de Hall avec $|U|\leq k$.

  Soit $G=(U,V;E)$ avec $|U|=k+1$ qui remplit la condition de Hall. Ainsi, deux cas se posent à nous :

  → $\forall U'\subseteq U,\ |U'|<|N(U')|$. On choisit un sommet $i\in U'$, on le couple avec un de ses voisins $j\in N(i)\subseteq N(U')$, en supprimant les sommets i, j et les arêtes qui leurs sont incidentes, on obtient un nouveau graphe $\bar{G}=(\bar{U},\bar{V};\bar{E})$, avec $|\bar{U}|=k$. $\bar{G}$ remplit la condition de Hall. Car chaque sommet i de U perd au plus un voisin. Donc $\bar{G}$ contient un couplage de taille k et si on ajoute $e=(i,j)$ on obtient un couplage de taille $k+1$.

  → $\exists U'\subseteq U,\ |U'|=|N(U')|$. Grâce à notre hypothèse, nous pouvons coupler chaque élément de U' à un de ses voisins approprié. Nous allons supprimer U' et $N(U')$ et les arêtes qui leurs sont incidentes. Nous obtenons un graphe $\bar{G}=(\bar{U},\bar{V};\bar{E})$ avec $|\bar{U}|\leqslant k$. Montrons que ce nouveau graphe remplit la condition de Hall.

Supposons qu'il ne remplit pas la condition de Hall. Alors $\exists W \subseteq \bar{U}$ tel que $|W| > |N_{\bar{G}}(W)|$. On sait que $N_{\bar{G}}(W) = N(W) \setminus N(U')$. Ainsi,

$$|N(U' \cup W)| = |N(U')| + |N(W) \setminus N(U')| = |N(U')| + |N_{\bar{G}}(W)| < |U'| + |W| = |U \cup W|$$

Donc G ne remplit pas la condition de Hall. Ce qui est absurde. Par conséquent, $\bar{G}$ remplit la condition de Hall. Puisque $|\bar{U}| \leq k$, chaque sommet de U peut être couplé. Si on ajoute les couples de U', on obtient un couplage de cardinal $|U|$. ce qui conclut notre démonstration. □

Corollaire 1.26. *Tout graphe biparti k-régulier* $(k \geqslant 1)$ *admet un couplage parfait.*

Démonstration. Soit $G = (U, V; E)$ un graphe biparti k-régulier. Puisque $k|U| = \sum_{j \in U} d(j) = \sum_{j \in V} d(j) = k|V|$, alors $|U| = |V|$. Soient $U' \subseteq U$, $E_{U'}$ l'ensemble des arêtes qui sont incidentes à U et $E_{N(U')}$ l'ensemble des arêtes incidentes à $N(U')$. Il est évident que $E_{U'} \subseteq E_{N(U')}$. Donc $k|U'| = |E_{U'}| \leq |E_{N(U')}| = k|N(U')|$. Ce qui implique que $|U'| \leq |N(U')|$. Par conséquent, G remplit la condition de Hall et $|U| = |V|$. □

Nous pouvons interpréter le Théorème 1.24 de Hall dans le langage matriciel en utilisant la matrice d'adjacence de G. La condition de Hall indique que, pour $k = 1, 2, ..., n-1$ la matrice A ne contient pas de sous matrice nulle de taille $(k+1) \times (n-k)$.

Théorème 1.27. (Frobenius[8]) *Soit A une matrice de taille* $n \times n$ *avec 0 et 1 comme coefficients. La matrice A contient une matrice de permutation si et seulement si, pour tout* $k = 1, 2, ..., n-1$, *A ne contient pas de sous matrice nulle de taille* $(k+1) \times (n-k)$.

Corollaire 1.28. *Chaque sous matrice de taille* $(k+1) \times (n-k)$ *de toute matrice de permutation* $(n \times n)$ *contient au moins un coefficient de valeur 1.*

König a prouvé un théorème sur les couplages qui se révèle être l'une des pierres angulaires des algorithmes pour les problèmes de couplage. Avant d'énoncer le théorème de couplage de König et prouver qu'il est équivalent au théorème de Hall, nous avons besoin de la définition suivante.

Définition 1.29. (couverture de sommets) *Étant donné un graphe G, une couverture de sommets dans G est un sous-ensemble C des sommets de G tel que chaque arête coïncide avec au moins un sommet de C.*

Soient $G = (U, V; E)$ un graphe biparti et $\mathcal{C}$ l'ensemble des couvertures de sommets de G. L'ensemble $\mathcal{C}$ est non vide (car $U \cup V \in \mathcal{C}$) et fini (car inclus dans $\mathcal{P}(U \cup V)$ qui est fini de cardinal $2^{|U \cup V|}$).

Définition 1.30. (couverture de sommet minimum) *Soit $G = (U, V; E)$ un graphe biparti et $\mathcal{C}$ l'ensemble des couvertures de sommets de G. On appelle couverture de sommets minimum de G et on note $C_{\min}$ toute couverture de sommets de G de cardinal $\min\{|C|, C \in \mathcal{C}\}$. Ainsi*

$$|C_{\min}| = \min\{|C|, C \in \mathcal{C}\}.$$

Le Théorème suivant établit que le problème de recherche de couplages et le problème de recherche de couvertures de sommets dans un graphe biparti sont duaux dans le sens suivant : trouver un couplage maximal revient à trouver une couverture de sommets minimale.

Théorème 1.31. (théorème de couplage de König[11]) *Dans un graphe biparti le nombre de sommets d'une couverture de sommets minimum est égal au nombre d'arêtes d'un couplage maximum :*

$$|C_{\min}| = |M_{\max}|$$

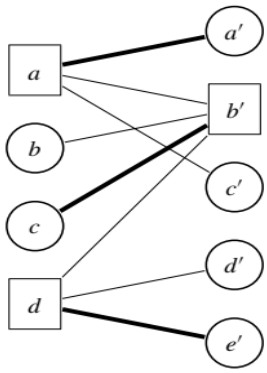


Figure 1.9. Couplage maximum (lignes gras) **versus** couverture de sommets minimum (sommets carrés)

Nous montrons maintenant que le théorème de König est équivalent au théorème de Hall.

Théorème 1.32. *Le théorème de Hall est équivalent au théorème de couplage de König.*

Démonstration. Soit $G = (U, V; E)$ un graphe biparti.

1. Montrons que le théorème de Hall implique celui de König. Soit M un couplage de G et C une couverture de sommets dans G. Par définition d'une couverture de sommets, tout élément de M rencontre au moins un élément de C. Par définition d'un couplage, deux éléments distincts de M ne peuvent rencontrer le même élément de C. Ainsi, on a $|M| \leq |C|$. Reste à prouver qu'il existe un couplage M de taille égale à celle d'une couverture de sommets minimum dans G.

Soit C une couverture de sommets minimum dans G. On définit un nouveau graphe biparti $G'=(U',V';E')$ avec $U'=U\cap C, V'=V\setminus C$ et $E'=E\cap(U'\times V')$. Supposons que ce graphe ne remplit pas la condition de Hall. Dans ce cas il existe $W\subseteq U'$ tel que $|W|>|N(W)|$. $C'=(U'\setminus W)\cup N(W)\cup(V\cap C)$ est une couverture de sommets dans G. Car, chaque arête qui a un sommet dans $U\cap C$ a un sommet soit dans $U'\setminus W$ ou dans $N(W)$. Les autres ont un sommet dans $V\cap C$. Nous obtenons donc :

$$|C'| = |U' \setminus W| + |N(W)| + |V \cap C| < |U' \setminus W| + |W| + |V \cap C| = |U\cap C|+|V\cap C|=|C|$$

ce qui contredit l'hypothèse selon laquelle C est minimum. Donc, G' remplit la condition de Hall. Par conséquent, on peut coupler chaque élément de $U\cap C$ à un de $V\setminus C$.

On définit un autre nouveau graphe biparti $G''=(U'', V''; E'')$ avec $U''=V\cap C, V''=U\setminus C$ et $E''=E\cap(U''\times V'')$. En suivant le même procédé on montre que G'' remplit la condition de Hall. par conséquent, on peut coupler chaque élément de $V\cap C$ à un de $U\setminus C$. Ainsi il existe un couplage M tel que $|M|=|U\cap C|+|V\cap C|=|C|$.

2. Montrons que le théorème de König implique celui de Hall. Soit $G=(U,V;E)$ un graphe biparti qui remplit la condition de Hall. Cela signifie qu'en particulier chaque sommet de $i\in U$ coïncide avec une arête $e\in E$. Nous savons que pour G le cardinal de la couverture minimum est égal à celui d'un couplage maximum. Donc si nous montrons que U est une couverture de sommets minimum alors son cardinal sera égal à celui du couplage maximum. Ce qui prouvera Hall.

   Soit C une couverture de sommet quelconque dans G. Si $U'=U\setminus C\neq\emptyset$, alors il n'y a pas d'arête qui a un sommet dans U' et un autre dans $V\setminus C$. Donc $N(U')\subseteq(V\cap C)$ et

$$|C|=|U\cap C|+|V\cap C|\geq|U\cap C|+|N(U')|\geq|U\cap C|+|U'|=|U|$$

   Selon le théorème de König il existe un couplage de taille $|U|$. En d'autre terme chaque sommet de U peut être couplé à un sommet de V. □

Le théorème de König est un cas particulier du théorème de Ford-Fulkerson. Soit $\mathcal{N}=(N,A,q)$ un réseau de source s et de cible t. Notons $\mathcal{F}$ l'ensemble des flux sur le réseau $\mathcal{N}$:

$$\mathcal{F}:=\{f\colon A\longrightarrow\mathbb{R}\,|\,(1.2)\text{ et }(1.3)\text{ sont satisfaites}\}. \tag{1.10}$$

Notons $\mathcal{X}$ l'ensemble des coupes du réseau $\mathcal{N}$:

$$\mathcal{X}:=\{(X,\bar{X})|s\in X, t\in\bar{X}\text{ et }X\uplus\bar{X}=N\}. \tag{1.11}$$

Le théorème de Ford-Fulkerson établit le fait que rechercher un flux de valeur maximale et rechercher une coupe de valeur minimale sont deux problèmes duaux.

Théorème 1.33. (Ford-Fulkerson[7]) *Soient $\mathcal{N} = (N,\ A,\ q)$ un réseau, $\mathcal{F}$ l'ensemble de ses flux et $\mathcal{X}$ l'ensemble de ses coupes. On a :*

$$\max_{f\in\mathcal{F}} \mathcal{Z}(f) = \min_{C\in\mathcal{X}} v(C) \tag{1.12}$$

où $\mathcal{Z}(f)$ et $v(C)$ désignent respectivement la valeur du flux f et la valeur de la coupe C.

Démonstration. Des équations (1.3), (1.5) et (1.6) découlent que pour tout flux f et toute coupe C, $\mathcal{Z}(f) \leq v(C)$. Le théorème est donc triviale pour $\mathcal{Z}(f) = \infty$. Supposons donc que $\mathcal{Z}(f)$ soit finie. Dans ce cas, si nous établissons qu'il existe un flux f et une coupe $C = (X, \bar{X})$ telle que $\mathcal{Z}(f) = v(C)$, alors ce flux sera maximal et cette coupe minimum.

Soient f un flux maximal, X et $\bar{X}$ deux sous ensembles de N définis par les trois conditions suivantes :

1. $s \in X$;
2. si $i \in X$ et $f(i,j) < q(i,j)$, alors $j \in X$ (arc avant);
3. si $i \in X$ et $f(j,i) > 0$, alors $j \in X$ (arc arrière).

Montrons que $(X, \bar{X})$ est une coupe du réseau, c'est à dire que $t \in \bar{X}$. Si on avait $t \in X$, alors il existerait un chemin $s = n_0, n_1, n_2, ..., n_l = t$ dans X qui relie s à t. Pour tout $k \in [\![1, l]\!]$, (n_k, n_{k+1}) est soit un arc avant, soit un arc arrière. Notons F l'ensemble des arcs avants et B l'ensemble des arcs arrières du chemin $s = n_0, n_1, n_2, ..., n_l = t$. On définit :

$$\varepsilon_1 = \min_{(i,j)\in F} q(i,j) - f(i,j) \quad \text{et} \quad \varepsilon_2 = \min_{(i,j)\in B} f(i,j).$$

Soit $\varepsilon = \min(\varepsilon_1, \varepsilon_2)$. On a $\varepsilon > 0$. Modifions maintenant notre flux f en ajoutant ε aux valeurs des flux de tous les arcs avants et en soustrayant ε aux valeurs des flux de tous les arcs arrières. Nous obtenons un nouveau flux f' qui, tel qu'il est défini, obéit aux lois de conservation des flux et aux contraintes de capacités. Montrons que sa valeur est égale à $\mathcal{Z}(f) + \varepsilon$. On a

$$\begin{aligned}
\mathcal{Z}(f') &= f'(s, n_0) + \sum_{(s,i)\in A\setminus(s,n_0)} f'(s,i) \\
\mathcal{Z}(f') &= f(s, n_0) + \varepsilon + \sum_{(s,i)\in A\setminus(s,n_0)} f(s,i) \\
\mathcal{Z}(f') &= \mathcal{Z}(f) + \varepsilon
\end{aligned}$$

Ceci est en contradiction avec l'hypothèse selon laquelle le flux f à une valeur maximale. Donc $t \in \bar{X}$. Par conséquent, selon la définition de notre ensemble X on a :

- un arc (i,j) avec $i \in X, j \in \bar{X}$ remplit $f(i,j) = q(i,j)$

- un arc (i, j) avec $i \in \bar{X}, j \in X$ remplit $f(i, j) = 0$

Donc $\sum_{(i,j)\in\delta^-(X)} f(i, j) = 0$. Par conséquent,

$$\mathcal{Z}(f) = v(C) \qquad \square$$

Remarque. La coupe C induite par $(X, \bar{X})$ dans le réseau $\mathcal{N} = (N, A, q)$ offre aussi un moyen de construire une couverture de sommets dans le graphe biparti sous-adjacent $G = (U, V; E)$, Rappelons que l'ensemble des nœuds de $\mathcal{N}$ est $N = U \cup V \cup \{s, t\}$, tandis que son ensemble d'arcs A est produit par les arêtes de E dirigé de U vers V, plus les arcs (s, i) et (j, t) avec $i \in U$ et $j \in V$ (tous de capacité 1).

Proposition 1.34. *Soit $G = (U, V; E)$ un graphe biparti, $\mathcal{N}$ le réseau correspondant à G et $C = (X, \bar{X})$ une coupe de $\mathcal{N}$. Une couverture de sommets de G est donnée par les sommets $i \in U \cap \bar{X}$ et $j \in V \cap X$.*

Exemple. Considérons le graphe biparti donnée dans la Figure 1.10. La coupe minimale du réseau correspondant, représentée à la Figure 1.11, est donnée par $X = \{s, b, c, d, b\}$ et $\bar{X} = \{a, a, c, d, t\}$. L'ensemble $\bar{X}$ ne contient qu'un seul sommet de U, à savoir, a; l'ensemble X ne contient qu'un seul sommet de V, à savoir, b. Ainsi, une couverture de sommets est donnée par les sommets a et b.

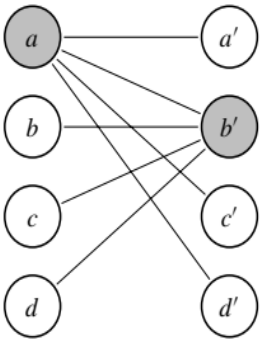


Figure 1.10. Couverture de sommets dans l'exemple ci dessus.

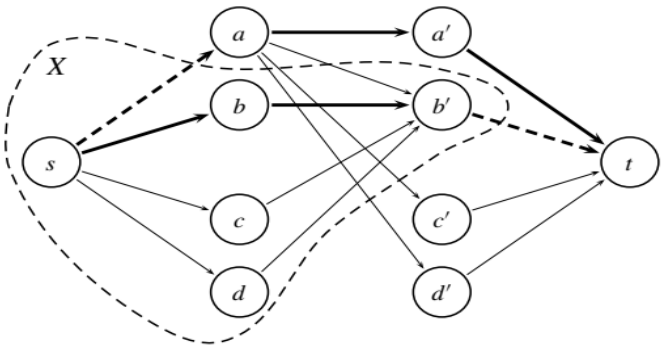


Figure 1.11. Un flux maximal et une coupe minimum dans le réseau de l'exemple ci dessus.

1.5 Algorithmes de couplage dans les graphes bipartis

Dans les sections précédentes, nous avons abordé les couplages dans les graphes bipartis. Nous avons montré, à travers quelques théorèmes, les conditions d'existence de couplage maximum et de couplage parfait dans un graphe biparti. Nous allons à présent étudier deux méthodes efficaces pour trouver un couplage maximum dans un graphe biparti. Sauf indication contraire, tout au long de cette section, nous supposerons, que $n = |U| \leqslant |V|$ et $m = |E|$.

1.5.1 Algorithme de couplage par étiquetage.

Définition 1.35. (arête active, arête passive, chemin alternatif) *Soient $G = (U, V; E)$ un graphe biparti et un couplage M de G (M peut être vide).*

*Une arête de E est dite **active** si elle appartient à M, elle est dite **passive** dans le cas contraire.*

*Un chemin dont les arêtes sont alternativement actives et passives est appelé **chemin alternatif**.*

Définition 1.36. (chemin d'augmentation) *Un chemin alternatif $i_1, j_1, i_2, j_2, ..., i_k, j_k$ avec $i_1, i_2, ..., i_k \in U$ et $j_1, j_2, ..., j_k \in V$ est appelé un **chemin d'augmentation** P pour le couplage M si les sommets i_1 et j_k sont non couplés (c'est-à-dire qu'aucune arête active ne rencontre ces sommets).*

Remarque. Étant donné que le premier et le dernier sommet d'un chemin d'augmentation sont non couplés, un chemin d'augmentation commence et se termine par une arête passive. Chaque chemin d'augmentation i_1, j_1, i_2, j_2, ..., i_k, j_k a une longueur impaire, c'est à dire, il contient k arêtes passives et $k-1$ arêtes actives. Une seule arête passive dont les points d'extrémités ne sont pas couplés est un chemin d'augmentation de longueur 1.

Le nom « chemin d'augmentation » découle de l'opération d'augmentation décrite dans la Définition 1.37 ci-dessous. Nous rappelons que la différence symétrique de deux ensembles A et B est définie par :

$$A \ominus B := (A \setminus B) \cup (B \setminus A).$$

Définition 1.37. (augmentation d'un couplage) *Soient M un couplage et P un chemin d'augmentation par rapport au couplage M. On appelle couplage augmenté de P l'ensemble d'arêtes $M \ominus P$.*

Remarque. Augmenter un couplage M par un chemin d'augmentation P revient à :

- changer le rôle des arêtes actives et passives dans P.
- maintenir actives toutes les arêtes actives qui ne se trouvent pas dans P.

Lemme 1.38. (Berge[1]) *Si M n'est pas un couplage maximum dans G, alors il existe un chemin d'augmentation P par rapport à M, et $M' = M \ominus P$ est un couplage dans G et $|M'| = |M| + 1$.*

Démonstration. Soient M un couplage de G donné et $\bar{M}$ n'importe quel couplage maximum de G. Si M n'est pas maximum, alors la différence symétrique n'est pas vide et ne peut pas être composée de cycle ou de chemin de longueur paire, Puisque $\bar{M}$ a plus d'arêtes que M. Ainsi, il doit exister un chemin P, inclus dans la différence symétrique, de longueur impaire qui commence et se termine par une arête de $\bar{M}$. Il s'agit d'un chemin d'augmentation par rapport au couplage donné M. En raison de la définition d'un chemin d'augmentation, il est simple de voir que $M' = M \ominus P$ est encore un couplage. En effet, chaque arête active de P est adjacente à une arête passive de P. Ainsi, $(P \setminus M)$ est un ensemble d'arêtes deux à deux non adjacentes. De même que $M \setminus P$. Par conséquent, $M' = M \ominus P$ est un ensemble d'arêtes deux à deux non adjacentes. Donc un couplage. Étant donné que $k-1$ arêtes de M sont échangées contre k arêtes n'appartenant pas à M, le nouveau couplage M' a la taille $|M| + 1$. □

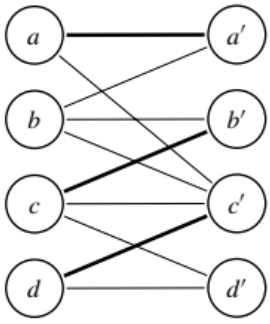


Figure 1.12. Couplage M (arêtes en gras).

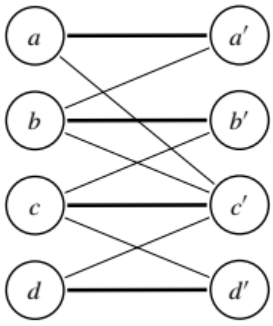


Figure 1.13. Couplage augmenté pour $P = (b, b', c, c', d, d')$.

Corollaire 1.39. *Un couplage M est maximum si et seulement s'il n'existe pas de chemin d'augmentation par rapport à M.*

Remarque. Du Lemme 1.38 et du Corollaire 1.39 découlent un algorithme d'étiquetage des sommets du graphe permettant d'obtenir un coulage maximum. L'algorithme démarre par un couplage arbitraire M de $G = (U, V; E)$ et procède par augmentation étape par étape en cherchant des chemins jusqu'à ce qu'un couplage maximum soit atteint.

Soit L l'ensemble des sommets non couplés de U (côté gauche). Les sommets de V étiquetés sont collectionnés dans l'ensemble R (côté droit). Initialement $R := \emptyset$.

Nous commençons par un $i \in L$ sur le côté gauche, Nous étiquetons comme successeur de i tous les voisins non étiquetés $\{l(j) = x, j \in N(i)\}$ et les ajoutons dans R.

- Si un sommet étiqueté j du côté droit est **non couplé**, alors nous avons trouvé un chemin d'augmentation P et on obtient un nouveau couplage $M \ominus P$ avec une arête de plus.

- Autrement, on supprime j de R : j est **couplé** par l'arête $(\bar{i}, j)$. Ensuite, le sommet $\bar{i}$ est étiqueté comme prédécesseur de j $\{r(\bar{i}) = x\}$ et ajouté à L.

Maintenant nous essayons de former le chemin d'augmentation du sommet nouvellement étiqueté $\bar{i}$. Dans ce cas soit nous trouvons un chemin d'augmentation ou nous concluons qu'un tel chemin n'existe pas. Dans ce dernier cas le couplage est déjà maximum.

Algorithme 1.1

Cardinality_Matching
Soit M un couplage dans $G = (U, V; E)$ ($M = \emptyset$ possible);
$L := \{i \in U : (i, j) \notin M, \forall j \in N(i)\}$;
$R := \emptyset$;
while $L \cup R \neq \emptyset$ **do**
  $x \in L \cup R$;
  if $x \in L$ **then :** scan_leftvertex(x) **else :** scan_rightvertex(x) ;
end while;
procédure scan_leftvertex(x)
  $L := L \setminus \{x\}$;
  while il existe une arête (x, j) avec j non étiqueté **do :**
    $l(j) := x$;
    $R := R \cup \{j\}$
  end while
procédure scan_rightvertex(x)
  $R := R \setminus \{x\}$;
  if il existe une arête $(i, x) \in M$ **then:**
    $r(i) = x$;
    $L := L \cup \{i\}$;
  else:
    à partir de x, trouvez le chemin alternatif P en faisant un retour en arrière des étiquettes
    $P := (..., r(l(x)), l(x), x)$;
    $M := M \ominus P$;
    $L := \{i \in U : (i, j) \notin M, \forall j \in N(i)\}$
    $R := \emptyset$; $l := \emptyset$, $r := \emptyset$
  end if

Description de l'algorithme. L'algorithme fonctionne de la manière suivante :

1. Il choisit un sommet x dans $L \cup R$.
   - S'il s'avère que $x \in L$:
     → il commence par supprimer x de L;
     → il range toutes les arêtes (x, j) tel que j non étiqueté dans un ensemble l (étiqueter);
     → il ajoute ces sommets j dans R.

– S'il s'avère que $x \in R$:

→ il retire x de R et deux cas de figure se propose à lui :

- Si x est déjà couplé, alors il ajoute l'arête $(i, x) \in M$ dans l'ensemble r (étiqueter) et ajoute i dans L.
- Sinon, il cherche un chemin d'augmentation en parcourant alternativement les ensembles l et r. Après parcours, il augmente le couplage M et range dans L les sommets de U non couplés. Les autres ensembles sont initialisés.

  La recherche du chemin commençant par x se fait comme suit : Il prend dans l l'arête qui à pour extrémité x et l'ajoute à P, nommons l'autre extrémité x_1. Ensuite il sélectionne dans r l'arête qui a pour extrémité x_1 et la range dans P, nommons l'autre extrémité x_2. Il continue le processus jusqu'à ce qu'il ne trouve plus de chemin. Notre chemin P, ainsi construit, est un chemin d'augmentation. Car P est alternatif puisque l'ensemble r, respectivement l, tel qu'il est défini étiquette les arêtes actives, respectivement passives. Et P commence par une arête de l et ce termine par une arête de l.

  A chaque fois que nous étiquetons un sommet j dans la procédure scan_leftvertex, ce sommet ne sera plus étiqueté jusqu'à ce qu'un chemin d'augmentation soit trouvé. Donc on est assuré qu'à partir d'un certain moment il ne restera que des sommets de V non couplé dans R. A cet instant, le prochain appel de la procédure scan_rightvertex trouvera un chemin d'augmentation.

– Notre algorithme ne s'arrête que si $L \cup R$ devient vide.

Exemple. Considérons le graphe biparti donné dans la Figure 1.14.

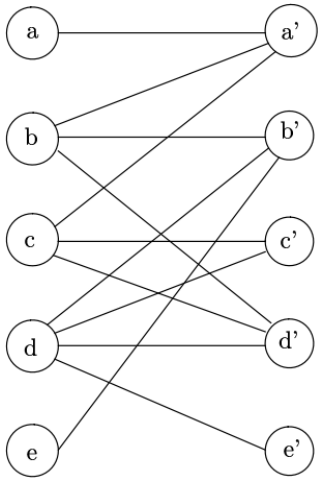


Figure 1.14. Graphe biparti pour l'exemple 2.35

Nous commençons par un couplage $M = \emptyset$. $L = \{a, b, c, d, e\}$ est composé de tous les sommets de U. On choisit un $a \in U$; il a un voisin a' non étiqueté. Par conséquent, $l(a') = a$. Maintenant on choisit $a' \in R$ et on obtient un chemin d'augmentation $P = (a, a')$. Ainsi l'arête (a, a') devient active et on continue avec le même processus en couplant les sommets b, c et d aux sommets b', c' et d' respectivement.

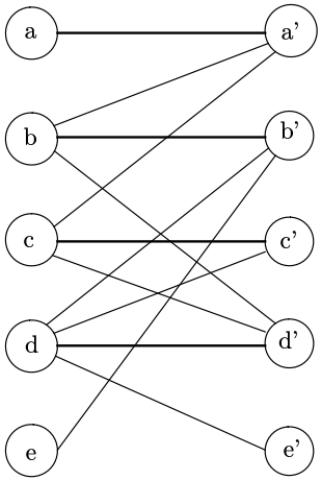


Figure 1.15. $M = \{(a, a'), (b, b'), (c, c'), (d, d')\}$

Maintenant, on a $L = \{e\}$,$R = \emptyset$ et l'étiquette $l(b') = e$. Donc $R = \{b'\}$ et b est étiqueté par b': $r(b) = b'$, $L = \{b\}$. Donc, on continue avec le sommet b et l'étiquette $l(a') = b$, $l(d') = b$. Nous obtenons $R = (a', d')$. Si nous choisissons a' puis d', nous obtenons $r(a) = a'$, $r(d) = d'$ et $L = \{a, d\}$. Nous continuons avec $a \in L$, mais ne pouvons pas trouver une arête passive commençant en a. Alors, d est choisi et nous étiquetons $l(c') = d$, $l(e') = d$. Ainsi $R = \{c', e'\}$. Si nous sélectionnons maintenant e, nous avons trouvé un chemin d'augmentation P qui peut être récupéré en suivant les étiquettes commençant par e. On obtient $P = \{(e, b'), (b', b), (b, d'), (d', d), (d, e')\}$ et l'augmentation conduit au couplage $M = \{(a, a'), (b, d'), (c, c'), (d, e'), (e, b')\}$. Puisque maintenant $L = \emptyset$, nous avons terminé. M est un couplage maximum.

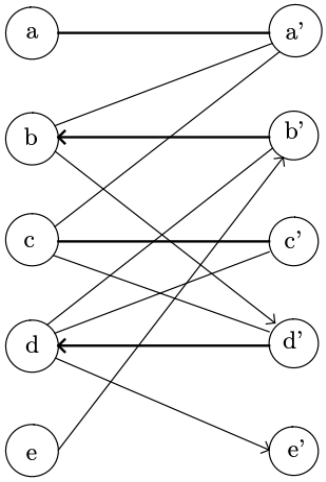


Figure 1.16. Chemin d'augmentation P

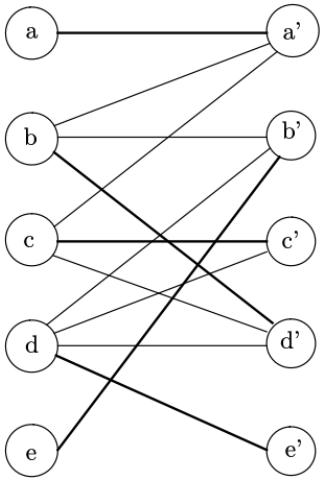


Figure 1.17. Couplage maximum M

Proposition 1.40. *La complexité de l'algorithme est* $\mathcal{O}(nm)$.

Démonstration. À chaque augmentation, un sommet supplémentaire de U est couplé. Il y a donc au plus n augmentations. De plus, chaque sommet est étiqueté au plus une fois par augmentation. Par conséquent, la recherche d'un chemin d'augmentation nécessite au plus $\mathcal{O}(m)$ étapes, et cet algorithme trouve un couplage maximum en $\mathcal{O}(nm)$. □

L'algorithme d'étiquetage discuté ci-dessus peut également être interprété en termes de flux de réseau. Nous intégrons G dans un réseau $\mathcal{N} = (N, A, q)$. La source est connectée à chaque nœud de U par un arc de capacité 1, chaque nœud de V est connecté à la cible par un arc de capacité 1, et chaque arc de E est dirigé de U à V et alimenté d'une capacité infinie (voir , par exemple, le graphe de la Figure 1.11). Comme nous le savons, un couplage maximum dans G correspond à un flux maximal dans le réseau $\mathcal{N}$.

Soit f un flux arbitraire dans $\mathcal{N}$. Nous définissons le réseau incrémental $\mathcal{N}_f$ par rapport au flux f comme suit.

Définition 1.41. (réseau incrémental) *Le réseau incrémental par rapport à* f *dans le réseau* $\mathcal{N}$ *est le réseau* $\mathcal{N}_f = (N, A_f, q_f)$ *qui a deux types d'arcs :*

- *Des arcs avants* (i,j) *si* $f(i,j) < q(i,j)$ *de capacité* $q_f(i,j) = q(i,j) - f(i,j)$*;*
- *Des arcs arrières* (j,i) *si* $f(i,j) > 0$ *de capacité* $q_f(j,i) = f(i,j)$.

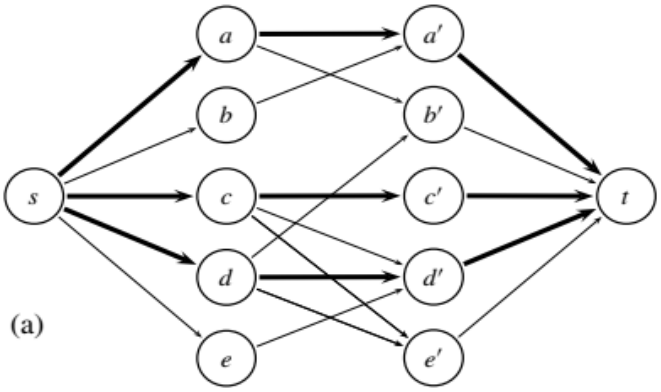
La Figure 1.18 montre un flux (arcs en gras), le réseau incrémental correspondant et les arcs qui sont pertinents pour trouver un flux maximum. Un chemin dirigé dans $\mathcal{N}_f$ de la source s à la cible t est à nouveau appelé chemin d'augmentation. Le Théorème 1.33 de Ford-Fulkerson dit que si f est un flux maximum, alors chaque chemin de la source à la cible contient un arc (i,j) avec $f(i,j) = q(i,j)$. Cela signifie que si f est un flux maximum dans $\mathcal{N}$, alors le réseau incrémental $\mathcal{N}_f$ ne contient pas de chemin dirigé de s vers t, c'est-à-dire qu'il n'y a pas de chemin d'augmentation dans $\mathcal{N}_f$. Un flux Δf dans le réseau incrémental est appelé flux incrémental.

Définition 1.42. (flux augmenté) *Soit* f *un flux dans* $\mathcal{N}$ *et* Δf *un flux dans le réseau incrémental* $\mathcal{N}_f$*. Alors le flux augmenté* $f \ominus \Delta f$ *est défini par :*

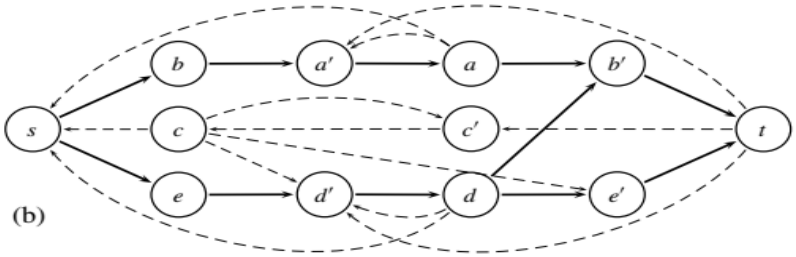
$$f \ominus \Delta f(i,j) = \begin{cases} f(i,j) + \Delta f(i,j) & \text{si } (i,j) \text{ est un arc avant} \\ f(i,j) - \Delta f(j,i) & \text{si } (i,j) \text{ est un arc arrière} \end{cases}$$

Lemme 1.43. *Soit* f *un flux dans le réseau* $\mathcal{N}$.

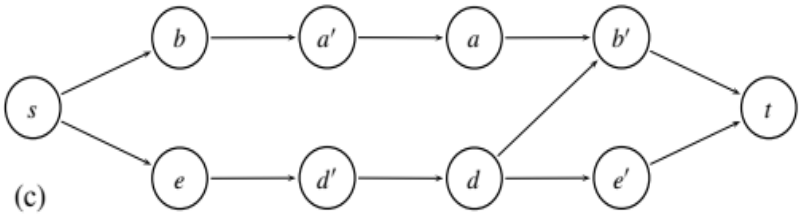
1. *Si* Δf *est un flux dans* $\mathcal{N}_f$ *alors* $f \ominus \Delta f(i,j)$ *est un flux dans* $\mathcal{N}$.
2. *pour tout flux* f *et* g *dans* $\mathcal{N}$ *dont les valeurs* $\mathcal{Z}(f)$ *et* $\mathcal{Z}(g)$ *remplissent* $\mathcal{Z}(f) > \mathcal{Z}(g)$*,*
   il existe un flux incrémental Δf *dans* $\mathcal{N}_f$ *tel que* $g = f \ominus \Delta f(i,j)$.



La figure (a) montre un flux dans un réseau provenant d'un problème de couplage maximum.



La figure (b) montre le réseau incrémental complet, y compris les arcs en pointillés qui ne se produisent jamais dans un chemin d'augmentation.



La figure (c) montre uniquement les arcs qui sont pertinents pour trouver un débit maximum.

Figure 1.18.

Les explications ci-dessus montrent qu'il existe une correspondance entre les chemins d'augmentation par rapport à un couplage M et les chemins d'augmentation dans le réseau incrémental correspondant $\mathcal{N}_f$. Il n'y a qu'une seule différence entre les étapes d'augmentation $M \ominus P$ et $f \ominus \Delta f$: dans ce dernier le flux f (le couplage M) est augmenté le long de plusieurs chemins d'augmentation en une seule étape.

Cette idée est utilisée dans la sous section suivante pour montrer que l'étiquetage peut être accélérée en créant plusieurs chemins d'augmentations en parallèle.

1.5.2 Algorithme de Hopcroft et Karp

L'idée de Hopcroft et Karp [10] était d'augmenter un couplage non pas par un chemin d'augmentation, mais par un système maximal de chemins d'augmentation à sommets disjoints de même longueur minimale.

Définition 1.44. *Soit $G=(U,V;E)$ un graphe biparti et M un couplage de G non maximum. Soit $\mathcal{P}$ l'ensemble des chemins d'augmentations de M. Le cardinal d'un chemin $P\in\mathcal{P}$, noté $|P|$, est le nombre d'arêtes que compte P.*

Un chemin d'augmentation $P\in\mathcal{P}$ est dite ***un plus court chemin d'augmentation****, si $|P|$ est minimal.*

Puisque la différence symétrique est une opération associative et commutative, on a

$$(M\ominus P_1)\ominus P_2=(M\ominus P_2)\ominus P_1$$

Si deux chemins d'augmentations P_1 et P_2 par rapport à M sont à sommets disjoints, l'ensemble $M'=(M\ominus P_1)\ominus P_2$ est encore un couplage. Nous pouvons donc augmenter M simultanément par P_1 et P_2 et obtenir un couplage plus large M'. Cela peut facilement être généralisé à k chemins à sommets disjoints.

Définition 1.45. *Soit $\Delta M=(P_1,P_2,...,P_k)$ un système de k chemins d'augmentations à sommets disjoints par paire par rapport au couplage M. Alors*

$$M\ominus\Delta M=(...((M\ominus P_1)\ominus P_2)...)\ominus P_k.$$

Puisque $\ominus$ est associative et commutative, la définition ne dépend pas de la numérotation des chemins. En particulier, puisque P_1, P_2, ..., P_k sont à sommets disjoints, on obtient

$$M\ominus\Delta M=M\ominus(P_1\cup P_2\cup...\cup P_k)$$

Pour construire un système de plus courts chemins d'augmentation nous introduisons la notion de graphe en couche.

Graphe en couche. Soit $G=(U,V;E)$ et M un couplage de G. Le graphe en couche LM par rapport à M est défini comme suit :

- → La première couche L_0 contient tous les sommets non couplés de l'ensemble U.
- → La deuxième couche L_1 contient les sommets de V qui sont des extrémités des arêtes (i,j) dans G avec $i\in L_0$. L'arête correspondante est collectée dans E_1.
- → Si L_1 contient un sommet non couplé de V, on arête et on nettoie le graphe en couche.
- → Autrement, chaque sommet $j\in L_1$ est couplé par une arête $(i,j)\in M$. Ces arêtes forment l'ensemble E_2. Les sommets i correspondants sont maintenant dans L_2. Donc $|L_1|=|L_2|$.
- → Maintenant, nous balayons toutes les arêtes (i,j) avec $i\in L_2$ et $j\notin L_1$ et les ajoutons à l'ensemble E_3. Les sommets v correspondants forment la couche L_3.
- → Nous arrêtons ce processus avec la première couche L_h, h impaire, qui contient soit un sommet $j\in V$ non couplé ou l'ensemble vide.
- → Dans le dernier cas il n'existe pas un chemin d'augmentation par rapport au couplage M donné, impliquant que le couplage M est maximum.

→ Sinon, nous commençons à nettoyer le graphe en couches : dans la dernière couche, tous les sommets couplés sont supprimés avec toutes les arêtes incidentes. Ainsi, dans la couche L_{h-1}, il peut maintenant y avoir des sommets à partir desquels aucune arête ne mène à un sommet dans la couche L_h. Encore une fois, tous ces sommets et leurs arêtes incidentes sont supprimés. Ce processus se poursuit jusqu'à la couche L_0.

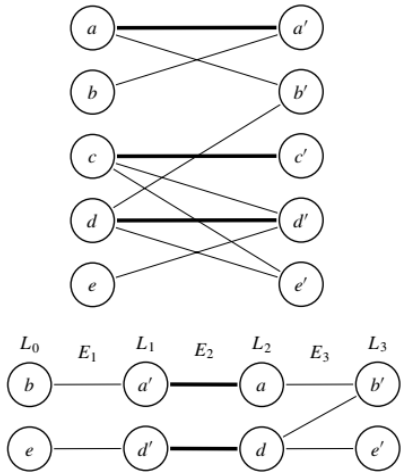


Figure 1.19. Graphe en couche

Remarque. De toute évidence, le graphe en couches nettoyé peut être construit en $\mathcal{O}(m)$ étapes car chaque arête est considérée au plus deux fois. Maintenant, nous orientons toutes les arêtes de la couche L_k vers la couche L_{k+1} $(k = 0, 1, ..., h-1)$. Il est à noter que chaque chemin depuis la première couche L_0 jusqu'au sommet de la dernière couche L_h utilise alternativement des arêtes passives et actives et a une longueur égale à h. Puisque h est impair et que chaque $j \in L_h$ est non couplé, chaque chemin de ce type est un chemin d'augmentation par rapport au couplage donné M. La longueur h est appelée la profondeur du graphe en couches.

Détermination d'un système maximal de chemins d'augmentation. La détermination d'un système maximal de chemins d'augmentation à sommets disjoints peut être fait en balayant les sommets du graphe en couches dans une première recherche en profondeur.

→ Nous commençons le premier chemin avec une arête arbitraire entre les deux premières couches et le continuons de couche en couche, jusqu'à ce qu'un sommet j dans la dernière couche L_h soit atteint. Ce chemin est le premier chemin d'augmentation P_1.

→ Pour chaque sommet v_k $(k = 0, 1, ..., H)$ sur le chemin P_1, nous supprimons tout arête n'appartenant pas à P_1 qui sont incidentes au sommet v_k.

→ Ainsi, certains sommets des couches L_1, ..., L_h peuvent maintenant n'avoir aucune arête entrant et certains sommets des couches L_0, ..., L_{h-1} peuvent ne pas avoir d'arêtes sortantes. Nous supprimons ces sommets et leurs arêtes incidentes. Cette étape est répétée autant de fois que possible.

→ Ensuite, nous recommençons, si possible, à partir d'un nouveau sommet dans la première couche pour obtenir un autre chemin d'augmentation.

Remarque. En balayant toutes arêtes du graphe en couches G une seule fois, un système maximal $\Delta\, M$ de chemins d'augmentation à sommets disjoints peut être trouvé.

Ainsi, une itération commence par la construction du réseau en couches et la détermination de $\Delta\, M$ et prend $\mathcal{O}(m)$ opérations . Alors que l'Algorithme 1.1 utilise des chemins d'augmentation en $\mathcal{O}(n)$ dans le pire des cas, le théorème suivant montre que nous n'avons que des augmentations en $\mathcal{O}(\sqrt{n})$ si nous utilisons des systèmes maximaux de plus courts chemins d'augmentation à sommets disjoints.

Théorème 1.46. (Hopcroft-Karp) *Soit $G=(U,V;E)$ un graphe biparti avec m arêtes et n sommets. Un couplage maximum dans G peut être trouvé en $\mathcal{O}\left(m\sqrt{n}\right)$ opérations.*

Démonstration. Nous allons prouver que le nombre d'itérations est borné par $\sqrt{n}$. Soit M' un couplage maximum dans G. A chaque itération la taille du couplage augmente d'au moins 1. Si $|M'|\leqslant\sqrt{n}$, alors il n'y aura rien à prouver. Supposons donc que $|M'|>\sqrt{n}$. Alors il existe une itération k' dans lequel $M_{k'}$ atteint la taille $|M'|-\sqrt{n}$. Après cette itérations il reste au moins $\sqrt{n}$ itération supplémentaire pour atteindre la taille du couplage maximum $|M'|$. Donc si nous prouvons que $M_{k'}$ a été obtenue en $\sqrt{n}$ itérations, alors le théorème sera prouvé. Nous prouverons ce fait en montrant que le graphe en couches à l'itération k' a une profondeur inférieure à $2\sqrt{n}+1$.

D'après le Lemme 1.38 de Berge, chaque chemin d'augmentation augmente le couplage de 1. Donc la différence symétrique de M' et $M_{k'}$ contient $\sqrt{n}$ chemins d'augmentations à sommets disjoints. Tous ces chemins ne peuvent pas avoir une longueur supérieure à $2\sqrt{n}+1$ car chaque chemin avec une telle longueur contient plus de $\sqrt{n}$ sommets de U. Puisqu'il y a plus de $\sqrt{n}$ chemins, cela signifierait que les chemins d'augmentation contiennent plus de n sommets de U, ce qui est impossible. Il existe donc au moins un chemin d'augmentation d'une longueur inférieure à $2\sqrt{n}+1$. Étant donné que le graphe en couches d'un couplage ne contient que les chemins d'augmentation les plus courts, la profondeur du graphe en couches du couplage $M_{k'}$ est inférieure à $2\sqrt{n}+1$, ce qui prouve le théorème. □

Algorithme 1.2

Soit $G=(U,V;E)$ un graphe biparti avec un couplage initial qui peut être nul;
Soit U_0 contenant tous les sommets non couplés de U;
Soit V_0 contenant tous les sommets non couplés de V;
while $U_0\neq\emptyset$ **do:**
    Construire_graphe_couches;
    Trouver_$\Delta\, M$;
    $M:=M\ominus\Delta\, M$;
    Mettre à jour les ensembles des sommets non couplés U_0 et V_0;
endwhile

Procédure Construire_graphe_couches
$L_0 := U_0, k^* := k := 0$
while $L_k \neq \emptyset$ **do:**
for $i \in L_k$ **do:**
$N(i) := \{j : [i, j] \in E \setminus M, j \notin L_1 \cup L_2 \cup ... \cup L_k\}$;
$L_{k+1} := \cup_{i \in L_k} N(i)$
if $L_{k+1} = \emptyset$ **then:** arrêter; [**commentaire:** M est un couplage maximum]
if $L_{k+1} \cap V_0 = \emptyset$ **then :**
$k^* := k + 1$;
$L_{k+2} = \emptyset$
else :
$L_{k+2} := \{\bar{i} : [\bar{i}, j] \in M, j \in L_{k+1}\}$
Endif
$k = k + 2$
Endwhile.
Procédure Trouver_ΔM
commentaire: trouver un ensemble maximal de chemins d'augmentations disjoints ΔM
while $L_0 \neq \emptyset$ **do:**
Choisir $x_0 = i \in L_0$ et le supprimer dans L_0;
$l := 0$;
while $l \geqslant 0$ **do:**
while x_l a un voisin non scanner dans L_{l+1} **do:**
choisir un voisin non scanner x_{l+1};
marquer x_{l+1} comme scanner;
$l := l + 1$;
$P_k := (x_0, x_1, ..., x_k)$;
$k := k + 1$;
Endwhile
if $l < k^*$ **then:** $l = l - 1$ **else:** $l = -1$
Endwhile
Endwhile
return $\Delta M := (P_1, P_2, ..., P_{k-1})$

Exemple. Nous illustrons cet algorithme sur le graphe de la Figure 1.19. Nous commençons avec le couplage vide $M = \emptyset$. Le premier graphe en couches coïncide avec le graphe $G = (U, V; E)$ de longueur $k^* = 1$. La procédure trouver_ΔM trouve un premier chemin $P_1 = (a, a')$. Le sommet a est supprimé dans L_0, et le sommet a' est scanné. Ensuite on trouve le chemin $P_2 = (b, b')$, le sommet b est supprimé et b' est scanné. De même on trouve $P_3 = (c, c')$ et $P_4 = (d, d')$. Finalement, le sommet $\{e\}$ n'a pas de voisin non couplé. Donc ΔM devient l'union des chemins P_1, P_2, P_3, P_4, l'étape d'augmentation donne, comme premier couplage non vide, $M = \{(a, a'), (b, b'), (c, c'), (d, d')\}$, et les ensembles U_0,V_0 deviennent $U_0 = \{e\}$ et $V_0 = \{e'\}$.

Le nouveau graphe en couches commence avec $L_0 = \{e\}$ et continue avec $L_1 = \{b'\}$, $L_2 = \{b\}$, $L_3 = \{a', d'\}$. Le voisin de a' est a; le voisin de d' est d. Donc $L_4 = \{a, d\}$. Le sommet a n'a pas de voisin et le sommet d a deux voisins, c' et e'. Donc $L_5 = \{c', e'\}$ et $k^* = 5$.

La procédure trouver_ΔM supprime en premier c' dans L_5 et commence avec $x_0 = e, x_1 = b', x_2 = b, x_3 = a', x_4 = a$ en développant un premier chemin d'augmentation. Puisque x_4 n'a pas de voisin, nous essayons avec x_3 qui n'a pas aussi de voisin. Nous venons à x_2, dont le seul voisin non scanner est d'. Nous obtenons $x_3 = d'$ et $x_4 = d$ et le seul voisin restant de x_4, à savoir, le sommet $x_5 = e'$. Ainsi, nous obtenons le seul chemin d'augmentation $\Delta M = P_1 = (e, b', b, d', d, e')$ de longueur 5. La nouvelle étape d'augmentation conduit à l'adéquation $M = \{(a, a), (b, d), (c, c), (d, e), (e, b)\}$. Puisque nous avons maintenant $U_0 = \emptyset$, nous avons terminé: M est un couplage maximum.

Chapitre 2

Etude d'un cas particulier du problème $(\mathcal{P})$

2.1 Introduction

Le problème d'optimisation combinatoire. L'objectif de ce mémoire est de proposer et d'implanter un algorithme ad-hoc efficace de résolution du problème d'optimisation combinatoire suivant :

$$(\mathcal{P}): \begin{cases} \min \quad \sum_{i=1}^{p} \sum_{j=1}^{n} c_{i,j} x_{i,j} \\ \forall i \in \{1, 2, ..., p\}, \sum_{j=1}^{n} x_{i,j} = 1, \\ \forall j \in \{1, 2, ..., n\}, \sum_{i=1}^{p} x_{i,j} \leqslant r, \\ X = (x_{i,j}) \in \mathcal{M}_{p,n}(\{0,1\}) \,. \end{cases} \tag{2.1}$$

dont la solution $X = (x_{i,j}) \in \mathcal{M}_{p,n}(\{0,1\})$ est la matrice d'attribution minimisant le coût total d'un marché subdivisé en p lots $L_1, L_2, ..., L_p$ auxquels ont postulé n candidats $C_1, C_2, ..., C_n$ avec c_{ij} le prix proposé au lot L_i par le candidat C_j avec la contrainte de ne permettre à un candidat que de gagner au plus r sur les p lots en compétition.

Interprétation fonctionnelle du problème $(\mathcal{P})$. Si nous désignons par le terme *« candidature »* l'action de postuler à un lot en proposant un prix, alors l'ensemble des candidatures peut être vu comme une relation de correspondance (au sens de la théorie des ensembles) entre l'ensemble des lots $L = \{L_1, L_2, ..., L_p\}$ et l'ensemble candidats $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ où chaque *candidature* d'un candidat C_j à un lot L_i est sanctionnée par un coût $c_{i,j}$. Soit $\Gamma: L \leftrightarrow \mathcal{C}$ une telle correspondance. Il est un fait qu'à partir d'une correspondance on peut extraire plusieurs fonctions. Notons $\mathcal{F}_{\leqslant r}$ l'ensemble des fonctions extraites de la correspondance Γ pour lesquelles chaque élément de $\mathcal{C}$ à un nombre d'antécédents inférieur ou égal à r. Ainsi notre problème consiste à choisir la fonction $f \in \mathcal{F}_{\leqslant r}$ qui minimise le coût total $\sum_{i \in L} c_{i,f(i)}$.

$$(\mathcal{P}_f) \begin{cases} \min \sum_{i \in L} c_{i,f(i)} \\ f \in \mathcal{F}_{\leqslant r} \end{cases} \tag{2.2}$$

Étude d'un cas particulier du problème ($\mathcal{P}_f$). Dans ce chapitre nous étudions le cas particulier le plus simple de notre problème ($\mathcal{P}_f$) dans l'optique d'en généraliser les techniques de résolution pour attaquer le problème ($\mathcal{P}_c$). Le cas particulier est le suivant

$$\begin{cases} \min \sum_{i \in L} c_{i,f(i)} \\ f \in \mathcal{F}_{=1} \end{cases} \tag{2.3}$$

avec $n = p$, où $\mathcal{F}_{=1}$ désigne l'ensemble des fonctions extraites de la correspondance Γ pour lesquelles chaque élément de $\mathcal{C}$ à exactement un antécédent dans L. C'est le cas pour lequel on contraint chaque candidat à gagner exactement un lot et qu'il y ait autant de candidats que de lots. Le modèle combinatoire correspondant est le suivant :

$$(\mathcal{P}_a): \begin{cases} \min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j} \\ \forall i \in [\![1, n]\!], \sum_{j=1}^{n} x_{i,j} = 1, \\ \forall j \in [\![1, n]\!], \sum_{i=1}^{n} x_{i,j} = 1, \\ \forall (i, j) \in [\![1, n]\!] \times [\![1, n]\!], x_{i,j} \in \{0, 1\}\,. \end{cases} \tag{2.4}$$

Ce modèle est celui du classique problème de somme linéaire d'affectation (PSLA).

Définition 2.1. (affectation) *Soit U et V deux ensembles finis de même cardinal. Une affectation φ est une application bijective de U dans V.*

Partant de cette définition, $\mathcal{F}_{=1}$ peut être vu comme l'ensemble des affectations extraites de la correspondance Γ.

D'un point de vue matriciel, le problème consiste à choisir une matrice de permutation $X = (x_{i,j}) \in \mathcal{M}_{n,n}(\{0, 1\})$ dont la somme des coûts correspondants est minimale.

Nous pouvons également interpréter le PSLA à travers un modèle de théorie des graphes. Désignons par $U = \{1, ..., n\}$ l'ensemble des indices de positions des lots de L et par $V = \{1, ..., n\}$ l'ensemble des indices de positions des candidats de $\mathcal{C}$. Soit $G = (U, V; E)$ un graphe biparti où E représente le graphe de la correspondance Γ, i.e, chaque correspondance d'un candidat C_j à un lot L_i est représentée par une arête $(i, j) \in E$. Résoudre le PSLA est donc équivalent à extraire un couplage parfait dans G dont la somme des coûts de ses arêtes est minimale.

Remarque. Aux vues des interprétations des deux problèmes, nous pouvons dire que le PSLA est une restriction de notre problème $\mathcal{P}$. Ainsi nous espérons que certains des algorithmes de résolutions des PSLA pourront se généraliser pour résoudre le problème $\mathcal{P}$.

2.2 Approche polyédrale de résolution du PSLA

L'approche polyédrale consiste à transformer le problème en un problème de programmation linéaire continue. En effet, le théorème de Birkhoff, combiné au théorème fondamental de la programmation linéaire montre que le PSLA est équivalent à sa relaxation continue. Cette approche permet également d'établir que, malgré cette équivalence, la célèbre méthode du simplexe reste inefficace sur le PSLA.

Définition 2.2. (matrice bistochastique) *Une matrice* $(x_{i,j})_{(i,j)\in[\![1,n]\!]\times[\![1,n]\!]}$ *est dite bistochastique si elle vérifie les conditions suivantes :*

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i \in [\![1,n]\!] \tag{2.5}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j \in [\![1,n]\!] \tag{2.6}$$

$$x_{ij} \geq 0, \forall i,j \in [\![1,n]\!] \tag{2.7}$$

Définition 2.3. (polytope d'affectation) *On appelle polytope d'affectation, noté* P_A*, l'ensemble des matrices bistochastiques.*

Remarque. On a :

$$P_A := \left\{ (x_{ij}) \in \mathbb{R}^{n^2} : \forall i \in [\![1,n]\!], \sum_{j=1}^{n} x_{ij} = 1; \forall j \in [\![1,n]\!], \sum_{i=1}^{n} x_{ij} = 1; \forall i,j \in [\![1,n]\!], x_{ij} \geq 0 \right\} \tag{2.8}$$

La représentation matricielle du polytope d'affectation est :

$$P_A := \{X \in \mathbb{R}^{n^2} : AX = B, X \geqslant 0\} \tag{2.9}$$

où

$$A = \left(\begin{array}{cccc|cccc|c|cccc} 1 & 1 & \dots & 1 & & & & & & & & \\ & & & & 1 & 1 & \dots & 1 & & & & \\ & & & & & & & & \dots & & & \\ & & & & & & & & & 1 & 1 & \dots & 1 \\ \hline 1 & & & & 1 & & & & & 1 & & \\ & 1 & & & & 1 & & & \dots & & 1 & \\ & & \dots & & & & \dots & & \dots & & & \dots \\ & & & 1 & & & & 1 & \dots & & & & 1 \end{array}\right) \tag{2.10}$$

et $B = \{1, ..., 1\}$ est le vecteur de longueur $2n$ engendré par les côtés droits des équations (2.5) et (2.6).

Remarque.

1. La matrice A peut être vue comme une matrice d'incidence sommet-arête d'un graphe biparti complet $K_{n,n} = (U, V; E)$ avec $|U| = |V|$. Les n premières lignes correspondent aux sommets de U et les n dernières lignes aux sommets de V. Chaque colonne correspond à une arête.
2. Le polytope d'affectation P_A est un polytope au sens géométrique du terme (il est fait de sommets, d'arêtes et de facettes).
3. Si M_φ désigne la matrice d'une permutation $\varphi \in S_n$ alors $M_\varphi \in P_A$.

Ainsi les matrices de permutations sont des éléments du polytope d'affectation. Le théorème de Birkhoff établie qu'elles sont exactement les sommets du polytope d'affectation.

Théorème 2.4. (Birkhoff[2]) *A chaque sommet du polytope d'affectation correspond une et une seule matrice de permutation.*

La démonstration du théorème de Birkhoff s'appuie sur le lemme élémentaire suivant :

Lemme 2.5. *Soit $R=(r_{ij})$ une matrice carrée de taille $(n\times n)$ définie comme suit :*

- $\forall i,j\in\{1,...,n\}$, $r_{ij}\geqslant 0$ *(les entrées de R sont positives);*
- $\forall i\in\{1,...,n\}$, $\sum_{j=1}^{n} r_{ij}=\alpha$ *(la somme des éléments de chaque ligne est égale à α);*
- $\forall j\in\{1,...,n\}$, $\sum_{i=1}^{n} r_{ij}=\alpha$ *(la somme des éléments de chaque colonne est égale à α).*

Pour tout $k\in\{0,1,...,n\}$, si R contient une sous matrice de taille $(k+1)\times(n-k)$ nulle, alors R est aussi une matrice nulle.

Démonstration. (du théorème de Birkhoff) Soit X une matrice bistochastique arbitraire. D'après le lemme précédent, la matrice ne contient pas de sous matrice nulle de taille $(k+1)\times(n-k)$ pour tout $k=1,2,...,n-1$. Car s'il en était ainsi, on aurait $X=0$ contredisant le fait que X soit bistochastique. Alors, selon le Théorème 1.27 de Frobenius, il existe une matrice de permutation $P_1=(p_{ij})$ ayant la propriété suivante : si $p_{ij}=1$, alors $x_{ij}>0$.

Soit λ_1 la plus petite des entrées positives de la matrice X pour lesquelles $p_{ij}=1$. Alors $X-\lambda_1 P_1$ est une matrice positive dont les sommes des lignes et des colonnes sont égales à $\alpha-\lambda_1$. On répète ce processus jusqu'à avoir une matrice R qui contient une sous matrice nulle de taille $(k+1)\times(n-k)$. On obtient donc :

$$X=\lambda_1 P_1+\lambda_2 P_2+...+\lambda_r P_r+R$$

avec des valeurs positives λ_1, λ_2, ..., λ_r et des matrices de permutations P_1, P_2, ..., P_r. La matrice R est positive, les sommes des lignes et colonnes sont égales à $\alpha=1-\lambda_1-\lambda_2-....-\lambda_r$. Puisque R contient une sous matrice nulle de taille $(k+1)\times(n-k)$, R est une matrice nulle. Donc, $\lambda_1+\lambda_2+....+\lambda_r=1$. Par conséquent, X est une combinaison convexe des matrices de permutations $P_1,P_2,...,P_r$. □

Tout élément d'un polytope est une combinaison convexe des sommets de ce polytope. Plus précisément, tout élément d'un polytope P est une combinaison convexe d'au plus $\dim(P)+1$ sommets de ce polytope où $\dim(P)$ est la dimension du polytope. Ainsi, on déduit du théorème de Birkhoff que chaque matrice bistochastique peut être écrite comme une combinaison convexe d'au plus $\dim(P_A)+1$ matrices de permutation choisies parmi les $n!$ que contient P_A.

La dimension du polytope d'affectation, notée $\dim(P_A)$, est la dimension du plus petit sous espace affine de $\mathbb{R}^{n^2}$contenant P_A.

Géométriquement, chaque équation du système $AX = B$ correspond à un hyperplan dans $\mathbb{R}^{n^2}$ de dimension $n^2 - 1$ et l'intersection de k hyperplans indépendants dans $\mathbb{R}^{n^2}$ est un sous espace affine de dimension $n^2 - k$. Comme le nombre d'hyperplans indépendants du système $AX = B$ est égale au rang de la matrice A, on a :

$$\dim(P_A) = n^2 - \text{rang}(A). \tag{2.11}$$

La somme des n premières lignes de A est égale à la somme des n dernières lignes. Donc, $\text{rang}(A) < 2n$. L'on peut voir que les $2n - 1$ lignes sont linéairement indépendants : Si on supprime la dernière ligne, on verra que les n premières colonnes forment, ensemble avec les $k\,n+1$ colonnes $\forall k = (1, ..., n-1)$, un système de $(2n-1)$ vecteurs colonnes linéairement indépendants. Par conséquent, $\text{rang}(A) = 2\ n - 1$. Ainsi, on a le résultat suivant.

Proposition 2.6. *Le polytope d'affectation P_A est de dimension $(n-1)^2$ et toute matrice bistochastique peut être écrite comme une combinaison convexe d'au plus $(n-1)^2+1$ matrices de permutations.*

La proposition suivante établie un lien entre les sous familles de colonnes de la matrice A et les sous-graphe de $K_{n,n}$.

Proposition 2.7. *Chaque sous-ensemble de colonnes linéairement indépendantes de la matrice A correspond à un sous-graphe G de $K_{n,n}$ qui ne contient aucun cycle.*

Démonstration. Nous allons montrer que les colonnes sont linéairement dépendantes si le sous-graphe correspondant G contient un cycle et vice versa.

1. **Supposons que G contient un cycle**. Puisque G est biparti, le cycle a un nombre pair d'arête que l'on va colorier en rouge et bleu de manière alterner. Soit $a_{[ij]}$ désignant la matrice de A correspondant à l'arête (i, j). On a

$$\sum_{(i,j)\text{bleu}} a_{[ij]} = \sum_{(i,j)\ \text{rouge}} a_{[ij]}$$

puisque chaque sommet du cycle coïncide avec une arête bleu et rouge. Cela montre que les vecteurs colonnes correspondants sont dépendants.

2. **Soit $\{a_{[ij]} : (i, j) \in D\}$ un ensemble de colonnes linéairement dépendantes de la matrice A.** Alors il existe des coefficients $a_{[ij]}$, pas tous nulles, telle que

$$\sum_{(i,j)\in D} a_{[ij]} = 0$$

Soit $D' = \{(i, j) \in D : a_{[ij]} \neq 0\}$. Alors chaque sommet qui coïncide avec une arête de D' doit aussi coïncider avec une autre arête de D'. Ainsi, à partir d'une arête arbitraire dans D, on peut former une séquence infinie d'arêtes où le point final d'une arête est le point de départ de la suivante. Puisque $|D'| < \infty$, cette séquence doit avoir un cycle. □

D'après la proposition 2.7, il existe une correspondance une à une entre l'ensemble des sous-graphes de $K_{n,n}$ qui ne contiennent aucun cycle et l'ensemble des sous matrices de A avec $2n-1$ colonnes linéairement indépendantes. Cette correspondance joue un rôle déterminant dans la résolution des problèmes d'affectation linéaire par des techniques de programmation linéaire.

Définition 2.8. (base, solution de base) *Soient A une matrice de taille $m \times n$ de rang m et $P = \{x \in \mathbb{R}^n : A\,x = b, x \geq 0\}$. Considérons le programme linéaire de la forme :*

$$\min \{c'\,x : A\,x = b, x \geqslant 0\} \tag{2.12}$$

base. *Un sous-ensemble B de m indices correspondant à m colonnes linéairement indépendantes de A est appelé une base. Les composants x_j du vecteur x pour lesquels j est une colonne de B forme le vecteur x_B.*

matrice de base. *Les colonnes d'une base peuvent être jointes pour former la matrice de base A_B.*

Solution de base. *Une matrice de base est inversible. Soit x_B la solution du système d'équations $A_B\,x_B = b$. Une solution du système $Ax = b$ formée à partir de x_B et $x_j = 0$ pour $j \notin B$ est dite solution de base.*

base faisable. *La base est dite faisable si la solution de base vérifie $x_B \geqslant 0$.*

solution faisable. *La solution de base est dite faisable si la base est faisable. Une solution faisable correspond à un sommet du polytope P.*

Solution dégénérée. *Différentes bases peuvent correspondre à la même solution de base faisable donc au même sommet du polytope P. Dans ce cas la solution est dite dégénérée.*

Le théorème principal de la programmation linéaire stipule que si un programme linéaire atteint une solution optimale finie, alors il existe un sommet du polytope engendré par l'ensemble des contraintes sur lequel l'optimum est atteint. Ce théorème combiné à celui de Birkhoff nous permet de relâcher les contraintes du problème de somme linéaire d'affectations

$$x_{ij} = \{0, 1\} \text{ à } 0 \leqslant x_{ij} \leqslant 1, \forall i, j \in \{1, 2, ..., n\} \tag{2.13}$$

Puisque chaque solution de base faisable du programme linéaire

$$(\mathcal{P}_l): \begin{cases} \min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j} \\ \forall i \in [\![1, n]\!], \sum_{j=1}^{n} x_{i,j} = 1, \\ \forall j \in [\![1, n]\!], \sum_{i=1}^{n} x_{i,j} = 1, \\ \forall (i, j) \in [\![1, n]\!] \times [\![1, n]\!], x_{i,j} \in [0, 1]\,. \end{cases}$$

correspond à un sommet du polytope d'affectation engendré par l'ensemble des contraintes, c'est-à-dire à une matrice de permutation, le problème de somme linéaire d'affectation est équivalent à sa relaxation continue. Par conséquent, il peut être résolu via des techniques de programmation linéaire.

2.2.1 Techniques classiques de résolution de programme linéaire

Méthode de la recherche exhaustive. Par le théorème principal de la programmation linéaire, la solution du problème $(\mathcal{P}_l)$ correspond à un sommet du polytope engendré par l'ensemble des contraintes. On pourrait alors penser qu'il suffit de déterminer les coordonnées de tous les sommets du polytope et calculer la valeur de la fonction objectif en chacun d'eux puis choisir la plus grande d'entre elles. Mais sachant que le polytope en question a $n!$ sommets, un algorithme fondé sur le parcours exhaustif des sommets du polytope ne saurait résoudre le problème $(\mathcal{P}_l)$ en un temps raisonnable pour peu que la valeur de n soit légèrement élevée.

Méthode du simplexe. L'algorithme dit du simplexe est le plus connu pour la résolution des programmes linéaires. Celui-ci, au lieu de calculer la valeur de la fonction objectif pour tous les sommets, la calcule seulement pour une suite de sommets convenablement choisis : disposant d'un sommet du polytope comme point de départ, une itération permet de passer d'un sommet M à un de ses sommets voisins, en lequel la valeur de la fonction objectif est plus petite. Lorsqu'on atteint un sommet Q pour lequel la valeur de la fonction objectif n'est plus petite en aucun des sommets voisins alors l'algorithme s'arrête et le sommet Q est la solution du problème $(\mathcal{P}_l)$.

Remarque. L'algorithme du simplexe présente de très bonnes performances dans la pratique malgré une complexité exponentielle qui le classe dans la catégorie des algorithmes inefficaces. Aussi, si les solutions de bases sont dégénérées, alors l'algorithme du simplexe peut ne pas converger dans la mesure où il peut y avoir bouclage, c'est-à-dire qu'au cours de l'algorithme on rencontre un sommet déjà rencontré.

La section suivante établit que les solutions de bases du PSLA sont hautement dégénérées. Ce qui disqualifie la méthode du simplexe comme technique de résolution efficace du PSLA.

2.2.2 Dégénérescence de solutions de bases du PSLA

Les propositions suivantes, dues à Balinski et Russakoff [14], nous donne le nombre exact de solutions de bases qui correspondent à un sommet du polytope d'affectation. Tout d'abord, nous avons besoin de la définition suivante et d'un résultat de Cayley sur le nombre d'arbres couvrants différents dans un graphe biparti complet $K_n = (U, V; E)$ avec $|U| = |V| = n$.

Définition 2.9. (arbre couvrant) *Un arbre couvrant T d'un graphe G est un sous graphe de G qui est connexe et qui ne possède pas de cycle.*

Théorème 2.10. (Cayley [4]) *Le graphe biparti complet K_n à n^{n-2} arbre couvrants.*

Considérons le PSLA $(\mathcal{P}_a)$ défini ci-dessus et le graphe $K_{n,n}$ sous adjacent au problème. La proposition suivante fournit une illustration graphique des solutions de bases théoriques des problèmes de somme linéaire d'affectation.

Proposition 2.11. *A chaque base de la matrice A du polytope P_A correspond un et un seul arbre couvrant du graphe biparti complet $K_{n,n}$ sous adjacent.*

Chaque base faisable correspond à un arbre couvrant qui contient un couplage parfait de $K_{n,n}$.

Avec l'aide du théorème de Cayley, Balinski et Russakoff ont compté le nombre de bases qui correspondent à un sommet du polytope d'affectation.

Proposition 2.12. *Chaque sommet du polytope d'affectation correspond à $2^{n-1}\, n^{n-2}$ bases faisables.*

Ainsi, les solutions de bases du PSLA sont **fortement** dégénérées. En effet à chaque sommet du polytope d'affectation correspond un nombre exponentiellement élevé de solutions. Donc l'utilisation de la méthode du simplexe pour résoudre le PSLA peut conduire à des bouclages. Ainsi, Il est pertinent de rechercher d'autres méthodes de résolutions plus efficaces.

2.3 Algorithmes de résolution du PSLA

Les algorithmes de résolution du PSLA sont basés sur différentes approches : la première classe de méthodes résout directement le problème primal, une seconde résout le problème dual et une troisième utilise une approche (primal-dual). Dans la suite, nous allons étudier l'algorithme Hongrois qui utilise l'approche primal-dual. Pour approfondir cette approche ainsi que les autres approches le lecteur peut consulter [3].

2.3.1 Relâchement complémentaire

Considérons la version duale du problème du problème $\mathcal{P}_a$

$$\max \{b^t y : A^t y \leqslant C\,,\, y \in \mathcal{M}_{2n,1}(\{0,1\})\} \tag{2.14}$$

En posant $u_i = y_i$ et $v_j = y_{n+j}$ $\forall i, j \in \{1, ..., n\}$, on obtient

$$\max \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \tag{2.15}$$

$$\text{sc} \quad u_i + v_j \leqslant c_{ij} \tag{2.16}$$

D'après la théorie de la dualité, une paire de solution faisable respectivement pour le primal et le dual est optimale si et seulement si

$$x_{ij}\,(c_{ij} - u_i - v_j) = 0\,, \forall i, j \in \{1, 2, ..., n\} \tag{2.17}$$

les valeurs

$$\bar{c}_{ij} = c_{ij} - u_i - v_j, \forall i, j \in \{1, 2, ..., n\} \tag{2.18}$$

sont les coûts réduits du PSLA.

La plupart des méthodes de résolution du PSLA adopte une phase de pré-traitement pour déterminer une solution duale faisable et une solution primale partielle (où moins de n lignes sont affectées) satisfaisant la condition de relâchement complémentaire. Une implémentation basique de complexité $\mathcal{O}(n^2)$ de cette phase est donnée dans l'Algorithme 2.1, qui stocke les affectations partielles dans X et dans

$$\text{ligne}(j) = \begin{cases} i & \text{si } j \text{ est affecté à } i \\ & \\ 0 & \text{sinon} \end{cases}, \forall j \in \{1, ..., n\} \tag{2.19}$$

L'ensemble *ligne* tel qu'il est défini étiquette la colonne j comme antécédent de i qui lui est affectée. Notez que les coûts réduits donnés par les variables duales résultantes sont positives.

Algorithme 2.1

Procédure Basic_preprocessing
Initialiser la matrice X
for $i = 1$ to n **do:** **for** $j = 1$ to n **do:** $x_{ij} = 0$
Initialiser l'ensemble ligne
for $j = 1$ to n **do:** $\text{ligne}(j) = 0$
déterminer les variables dual
for $i = 1$ to n **do:** $u_i = \min\{c_{ij} : j = 1, ..., n\}$
for $j = 1$ to n: $v_j = \min\{c_{ij} - u_i : i = 1, ..., n\}$
Trouver une solution partielle faisable
for $j = 1$ to n **do:** $\text{ligne}(j) = 0$
for $i = 1$ to n **do:**
  for $j = 1$ to n **do:**
    if $\text{ligne}(j) = 0$ **and** $c_{ij} - u_i - v_j = 0$**:**
      $x_{ij} = 1$
      $\text{ligne}(j) = i$
      break
    endif
  endfor
endfor

L'algorithme est subdivisé en deux phases :

→ Dans la première phase il construit les solutions duales u et v comme suit :

- u_i contient le minimum de chaque ligne i de C ;
- v_j contient le minimum de la différence de chaque colonne j de C et de u.

→ Dans la deuxième phase il parcourt la matrice C. Si un élément de coût c_{ij} à un coût réduit nul et ligne$(j)=0$, alors il affecte i à j et étiquette j.

Les valeurs u_i et v_j des deux premières affirmations satisfont les contraintes du dual 2.16. Les valeurs x_{ij} obtenues par la suite garantissent la satisfaction des conditions de relâchement complémentaire. Tandis que pour les contraintes du primal, le signe $=$ est remplacé par $\leqslant$.

Dans la suite nous utiliserons fréquemment une affectation φ défini comme suit :

$$\varphi(i)=\begin{cases} j & \text{si } x_{ij}=1 \\ & \forall i\in\{1,2,...,n\} \\ 0 & \text{sinon} \end{cases} \tag{2.20}$$

Exemple 2.13. Étant donnée la matrice de coût C ci dessous. L'algorithme produit en premier lieu les solutions duales faisables u et v (montrés à gauche et au dessus de la matrice C). Ce qui donne la matrice de coût réduit $\bar{C}$ (représentée à droite)

$$\begin{array}{c} \\ 7 \\ 2 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 2 & 0 & 0 \end{array} \\ \begin{pmatrix} 7 & 9 & 8 & 9 \\ 2 & 8 & 5 & 7 \\ 1 & 6 & 6 & 9 \\ 3 & 6 & 2 & 2 \end{pmatrix} \\ C \end{array} \qquad \begin{array}{c} \begin{pmatrix} \underline{0} & 0 & 1 & 2 \\ 0 & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & \underline{0} & 0 \end{pmatrix} \\ \bar{C} \end{array}$$

Nous obtenons alors ligne$=(1,0,4,0)$ (donc $\varphi=(1,0,0,3)$) et l'affectation partielle montrée par les zéros soulignés dans $\bar{C}$.

2.3.2 L'algorithme Hongrois

L'algorithme Hongrois est reconnu comme un prédécesseur de la méthode primal-dual pour la programmation linéaire, conçu par **Dantzig, Ford, et Fulkerson[5]**. Il commence par une solution duale faisable u, v satisfaisant la contrainte 2.16 et une solution primale partielle (dans laquelle moins de n lignes sont affectées) satisfaisant la condition de relâchement complémentaire 2.17 par rapport à u et v. Chaque itération résout un problème primal restreint indépendant des coûts. Il essaye d'augmenter le cardinal de l'affectation courante en opérant sur le graphe partiel G^0 de G qui contient que les arêtes de E ayant des coûts réduits nuls. Si la tentative réussit une nouvelle solution primale, dans laquelle une ligne supplémentaire est affectée, est obtenue. Sinon la solution duale actuelle est mise à jour de sorte que de nouvelles arêtes ayant des coûts réduits nuls soient obtenues.

Afin de décrire l'algorithme, nous avons besoin de la définition suivante.

Définition 2.14.

Arbre enraciné. *Un arbre enraciné d'un graphe biparti complet G est un arbre, i.e un graphe connexe sans cycle, avec un sommet distingué comme racine. Les arcs sont orientés de telle sorte que tous les chemins de l'arbre mènent à la racine.*

arbre alternatif enraciné. *Un arbre alternatif enraciné en un sommet k est un arbre dans lequel tous les chemins émanant de k sont alternés.*

La restriction du problème de l'algorithme Hongrois consiste à chercher un chemin d'augmentation dans le graphe biparti partiel $G^0 = (U, V; E^0)$ où $E^0 = \{(i, j) \in E : \bar{c}_{ij} = 0\}$. Si un tel chemin P est trouvé, l'amélioration de l'affection est obtenu en inter-changeant les arêtes actives et passive le long de P.

La procédure de "Dijkstra-like" donnée dans l'Algorithme 2.2 (voir Dijkstra[6]) cherche un possible chemin d'augmentation partant d'un sommet non affecté $k \in U$ en construisant progressivement un arbre alternatif enraciné en k. Il est aussi étroitement lié à l'Algorithme 1.1 pour les couplages bipartis. A chaque itération chaque sommet est étiqueté (s'il appartient à un chemin émanant de k) ou non étiqueté. Un sommet étiqueté de V peut être numérisé (s'il a été utilisé pour étendre l'arbre) ou non numérisé. Dans cette implémentation l'étiquetage et la numérisation coïncident pour un sommet de U. L'ensemble $L\ V$ contient les sommets de V actuellement étiquetés tandis que l' ensemble SU, respectivement SV, contient les sommets de U, respectivement V, numérisés.

Algorithme 2.2

Procédure alternate(k)
SU = SV = LV = ∅
$fail$ = false, $sink = 0$, $i = k$
while $fail$ = false **and** $sink = 0$ **do:**
  SU = SU ∪ {i}
  for each $j \in V \setminus$ LV: $c_{ij} - u_i - v_j = 0$ **do:**
    $\text{pred}_j = i$, LV = LV ∪ {j}
  endfor
  if LV \ SV = ∅ **then:**
    $fail$ = true
  else:
    Soit $j \in$ LV \ SV
    if ligne(j) = 0 **then:** $sink = j$ **else:** i = ligne(j)
  endif
endwhile
return $sink$

L'exécution de l'algorithme se déroule en de deux phases.

→ Dans la première phase, l'unique sommet candidat $i \in U$ est en premier étiqueté et numérisé ($i = k$ pour la première itération). La numérisation consiste à étendre l'arbre en affectant i comme prédécesseur de tous les sommets $j \in V$ tel que $(i, j) \in E^0$ et étiquette ces sommets dans LV.

→ Dans la deuxième phase, un sommet étiqueté et non numérisé $j \in V$ est sélectionné et numérisé en ajoutant l'unique arête $(\text{ligne}(j), j) \in E^0$ à l'arbre et j à SV. Le sommet $i' = \text{ligne}(j) \in U$ devient alors la nouvelle candidate pour la prochaine itération.

Cette exécution peut se terminer, dans cette seconde phase, en deux situations possibles :

i. Si le sommet j sélectionné est non affecté, nous obtenons un chemin d'augmentation allant de k à j;

ii. Si V ne contient pas de sommet étiqueté et non numérisé ($LV \setminus \mathrm{SV} = \emptyset$), l'arbre actuel ne peut être agrandit.

Remarque. Chaque itération de la boucle principale exige une complexité $\mathcal{O}(n)$. A chaque itération, un sommet différent $j \in LV \setminus SV$ est sélectionné et ajouté à SV donc un sommet différent $i = \text{ligne}(j)$ est considéré. il s'en suit que la complexité de la procédure *alternate* est $\mathcal{O}(n^2)$.

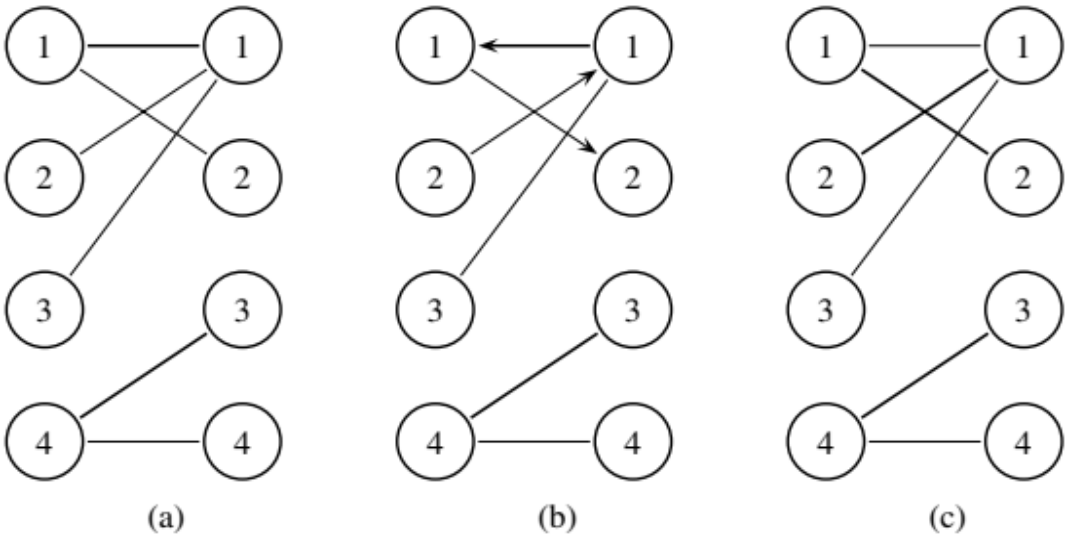


Figure 2.1. (a) Graphe G^0; (b) arbre alterné; (c) nouveau graphe G^0

Exemple 2.15. Nous continuons avec l'exemple 2.13. Partons de la matrice de coût réduit $\bar{C}$ et des vecteurs u, v, et *ligne* que nous avons obtenu et supposons que $k = 2$. La Figure 2.1(a) montre le graphe biparti partiel $G = (U, V; E^0)$, avec les lignes épaisse désignant l'affectation partielle actuelle. la procédure *alternate*(2) produit :
$SU = LV = SV = \emptyset$, $fail = false$, $sink = 0$;
$i = 2$: $SU = \{2\}$, $pred_1 = 2$, $LV = \{1\}$;
$j = 1$: $SV = \{1\}$;
$i = 1$: $SU = \{2, 1\}$, $pred_2 = 1$, $LV = \{1, 2\}$;
$j = 2$: $SV = \{1, 2\}$, $sink = 2$
Nous avons donc obtenu un arbre d'augmentation qui est montré dans la Figure 2.1(b).
La nouvelle solution est : $x_{12} = x_{21} = x_{43} = 1$ ($x_{ij} = 0$ ailleurs) (voir Figure 2.1(c)).

L'algorithme *Hongrois* est donné dans l'Algorithme 2.3. Soit (u, v) le couple de solutions duales faisables actuelles. L'affectation actuelle est rangée dans les vecteurs *ligne* et φ. Ils peuvent être initialisé à travers un phase de pré-traitement tel que, par exemple, la Procédure de **Basic_preprocessing**. nous désignons par $\bar{U} \subseteq U$ l'ensemble des sommets affectées de U.

Algorithme 2.3

Hongrois.
initialiser $u, v,$ ligne, φ et $\bar{U}$
while $|\bar{U}| < n$ **do:**
  Soit $k \in U \setminus \bar{U}$
  while $k \notin \bar{U}$ **do:**
    $sink = \text{alternate}(k)$
    if $sink > 0$ **then:**
      $\bar{U} = \bar{U} \cup \{k\}$
      repeat:
        $i = \text{pred}_j,\ \text{ligne}(j) = i, \quad h = \varphi(i), \quad \varphi(i) = j, \quad j = h$
      until $i = k$
    else:
      $\delta = \min(c_{ij} - u_i - v_j: \quad i \in \text{SU}, \quad j \in V \setminus \text{LV})$
      for each $i \in \text{SU}$ **do:** $u_i = u_i + \delta$
      for each $j \in \text{LV}$ **do:** $v_j = v_j - \delta$
    endif
  endwhile
endwhile

A chaque itération de la boucle extérieur, un sommet non affecté $k \in U$ est sélectionné à travers la boucle interne, c'est à dire, à travers une série d'appels de la procédure *alternate* suivi par une mise à jour des variables duales, jusqu'à ce qu'un chemin d'augmentation soit obtenu.

Chaque fois que *alternate* ne parvient pas à trouver un chemin d'augmentation, les variables duales correspondantes aux sommets étiquetés sont mises à jour. La mise a jour se fait comme suit :

- déterminer le coût réduit minimum δ des arêtes reliant les sommets étiquetés de U aux sommets non étiqueté de V;
- ajouter δ aux valeurs de u correspondant aux sommets étiquetés de U;
- soustraire δ aux valeurs de v correspondant aux sommets étiquetés de V.

La Proposition 2.16 montre qu'après cette mise à jour la prochaine exécution de *alternate* produira un arbre élargit.

Proposition 2.16. *La mise à jour du dual de la méthode Hongroise est tel que*

1. *Les arêtes de E^0 qui relient les paires de sommets étiquetés ou les paires de sommets non étiquetés maintiennent un coût réduit nul.*
2. *Au moins une nouvelle arête, connectant un sommet étiqueté de U à un sommet non étiqueté de V, entre dans E^0.*
3. *Il n'y a pas de coût réduit qui devient négatif.*

Démonstration. Rappelons que la numérisation et l'étiquetage coïncide pour le sommet U. Nous considérons les quatre ensembles d'arêtes (i, j) dont les coûts sont mises à jour de manière différent:

- $i \notin SU, j \notin LV$: aucune mise à jour ne se produit;

- $i \in SU, j \in LV$: la mise à jour a produit $\bar{c}_{ij} := \bar{c}_{ij} - \delta + \delta$, donc on est dans le premier cas ;
- $i \in SU, j \notin LV$: la mise à jour a produit $\bar{c}_{ij} := \bar{c}_{ij} - \delta$. Par définition de δ, le coût réduit résultant est non négative, et au moins un d'entre eux a la valeur nulle, et par conséquent, on est dans la deuxième situation ;
- $i \notin SU, j \in LV$: la mise à jour produit $\bar{c}_{ij} := \bar{c}_{ij} + \delta$. Ainsi aucun coût réduit ne peut être nulle. On se retrouve dans le dernier cas. □

Il s'en suit qu'à la prochaine exécution, *alternate* étiquettera au moins un sommet de V de plus, donc il en résultera un chemin d'augmentation après au plus n appels de *alternate*.

Proposition 2.17. *La complexité de l'algorithme Hongrois est donc* $\mathcal{O}(n^4)$

Démonstration. L'étape d'initialisation nécessite $\mathcal{O}(n^2)$ si elle est effectuée via la procédure Basic_preprocessing ou une méthode similaire. La boucle extérieure de l'algorithme Hongrois est exécutée en $\mathcal{O}(n)$. A chaque itération, la procédure *alternate* et la mise à jour des variables duales sont exécutées en $\mathcal{O}(n)$ dans la boucle intérieur. Nous devons déjà observer que chaque exécution de *alternate* trouve un arbre alternatif en $\mathcal{O}(n^2)$. La valeur de δ est aussi comptée en $\mathcal{O}(n^2)$. La complexité total de l'algorithme Hongrois est donc $\mathcal{O}(n^4)$. □

Exemple 2.18. Nous utilisons le problème introduit dans l'Exemple 2.13 et nous supposons que l'initialisation est effectuée via la procédure de Basic_preprocessing. Nous avons donc $u = (7, 2, 1, 2)$, $v = (0, 2, 0, 0)$ et ligne $= (1, 0, 4, 0)$; par conséquent, nous avons $\varphi = (1, 0, 0, 3)$ et $\bar{U} = \{1, 4\}$.

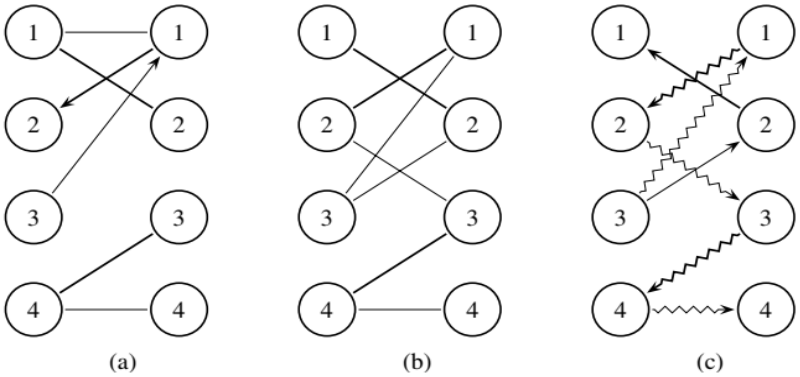
Comme $|\bar{U}| = n - 2$, la boucle extérieur sera exécutée à deux reprises. Le premier appel de la procédure *alternate(k)* avec $k = 2$ retourne $sink = 2$ et pred $= (2, 1, -, -)$; L'algorithme Hongrois augmente la solution primale. Après augmentation on obtient ligne $= (2, 1, 4, 0)$, $\varphi = (2, 1, 0, 3)$. On a maintenant $\bar{U} = \{1, 4, 2\}$. La procédure *alternate(k)* s'exécute alors pour $k = 3$.

$SU = LV = SV = \emptyset$, $fail = false$, $sink = 0$;
$i = 3$: $SU = \{3\}$, $pred_1 = 3$ $LV = \{1\}$;
$j = 1$: $SV = \{1\}$;
$i = 2$: $SU = \{3, 2\}$, $fail = true$
La solution dual est mise à jour. On obtient $\delta = 3$, $u = (7, 5, 4, 2)$, $v = (-3, 2, 0, 0)$, et

$$\bar{C} = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 5 \\ 4 & 2 & 0 & 2 \end{pmatrix}$$

La Figure 2.2(b) montre le nouveau graphe biparti partiel $G^0 = (U, V; E^0)$, avec les lignes épais désignant l'affectation partielle actuelle. La procédure *alternate(3)* est encore exécutée, produisant l'arbre augmenté montré par les flèches dans la figure 2.2(c), où les lignes en zig-zag montrent un chemin augmentation. La solution primale est augmentée produisant $\bar{U} = \{1, 4, 2, 3\}$, ligne $= (3, 1, 2, 4)$, $\varphi = (2, 3, 1, 4)$.

Nous avons ainsi obtenu une solution optimale, de valeur 17, définie par $x_{12} = x_{23} = x_{31} = x_{44} = 1$ (et $x_{ij} = 0$ ailleurs).



(a) arbre alternatif; (b) nouveau graphe G^0; (c) arbre d'augmentation

Figure 2.2.

2.3.3 Implémentation en $\mathcal{O}(n^3)$ de l'algorithme Hongrois

Nous avons montré, dans la Proposition 2.16, qu'après une mise à jour du dual, l'arbre alternatif enraciné en k précédent appartient toujours au nouveau graphe $G^0 = (U, V; E^0)$. Cette observation conduit à une implémentation (développée par Lawler dans les années 1970) qui fait croître l'arbre actuel sans réinitialiser les ensembles de sommets numérisés et étiquetés (comme cela est fait à chaque appel de *Alternate*). La procédure donnée dans l'Algorithme 2.4 itère les extensions d'arbre et les mises à jour des solutions duales jusqu'à ce qu'un chemin d'augmentation émanant du sommet d'entrée k soit obtenu. La signification de toutes les variables est la même que précédemment. De plus, nous utilisons $\pi_j\,(j \in V)$ pour désigner le coût réduit minimal d'une arête reliant un sommet étiqueté $i \in U$ à j. Notez que

i. A chaque itération, le sommet actuel $i \in SU$ est affecté comme prédécesseur au sommet $j \in V \setminus LV$ chaque fois que le coût réduit de (i, j) (même s'il est supérieur à zéro) améliore la valeur actuelle de π_j. Cependant, j n'est ajouté à LV que si ce coût réduit est nul;

ii. Après chaque mise à jour du dual, la procédure ajoute dans $L\,V$ tous les sommets non étiquetés de V pour lequel une nouvelle arête incidente ayant un coût réduit égal à zéro a été obtenue.

Algorithme 2.4

Augment(k)
Trouver un arbre augmentant enraciné à un sommet k non attribué
for each $j \in V$ **do:** $\pi_j = +\infty$
SU=SV=LV=$\emptyset$
$sink = 0, \quad i = k$
while $sink = 0$ **do:**
  SU = SU $\cup \{i\}$
  for each $j \in V \setminus$ LV: $\quad c_{ij} - u_i - v_j < \pi_j$ **do:**
    pred$_j = i$

$\pi_j = c_{ij} - u_i - v_j$
if $\pi_j = 0$ **then** $\mathrm{LV} = \mathrm{LV} \cup \{j\}$
endfor
if $\mathrm{LV} \setminus \mathrm{SV} = \emptyset$ **then:**
mettre à jour les variables dual
$\delta = \min(\pi_j : \quad j \in V \setminus \mathrm{LV})$
for each $i \in \mathrm{SU}$ **do:** $u_i = u_i + \delta$
for each $j \in \mathrm{LV}$ **do:** $v_j = v_j - \delta$
for each $j \in V \setminus \mathrm{LV}$ **do:**
$\pi_j = \pi_j - \delta$
if $\pi_j = 0$ **then:** $\mathrm{LV} = \mathrm{LV} \cup \{j\}$
endfor
endif
Soit $j \in \mathrm{LV} \setminus \mathrm{SV}$
$\mathrm{SV} = \mathrm{SV} \cup \{j\}$
if $\mathrm{ligne}(j) = 0$ **then:**
$sink = j$
else:
$i = \mathrm{ligne}(j)$
endif
endwhile
return $sink$

L'algorithme Hongrois se réduit alors à des exécutions itératives d'*augment(k)*, chacune suivie d'une augmentation de la solution primal, comme le montre l'Algorithme 2.5.

Algorithme 2.5

Hongrois_3
initialiser $u, v, \mathrm{ligne}, \varphi$ et $\bar{U}$
while $|\bar{U}| < n$ **do:**
soit k un sommet de $U \setminus \bar{U}$
$sink = \mathrm{Augment}(k)$
$\bar{U} = \bar{U} \cup \{k\}$
repeat:
$i = \mathrm{pred}_j,\ \mathrm{ligne}(j) = i, \quad h = \varphi(i), \quad \varphi(i) = j, \quad j = h$
until $i = k$
endwhile

Exemple 2.19. Dès le départ, le problème numérique est développée dans les exemples 2.13, 2.15 et 2.18. L'initialisation produite par la procédure Basic_preprocessing est évidemment la même, et le premier appel de *Augment(2)* effectue essentiellement les mêmes opérations que *Alternate (2)*, produisant ainsi la solution $x_{12} = x_{21} = x_{43} = 1$ (et $x_{ij} = 0$ ailleurs) et $\bar{U} = \{1, 2, 4\}, \mathrm{ligne} = (2, 1, 4, 0), \varphi = (2, 1, 0, 3)$. La procédure *Augment(3)* est alors exécutée. Les deux premières itérations sont très similaires aux itérations effectuées par *Alternate (3)* et produisent les mêmes arbres alternatifs:

$\pi = (\infty, \infty, \infty, \infty), SU = LV = SV = \emptyset, sink = 0;$
$i = 3: SU = \{3\}, pred = (3, 3, 3, 3), \pi = (0, 3, 5, 8), LV = \{1\};$
$j = 1: SV = \{1\};$
$i = 2: SU = \{3, 2\}, pred = (3, 3, 2, 2), \pi = (0, 3, 3, 5).$
La première mise à jour du dual est la suivante:
$\delta = 3, u = (7, 5, 4, 2), v = (-3, 2, 0, 0)$ et $\pi = (0, 0, 0, 2)$, donc $LV = \{1, 2, 3\}$.
L'exécution de *augment* continue avec
$j = 2: SV = \{1, 2\};$
$i = 1: SU = \{3, 2, 1\};$
$j = 3: SV = \{1, 2, 3\};$
$i = 4: SU = \{3, 2, 1, 4\}, pred = (3, 3, 2, 4), \pi = (0, 0, 0, 0), LV = (1, 2, 3, 4);$
$j = 4: SV = \{1, 2, 3, 4\}, sink = 4.$
En retour, Hongroise_3 produit les mêmes les solutions optimales que Hongrois.

Remarque. La boucle principale de *Augment* est exécutée en $\mathcal{O}(n)$, car (de la même manière que pour *Alternate*) à chaque itération, un sommet différent $j \in LV \setminus SV$ est sélectionné et ajouté à SV. Chaque itération est donc effectuée pour un i différent et nécessite un temps $\mathcal{O}(n)$, puisque les valeurs π_j permettent de calculer δ en $\mathcal{O}(n)$. La complexité de la procédure *Augmente(k)* est donc $\mathcal{O}(n^2)$. La boucle principale de *Hongrois_3* est exécutée en $\mathcal{O}(n)$. Par conséquent, l'algorithme a une complexité globale $\mathcal{O}(n^3)$.

Chapitre 3
Résolution du problème $\mathcal{P}$

3.1 Introduction

Dans le chapitre précédent, nous avons étudié le problème de somme linéaire d'affectations (PSLA) qui est un cas particulier de notre problème $\mathcal{P}$. Dans cette étude nous avons présenté et implémenté l'algorithme Hongrois qui est la première méthode de résolution en un temps polynomial du PSLA. Dans ce chapitre, nous allons adapter l'implémentation en $\mathcal{O}(n^3)$ de l'algorithme Hongrois pour résoudre notre problème d'optimisation $\mathcal{P}_c$.

3.2 Généralisation de l'algorithme Hongrois

Le coeur de l'algorithme Hongrois est, partant d'un couple de solutions duales, de parcourir les sommets qui respectent la condition de relâchement complémentaire pour effectuer les affectations à travers des chemins d'augmentations. Ces affectations ce font dans le respect des contraintes du PSLA, i.e, chaque lot de L est affecté à un unique candidat de $\mathcal{C}$ et vice versa. Notre algorithmes maintient les mêmes procédés que l'algorithmes Hongrois à la seule différence que dans le notre un candidat $C_j \in \mathcal{C}$ peut être utilisé pour affecter au plus r lots de L.

Soit $U = \{1, ..., p\}$ l'ensemble des indices de positions des lots de L et $V = \{1, ..., n\}$ l'ensemble des indices de positions des candidats de $\mathcal{C}$. Désignons par $K_{p,n} = (U, V; E)$ le Graphe de la correspondance Γ (graphe biparti au sens de la théories des graphes) où chaque correspondance entre un candidat C_j et un lot L_i est représentée par une arête $(i, j) \in E$.

3.2.1 Phase de pré-traitement

La phase de pré-traitement permet de déterminer une solution dual faisable et une solution primal partielle (où moins de p lots sont affectés) satisfaisant la condition de relâchement complémentaire. Une implémentation basique de complexité $\mathcal{O}(np)$ de cette phase est donnée dans l'algorithme suivant, qui stocke les affectations partielles dans

$$\varphi(i) = \begin{cases} j & \text{si } i \text{ est affecté à } j \\ 0 & \text{sinon} \end{cases}$$

et étiquette les sommets de V affectés dans

$$\text{ligne}(j) = \begin{cases} \{i : j \text{ affecté à } i\} \\ \\ \emptyset \quad \text{ailleurs} \end{cases}$$

Ici, ligne(j), ne stock pas une valeur, comme c'est le cas pour les PSLA, mais plutôt un ensemble de valeurs représentant les sommets de U affectés au sommet j.

Algorithme 3.1

Procédure Basic_preprocessing_r
initialiser φ, et ligne
for $i=1$ to p **do:** $\varphi(i)=0$
for $j=1$ to n **do:** ligne$(j)=\emptyset$
déterminer les variables dual
for $i=1$ to p **do:** $u_i=\min\{c_{ij}: j\in\{1,...,n\}\}$
for $j=1$ to n **do:** $v_j=\min\{c_{ij}-u_i: i\in\{1,...,p\}\}$
trouver une solution partielle
for $i=1$ to p **do:**
  for $j=1$ to n **do:**
    if ligne$(j)<r$ **and** $c_{ij}-u_i-v_j=0$ **then:**
      $\varphi(i)=j$
      ligne$(j)=$ligne$(j)\cup\{i\}$
      break

La procédure fonctionne de la manière suivante:

→ Elle construit les solutions duales. Le processus de construction des solutions duales est similaire à celui de l'algorithme Hongrois.

→ Elle parcourt les éléments de la matrice. Si un élément à un coût réduit $\bar{c}_{ij}$ nul et j à moins de r affectations, alors elle affecte i à j et ajoute i à ligne(j).

3.2.2 Adaptation de la procédure Augment de l'implémentation en $\mathcal{O}(n^3)$ de l'algorithme hongrois

Tout comme la procédure "Augment" de l'algorithme Hongrois, notre algorithme cherche un chemin d'augmentation partant d'un sommet non affecté $k \in U$ en construisant progressivement un arbre alternatif enraciné en k.

Algorithme 3.2

Procédure augment_r(k)
Trouver un arbre augmentant enraciné à un sommet k non attribué
for each $j\in V$ **do:** $\pi_j=+\infty$
SU=SV=LV=$\emptyset$
$sink=0,\ i=k$
while sink=0 **do:**
  SU$=$SU$\cup\{i\}$
  for each $j\in V\setminus$LV: $c_{i,j}-u_i-v_j<\pi_j$ **do:**

$\text{pred}_j = i$
$\pi_j = c_{i,j} - u_i - v_j$
if $\pi_j = 0$ **then** $\text{LV} = \text{LV} \cup \{j\}$
endfor
if $\text{LV} \setminus \text{SV} = \emptyset$ **then:**
mettre à jour les variables dual
$\delta = \min(\pi_j : j \in V \setminus \text{LV})$
for each $i \in \text{SU}$ **do:** $u_i = u_i + \delta$
for each $j \in \text{LV}$ **do:** $v_j = v_j - \delta$
for each $j \in V \setminus \text{LV}$ **do:**
$\pi_j = \pi_j - \delta$
if $\pi_j = 0$ **then:** $\text{LV} = \text{LV} \cup \{j\}$
endfor
endif
Soit $j_0 \in V \setminus \text{LV}$ tel que c_{i,j_0} minimale
$\text{SV} = \text{SV} \cup \{j_0\}$
if $\text{ligne}(j_0) < r$:
$sink = j_0$
else:
soit $i' \in \text{ligne}(j_0)$ tel que c_{i',j_0} maximal
$i = i'$
endif
endwhile
return $sink$

Ce algorithme ce déroule en deux phases.

→ Dans la première phase, notre algorithme effectue les mêmes opérations que celle de la procédure *augment* de l'algorithme Hongrois.

→ La deuxième phase est subdivisée en deux partie.

- le sommet étiqueté et non numérisé $j_0 \in V$ qui coïncide avec l'arête (i, j_0) au coût c_{i,j_0} minimal est sélectionnée et numérisée. La numérisation consiste à ajouter à l'arbre l'arête $(i', j_0) \in E^0$ qui a le coût maximal parmi les arêtes qui ont leur extrémité dans $\text{ligne}(j_0)$. Le sommet i' devient alors la nouvelle candidate pour la prochaine itération.
- S'il n'existe pas de sommets étiquetés et non numérisés dans V, i.e $\text{LV} \setminus \text{SV} = \emptyset$, alors la procédure met à jour les variables duales des sommets étiquetés et ajoute dans LV les nouveaux sommets de V qui ont coût réduit nul.

Soit (u, v) la solution dual faisable actuelle. Désignons par $\bar{U}$ l'ensemble des sommets de U affectés.

Algorithme 3.3

Hongrois_r
initialiser $u, v, \text{ligne}, \varphi$
identifier les sommets affectés

$\bar{U} = \{i \in U \text{ tel que } \varphi(i) \neq 0\}$
affecter les sommets de U non affectés
while $|\bar{U}| < p$ **do:**
soit $k \in U \setminus \bar{U}$
$sink = \text{Augment\_r}(k)$
$\bar{U} = \bar{U} \cup \{k\}$
repeat:
$i = \text{pred}_j$
désaffecter i à $\varphi(i)$
$\text{ligne}(\varphi(i)) = \text{ligne}(\varphi(i)) - \{i\}$
$\text{ligne}(j) = \text{ligne}(j) \cup \{i\}$,
$h = \varphi(i), \varphi(i) = j, j = h$
until $i = k$
endwhile
rendre la solution optimale
identifier les colonnes saturées
colonneSature=$\emptyset$
for $j \in V$ **do:**
if $|\text{ligne}(j)| = r$ **then:**
colonneSature=colonneSature $\cup \{j\}$
endif
endfor
$U_{\text{opt}} = \{i \in U$: $c_{i\varphi(i)} = \min \{c_{ij} : j \in V\}$ ou $c_{i\varphi(i)} = \min \{c_{ij} : j \in V \setminus \text{colonneSature}\}\}$
for $i \in U \setminus U_{\text{opt}}$ **do:**
Soit $j \in V \setminus \text{colonneSature}$ tel que c_{ij} minimal
if $c_{ij} < c_{i\varphi(i)}$ **then:**
$\text{ligne}(\varphi(i)) = \text{ligne}(\varphi(i)) - \{i\}$; **commentaire:** désaffecter i à $\varphi(i)$
commentaire: affecter i à j
$\varphi(i) = j$
$\text{ligne}(j) = \text{ligne}(j) \cup \{i\}$
if $\text{ligne}(j) = r$ **then:**
colonneSature=colonneSature $\cup \{j\}$
endif
endif
endfor

L'algorithme est subdivisé en trois phases que sont :

La phase de prétraitement.

La phase d'affectation. Elle se réduit à des exécutions itératives d'*augment_r*, chacune suivie d'une augmentation de la solution primale.

La phase d'optimisation. On dit qu'une affectation $\varphi(i)$ est optimale si $c_{i\varphi(i)} = \min \{c_{ij} : j \in V\}$ ou $c_{i\varphi(i)} = \min \{c_{ij}$: $j \in V \setminus \text{colonneSature}\}$. Une sommet $j \in V$ est dit saturé si $\text{ligne}(j) = r$. A chaque itération, il effectue les opérations suivantes :

- sélectionner un sommet i non optimale,

– chercher le coût minimal parmi les sommets non saturés,
– Affecter i au sommet j qui coïncide avec ce coût.

Proposition 3.1. *La complexité de notre algorithme est* $\mathcal{O}(np^2)$

Démonstration. La première phase de notre algorithme est exécutée en $\mathcal{O}(np^2)$. Dans la deuxième phase, la recherche des sommets saturés se fait en $\mathcal{O}(n)$. Celle des affectations optimales se fait en $\mathcal{O}(p^2)$. Au sortir de la première phase nous aurons au plus p sommets de U non optimals et pour chaque sommet la recherche de l'optimum nécessite au pire $\mathcal{O}(np)$ opérations. Donc, notre algorithme a une complexité de $\mathcal{O}(np^2)$. □

Exemple 3.2. Étant donnée la matrice d'entrée C ci dessous.

$$\begin{array}{c} \\ 7 \\ 2 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 2 & 0 & 0 \end{array} \\ \begin{pmatrix} 7 & 9 & 8 & 9 \\ 2 & 8 & 5 & 7 \\ 1 & 6 & 6 & 9 \\ 3 & 6 & 2 & 2 \end{pmatrix} \\ C \end{array} \qquad \begin{array}{c} \begin{pmatrix} \underline{0} & 0 & 1 & 2 \\ \underline{0} & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & \underline{0} & 0 \end{pmatrix} \\ \bar{C} \end{array}$$

En faisant appel à la procédure **Basic_preprocessing_r** pour $r=2$, on obtient les variables duales u et v (montrés a gauche et au dessus), l'ensemble *ligne*=($\{1,2\}$, $\emptyset, 4, \emptyset$) (donc $\varphi=(1,1,0,3)$) et l'affectation partielle montré par les zéros soulignés dans $\bar{C}$.

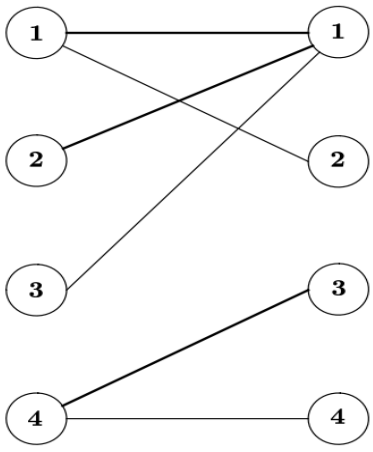


Figure 3.1.

Alors on a $\bar{U}=(1,2,4), \quad |\bar{U}|=3$.
on fait donc appel à augment_r(3).
$\pi=(+\infty,+\infty,+\infty,+\infty)$
$i=3, \mathrm{SU}=\{3\}, \mathrm{LV}=\{1\}, \mathrm{pred}=(3,3,3,3),\ \pi=(0,3,5,8)$
$j=1, \mathrm{SV}=\{1\}$
puisque ligne(1) à déjà deux lignes affectées qui sont 1 et 2 et que $c_{12}<c_{11}$, $i=1$ devient la nouvelle candidate.
$i=1, \mathrm{SU}=\{1,3\}, \mathrm{LV}=\{1,2\}, \mathrm{pred}=(3,1,1,1),\ \pi=(0,0,1,2)$

$j = 2, \mathrm{SV} = \{1, 2\}$
$sink = 2$
L'affectation donne $\varphi = (2, 1, 1, 3)$, $\bar{U} = (1, 2, 3, 4)$.

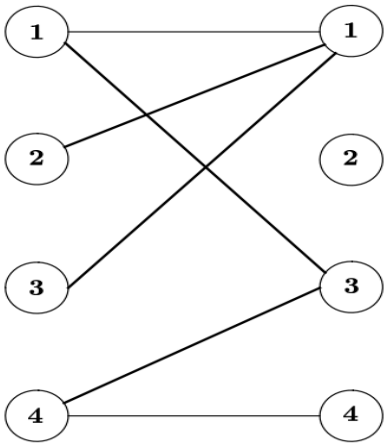


Figure 3.2.

Tous les sommets de U sont affectés. Maintenant, rendons la solution optimale.
colonneSaturer=$\{1\}$
$U_{\mathrm{opt}} = \{2, 3, 4\}$
$i = 1,\ j = 3,$ ligne$(3) < 2$
donc $\varphi(1) = 3$.
par conséquent, la solution optimale est $x_{13} = x_{21} = x_{31} = x_{43} = 1$. Les autres x sont nulles.

Épilogue

Dans ce memoire nous avons fait :

- des mathématiques
  - modélisation
  - optimisation combinatoire
  - théories des graphes
- de l'informatique
  - écriture d'algorithmes
  - analyse de complexités
  - implémentation d'algorithmes
  - simulations

pour proposer un début de solution à un problème complexe : la dématérialisation puis l'automatisation des procédures de passations des marchés publics.

Notre algorithme est une adaptation naïve du célèbre algorithme Hongrois qui résout un cas particulier de notre problème de départ. De plus, nous avons eu recours à une procédure d'optimisation pour corriger l'optimalité des affectations.

Ce premier pas est encourageant. Nous comptons poursuivre le travail dans cette direction en vu d'approfondir l'analyse mathématique de notre algorithme. Nous avons également l'ambition d'explorer en profondeur d'autres pistes évoquées dans ce mémoire, particulièremment celles passant par l'interpretation fonctionnelle du problème.

Annexe A
Implémentation des algorithmes

Pour l'implémentation de ces algorithmes nous utiliserons les bibliothèques **networkx** pour la manipulation des graphes et **numpy** pour la manipulation des matrices. Nous allons aussi utiliser la fonction **choice** et **deque** et les objets **set**, **list** et **dictionnaire**.

Choice est une fonction de la bibliothèque random qui sert à choisir de manière aléatoire un élément dans une liste d'éléments. Quant à **deque**, c'est une liste particulière qui permet l'ajoutet l'extraction des éléments dans les deux sens.

A.1 Implémentation des algorthimes de couplages maximums sur un graphe biparti

A.1.1 Algorithme cardinality matching

```
def Cardinality_matching(G,M,U,V):
    R=set()
    r={} #dictionnaire qui stocke les aretes ettiquétes dans
scan_leftvertex
    l={} ##dictionnaire qui stocke les aretes ettiquétes dans
scan_rightvertex
    # determiner l ensemble des sommets de U non couplés
(cote_gauche L)
    L=set(filter(lambda i:len({(i,j) for j in G[i]}&M)==0,U))
    # scanner caute gauche
    def scan_leftvertex(x,L,R,l):
        L=L-{x}
        # parcourir l ensemble des voisins non etiquete de x
        for j in set(filter(lambda u: u not in l,G[x])):
            l[j]=x
            R|={j}
        return L,R,l
```

```
    def scan_rightvertex(x,M,L,R,l,r):
        R=R-{x}
        p={}
        matched_neighbour=set(filter(lambda i:(i,x) in M,G[x]))
        if len(matched_neighbour)!=0:
            i=matched_neighbour.pop()
            r[i]=x
            L|={i}
        else:
            # recherche de chemin de type P := (. . . , r(l(x)),
l(x), x);
            p[x]=l[x]
            u=l.pop(x)
            test=True
            while len(l)!=0 and len(r)!=0 and test:
                try:
                    p[u]=r[u]
                    u=r.pop(u)
                    p[u]=l[u]
                    u=l.pop(u)
                except:
                    test=False
            # transformer le dictionnaire P en un ensemble
            P={(i,j) if i in U else (j,i) for i,j in p.items()}
            #faire la difference symetrique
            M=M^P
            # stoquer les sommets de U non couplé
            L=set(filter(lambda i:len({(i,j) for j in
G[i]}&M)==0,U))
            R=set()
            l={}
            r={}
        return L,R,l,r,M
    while len(L|R)!=0:
        x=choice(list(L|R))
        if x in L:
            L,R,l=scan_leftvertex(x,L,R,l)
        else:
            L,R,l,r,M=scan_rightvertex(x,M,L,R,l,r)
    return M
```

A.1.2 Algorithme de Hopcroft–Karp

```
def hopcroft_karp(G,M,U,V):
```

```
    # U_0 et _V_0 contient tous les sommets non couplés de et U et
V
    U_0=set(U)-{u for u,v in M}
    V_0=set(V)-{v for u,v in M}
    # fonction qui construit un graphe en couche
    def construire_graphe_couche():
        # cette fonction renvoyera:
        # - les couches du graphe sous forme de dictionnaire
        # - la longueur de la couche
        # - une booleen qui indique si le couplage est maximum ou
non
        couche={}
        couche[0]=set(U_0)
        k=0
        k_e=int(k)
        union_couche=set()
        while len(couche[k])!=0:
            couche[k+1]=set()
            for i in couche[k]:
                couche[k+1]=couche[k+1]|{j for j in G[i] if (i,j)
not in M and                                          j not in
union_couche }
            union_couche=union_couche|couche[k+1]
            if len(couche[k+1])==0:
                return couche, k_e, False
            if len(couche[k+1]&set(V_0))!=0:
                k_e=k+1
                couche[k+2]=set()
            else:
                couche[k+2]={i for i,j in M if j in couche[k+1]}
            k=k+2
        return couche,k_e, True
    # fonction qui cherche un systeme maximal de chemin
d'augmentation
    def trouver_delta_M():
        couche[k_e]=couche[k_e]&set(V_0)
        scan,matching=set(),{}
        k=1
        x,P={},{}
        while len(couche[0])!=0:
            x[0]=couche[0].pop()
            l=0
```

```
            while l>=0:
                try:
                    while len((set(G[x[l]])-scan)&couche[l+1])!=0:
                        v=(set(G[x[l]])-scan)&couche[l+1]
                        if l%2!=0:
                            u={i for i in v if (i,x[l]) in
M}.pop()
                        else:
                            u={j for j in v if (x[l],j) not in
M}.pop()
                        x[l+1]=u
                        scan.add(u)
                        l=l+1
                        if l==k_e:
                            P[k]={(x[i],x[i+1]) if x[i] in U
else (x[i+1],x[i])                                    for i in
range(k_e)}
                            k+=1
                except:
                    pass
                if l<k_e:
                    l-=1
                else:
                    l=-1
        return P
    # procedure de implementation de l algorithme de Hopcroft -
Karp
    test=True
    while len(U_0)!=0 and test:
        couche,k_e,test=construire_graphe_couche()
        if test:
            chemin=trouver_delta_M()
            for i in chemin:
                M^=chemin[i]
            U_0=set(U)-{u for u,v in M}
            V_0=set(V)-{v for u,v in M}
    return M
```

A.1.3 Illustration numérique de la complexité des deux méthodes

Maintenant illustrons numériquement la complexité des deux méthodes et comparons. Pour ce faire, nous allons générer aléatoirement 19 graphes en variant le nombre de sommets par pas de dix et sur chaque graphe nous allons prendre le temps de calcul de chaque méthode et les représenter.

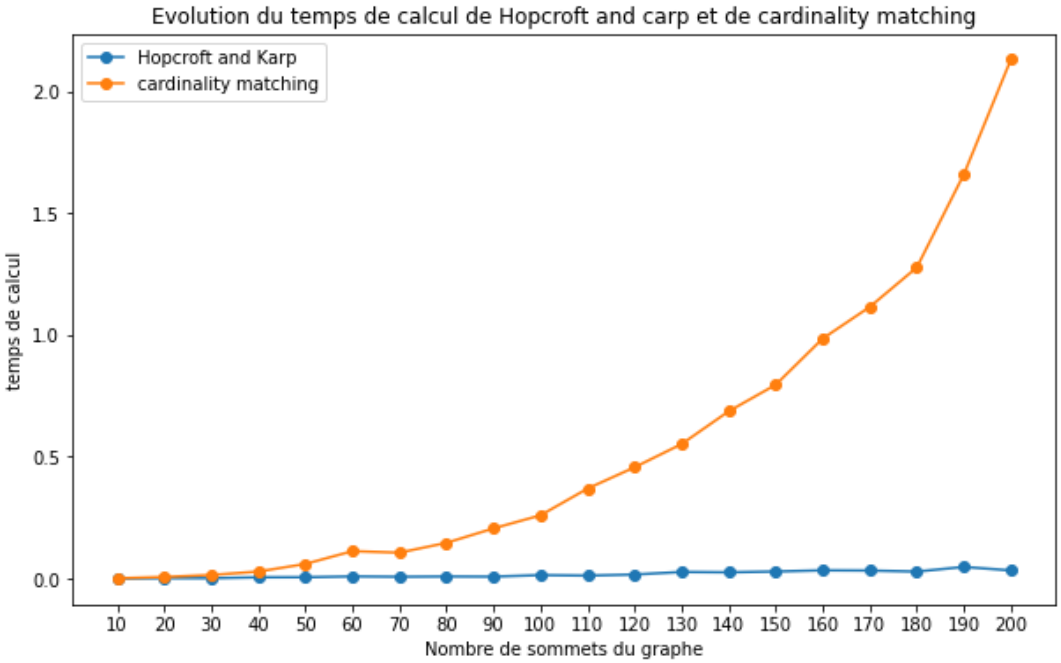


Figure A.1.

A.2 Implémentation de l'algorithme Hongrois

A.2.1 Implémentation basique l'algorithme Hongrois

```
def hongrois(C):
    n=C.shape[0]
    U={i+1 for i in range(n)}
    V=set(U)
    # processus de pretraitement
    def basic_preprocessing():
        # Ligne et colonne de reduction
        u=[min(j for j in i) for i in C]
        v=[min(j-u[n] for n,j in enumerate(i)) for i in C.T]

        # Trouver une solution faisable
        ligne={j:0 for j in V}
        phi={j:0 for j in U}

        for i in U:
            for j in V:
                if ligne[j]==0 and C[i-1,j-1]-u[i-1]-v[j-1]==0:
                    ligne[j]=i
                    phi[i]=j
                    break
```

```
    return ligne, phi,u,v

def alternate(k):
    # Initialisation
    SU,LV,SV=set(),set(),set()
    fail=False
    sink=0
    i=k
    pred={}

    while sink==0 and not fail:
        SU=SU|{i}

        for j in set(V)-LV:
            if C[i-1,j-1]-u[i-1]-v[j-1]==0:
                pred[j]=i
                LV=LV|{j}

        if len(set(LV)-SV)==0:
            fail=True
        else:
            j=set(set(LV)-SV).pop()
            SV=SV|{j}

            if ligne[j]==0:
                sink=j
            else:
                i=ligne[j]
    return sink,pred,SU,LV

# initialisation
ligne,phi,u,v=basic_preprocessing()
Ubar={i for i in phi if phi[i]!=0}

while len(Ubar)<n:
    k=(set(U)-Ubar).pop()
    while k not in Ubar:
        sink,pred,SU,LV=alternate(k)
        # augmenter la solution primale
        if sink>0:
            Ubar=Ubar|{k}
            j=sink
            i=None
            while i!=k:
                i=pred[j]
                ligne[j]=i
```

```
                    phi[i],j=j,phi[i]
            else:
                # mettre à jour la solution dual
                delta=min(C[i-1,j-1]-u[i-1]-v[j-1] for i in SU
                                    for j in set(V)-LV)
                for i in SU:
                    u[i-1]+=delta
                for j in LV:
                    v[j-1]-=delta
                print(u,v)
    return phi
```

Appliquons le code sur l'Exemple 2.13.

```
# initialisation avec basic_preprocessing
phi = [1, 0, 0, 3] ligne = [1, 0, 4, 0], u=[7, 2, 1, 2], v=[0, 2,
0, 0]
Ubar={1,4}

alternate(2)
SU =  {1, 2}, SV={1, 2}, LV={1, 2}
sink= 2, pred={1: 2, 2: 1}
augmenter la solution partielle
phi = [2, 1, 0, 3] ligne = [2, 1, 4, 0]
Ubar={1,2,4}

alternate(3)
SU =  {2, 3},SV={1}, LV={1}
mettre à jour les variables dual
u=[7, 5, 4, 2], v=[-3, 2, 0, 0]
Ubar={1,2,4}

alternate(3)
SU =  {1, 2, 3, 4}, SV={1, 2, 3, 4}, LV={1, 2, 3, 4}
sink= 4, pred={1: 3, 2: 3, 3: 2, 4: 4}
augmenter la solution partielle
phi = [2, 3, 1, 4] ligne = [3, 1, 2, 4]
```

A.2.2 Implémentation en $\mathcal{O}(n^3)$ de l'algorithme Hongrois

```
def hongrois_ameliorer(C):
    #
    n=C.shape[0]
    INF=float("inf")
    V={j+1 for j in range(n)}

    def procedure_preliminaire_basique():
```

```
            # Ligne et colonne de reduction
            u=[min(j for j in i) for i in C]
            v=[min(j-u[n] for n,j in enumerate(i)) for i in C.T]

            # Trouver une solution faisable
            ligne={j:0 for j in range(1,n+1)}
            phi={j:0 for j in range(1,n+1)}

            for i in range(n):
                for j in range(n):
                    if ligne[j+1]==0 and C[i,j]-u[i]-v[j]==0:
                        ligne[j+1]=i+1
                        break

            for j,i in ligne.items():
                if i!=0:
                    phi[i]=j
            return ligne, phi,u,v

        def procedure_augmente(k):
            # Initialisation
            pi={j+1:INF for j in range(n)}
            pred={}
            SU=set()
            SV=set()
            LV=set()
            sink=0
            i=k

            while sink==0:
                SU|={i}
                for j in {x for x in set(V)-LV if
C[i-1,x-1]-u[i-1]-v[x-1]<pi[x]}:
                    pred[j]=i
                    pi[j]=C[i-1,j-1]-u[i-1]-v[j-1]
                    if pi[j]==0:
                        LV|={j}

                if len(set(LV)-SV)==0:
                    delta=min(pi[j] for j in set(V)-LV)
                    for i in SU:
                        u[i-1]+=delta
                    for j in LV:
                        v[j-1]-=delta
                    print(u,v)
```

```
                for j in set(V)-LV:
                    pi[j]-=delta
                    if pi[j]==0:
                        LV|={j}

            j=set(set(LV)-SV).pop()
            SV|={j}

            if ligne[j]==0:
                sink=j
            else:
                i=ligne[j]
        return sink,pred

    # initialisation
    n=C.shape[0]
    U={i+1 for i in range(n)}
    ligne,phi,u,v=procedure_preliminaire_basique()
    Ubar={i for i,j in phi.items() if j!=0}

    while len(Ubar)<n:
        k=set(set(U)-Ubar).pop()
        sink,pred=procedure_augmente(k)
        Ubar|={k}
        j=sink
        # repeter ces insrtructions jusqua i=k
        i=None
        while i!=k:
            i=pred[j]
            ligne[j]=i
            phi[i],j=j,phi[i]

    return phi
```

A.3 Implémentation de l'algorithme de résolution du problème

```
def hongrois_r(C,r):

    # initialisation
    INF=float("inf")
    n=C.shape
    U={i+1 for i in range(n[0])}
    V={i+1 for i in range(n[1])}
```

```
        def Basic_preprocessing_r():
            # Ligne et colonne de reduction
            u=[min(j for j in i) for i in C]
            v=[min(j-u[n] for n,j in enumerate(i)) for i in C.T]
            # Trouver une solution faisable
            ligne={j:set() for j in V}
            phi={j:0 for j in U}
            for i in range(n[0]):
                for j in range(n[1]):
                    if len(ligne[j+1])<r and C[i,j]-u[i]-v[j]==0:
                        ligne[j+1].add(i+1)
                        phi[i+1]=j+1
                        break
            return ligne, phi,u,v

        def augment_r(k):
            # Initialisation
            pi={j+1:INF for j in range(n[1])}
            pred={}
            SU,LV,SV=set(),set(),set()
            sink=0
            i=k
            while sink==0:
                SU|={i}
                for j in {x for x in set(V)-LV if
C[i-1,x-1]-u[i-1]-v[x-1]<pi[x]}:
                    pred[j]=i
                    pi[j]=C[i-1,j-1]-u[i-1]-v[j-1]
                    if pi[j]==0:
                        LV|={j}
                if len(set(LV)-SV)==0:
                    delta=min(pi[j] for j in set(V)-LV)
                    for i in SU:
                        u[i-1]+=delta
                    for j in LV:
                        v[j-1]-=delta
                    print(u,v)
                    for j in set(V)-LV:
                        pi[j]-=delta
                        if pi[j]==0:
                            LV|={j}

                j_min={j for j in set(LV)-SV if
C[i-1,j-1]==min(C[i-1,h-1] for h in
                        set(LV)-SV)}
```

```
                if len({h for h in j_min if len(ligne[h ])<p})!=0:
                    j={h for h in j_min if len(ligne[h])<p}.pop()
                    sink=j
                    SV=SV|{j}
                else:
                    j=j_min.pop()
                    SV=SV|{j}
                    i={h for h in ligne[j] if
C[h-1,j-1]==max(C[i-1,j-1] for i in
                       ligne[j])}.pop()
            return sink,pred

        # initialisation
        ligne,phi,u,v=Basic_preprocessing_r()
        Ubar={i for i in phi if phi[i]!=0}
        while len(Ubar)<n[0]:
            k=set(set(U)-Ubar).pop()
            sink,pred=augment_r(k)
            Ubar|={k}
            j=sink
            # repeter ces insrtructions jusqu a i=k
            i=None
            while i!=k:
                i=pred[j]
                #desaffecter i à phi(i)
                if phi[i]!=0:
                    ligne[phi[i]].discard(i)
                #affecter i à j
                ligne[j].add(i)
                phi[i],j=j,phi[i]

        # rendre la solution optimale
        # colonne saturer
        sature=set()
        for j in ligne:
            if len(ligne[j])==r:
                sature.add(j)
        #sommet affecte de maniere optimale
        Uop={i for i in phi if C[i-1,phi[i]-1]==min(C[i-1,:]) or
             C[i-1,phi[i]-1]==min(C[i-1,j-1] for j in set(V)-sature)}

        for i in set(set(U)-Uop):
            j={h for h in set(V)-sature if C[i-1,h-1]==min(C[i-1,j-1]
for j in
               set(V)-sature)}.pop()
            if len(ligne[j])<r and C[i-1,j-1]<C[i-1,phi[i]-1]:
```

```
                #desaffecter i à phi[i] et affecter i à j
                ligne[phi[i]]-={i}
                #affecter i à j
                phi[i]=j
                ligne[j].add(i)
                if len(ligne[j])==p and max(C[i-1,j-1] for i in
                       ligne[j])<=min(C[i-1,j-1] for i in
set(U)-ligne[j]):
                    sature.add(j)

        return phi
```

Bibliographie

[1] CLAUDE BERGE. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America 43:842–844,1957. (Cited on p. 36.)*, 1957.

[2] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Revista Facultad de CienciasExactas, Puras y Aplicadas Universidad Nacional de Tucuman, Serie A(Matematicasy Fisica Teorica), 5:147–151, 1946. (Cited on pp. 25, 75.)*, 1946.

[3] Mauro Dell'Amico Burkard et Silvano Martello. Assignment problems. *Society for Industrial and Applied Mathematics*, 2009.

[4] Arthur Cayley. A theorem on trees. *The Quarterly Journal of Mathematics , 23:376–378, 1889. (Citedon p. 30.)*, 1889.

[5] G B Dantzig, L R Ford, et D.R. Fulkerson. *A primal-dual algorithm for linearprograms.* Princeton University Press,Princeton, NJ, 1956. (Cited on p. 79.), 1956.

[6] E.W. Dijkstra. A note on two problems in connection with graphs. *NumerischeMathematik, 1:269–271, 1959. (Cited on p. 80.)*, 1959.

[7] L.R. Ford et D.R. Fulkerson. Maximal flow through a network. *Canadian J. Math.,8:399–404, 1956. (Cited on p. 19.)*, 1956.

[8] F.G. Frobenius. Über zerlegbare determinanten. *Sitzungsberichte der KöniglichPreussischen Akademie der Wissenschaften zu Berlin, XVIII:274–277, 1917. (Citedon p. 16.)*, 1917.

[9] U S R Murty J.A. Bondy. *Graph theory*. Elsevier Science Publishing Co, 1976.

[10] Richard Mainning Karp John Hopcroft. An n5/2 algorithm for maximum matchings in bipartitegraphs. *SIAM J. Comput., 2:225–231, 1973. (Cited on pp. 3, 35, 42, 44, 88, 127.)*, 1973.

[11] Dénes Kőnig. Über graphen und ihre anwendungen. *Mathematische Annalen, 77:453–465,. (Cited on p. 16.)*, 1916.

[12] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Res. Log.Quart., 2:83–97. (Cited on pp. 77, 79, 128.)*, 1955.

[13] H.W. Kuhn. Variants of the hungarian method for the assignment problem. *NavalRes. Log. Quart., 3:253–258. (Cited on pp. 77, 79.)*, 1956.

[14] Andrew Russakoff Michel Balinski. On the assignment polytope. *SIAM Rev., 16:516–525, 1974. (Cited on pp. 30, 31, 31, 33, 34.)*, 1974.